DNS Cache Poisoning Attack

Federica Consoli - 1538420

April 24, 2018

Abstract

The goal of the assignment was to simulate a DNS Cache Poisoning attack. In this report I will discuss the steps taken to prepare for the hack, how it was implemented and what results were obtained.

1 Getting started

The idea behind this hack is to make the DNS accept a specially forged packet. By design, a DNS accepts a response packet only if it matches one of its pending queries. For this to happen, the following conditions need to be satisfied:

- the response arrives on the same UDP port it was sent from
- the response's query ID matches the one of the pending query
- the response's question section matches the one of the pending query

The first step is, of course, to obtain the query id and the source port from the DNS server.

In order to obtain these two information, I decided to use two sockets: one to query the DNS server and one to obtain its answer. The first sockets connects to the IP address of the DNS server on port 53, while the second one is listening on the same port, only this time on my machine.
I then crafted a query packet for the DNS, asking it to lookup the address for `badguy.ru` (which corresponds to my machine's address), and sent it to the DNS server via the first socket: within milliseconds, I received the information I needed on the second socket.

Using a similar approach, I queried the DNS server for the IP address and the hostname of the name server responsible for the victim's domain. This step

```
niccals@niccals-laptop:~/Desktop/DNS_Attack$ sudo python script.py
Listening on port 1337, waiting for secret...

QUERY ID: 26078 ; SOURCE PORT: 38561
The name server for ns.bankofallan.co.uk is at 10.0.0.1
```

Figure 1: First steps

is necessary because the source ip for the forged DNS responses needs to be spoofed, or else the response would not be accepted by the DNS, since it is not coming from an authoritative server for the targeted domain. Once I obtained the IP address, I added it as a static address for one of my machine's interfaces using the command `ip addr add 10.0.0.1 dev lo`, which later allowed me to perform the spoofing.

I then proceeded to configure the file `config.json` on my virtual machine as shown below. Finally, I connected the VM to my laptop using a bridged network adapter.

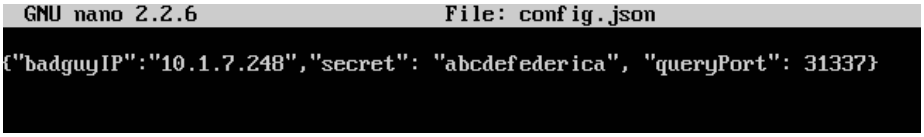

Figure 2: the VM configuration file

2 Implementation

The attack was implemented in Python, using the `dnslib` library, which allowed me to easily create and manipulate DNS packets. The program is launched via `sudo python bad_guy_IP target_IP`, where `bad_guy_IP` is the attacker's IP addres (namely, my own machine's), while `target_IP` represents the IP of the target (the VM).

After checking if the arguments are correct, the program spawns two concurring threads, with two different target function: `poison` and `listen`.

The `poison` function is the one that actually implements the attack. As mentioned before, this function uses two functions (`get_qid_and_port` and `get_ns`) to retrieve the information needed to carry out the attack, namely the query ID, the source port of the DNS, the IP and the hostname of the target's name server.

After this, the program creates a socket for UDP traffic and binds it to the IP address obtained from `get_ns` (which was previously added to my `lo` interface). It then queries the DNS for the address of the domain `random.bankofallan.co.uk` and starts flooding the server with response packets.

```
q_id = qid + i
query = DNSRecord(DNSHeader(qr=1,ra=1,id=q_id,aa=1), q=DNSQuestion("random.bankofallan.co.uk"))
answer = query.reply()
answer.add_answer(RR("random.bankofallan.co.uk",QTYPE.A,rdata=A(badguy_IP),ttl=4800))
answer.add_auth(RR("bankofallan.co.uk",QTYPE.NS,rdata=NS(ns_hostname),ttl=4800))
answer.add_ar(RR(ns_hostname,QTYPE.A,rdata=A(badguy_IP),ttl=4800))
sock_spoof.send(answer.pack())
```

Figure 3: Response structure

From the code snippet above, we can notice the following:

1. the *query ID* is incremented at each iteration, to try and match the query ID generated from the DNS server (which is randomly incremented)

2. the *answer* section contains the resolution for the requested domain, with an IP address of choice (in a real attack, this would be the address of the attacker's web server).

3. the *authority* section contains the hostname for the authoritative name server for `bankofallan.co.uk`, which was previously retrieved with the function `get_ns`. However, in the *additional record* section, the name server is linked to an IP chosen by the attacker: this means that after a successful poisoning, the attacker will become the owner of the whole zone.

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 21881
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 1, ADDITIONAL: 1
;; QUESTION SECTION:
;random.bankofallan.co.uk.      IN      A
;; ANSWER SECTION:
random.bankofallan.co.uk. 4800    IN      A       10.1.7.248
;; AUTHORITY SECTION:
bankofallan.co.uk.      4800    IN      NS      ns.bankofallan.co.uk.
;; ADDITIONAL SECTION:
ns.bankofallan.co.uk.   4800    IN      A       10.1.7.248
```

Figure 4: The forged response

The `listen` function, on the other hand, creates a socket listening on port 1337. This is where the *secret* is received, which indicates that the cache was successfully poisoned. When the secret is received, it is simply printed by the function.

3 The secret

As mentioned before, when I succeeded in carrying out the attack I received a *secret* on port 1337: what I received was the hash of the string I inserted in the field "secret" in the `config.json` file, as shown in Figure 2.