Lorenzo Cavada Mat. 220502

Third exercise of OffTech about DNS MitM Attack. A description of the exercise can be found here. All the code presented in this recap can istead be found at this link.

PART 1

1.1 What is the IP of the DNS server?

The IP of the cache DNS server is `10.1.1.3`.

1.2 What is the status of the request?

The status of the request is: `NOERROR`.

1.3 What is the IP address of `www.google.com`?

The IP address of `www.google.com` is `10.1.2.155`.

1.4 How long will the `www.google.com` IP be stored in the cache?

The IP of `www.google.com` will be stored in the cache for 10 seconds.

1.5 What is the authoritative name server for `google.com` and what is its IP?

The authoritative name server for `google.com.` is `ns.google.com.` and its IP is `10.1.2.3`.

PART 2

Understanding the topology

The info are obtained performing the following commands on each device:

```
arp -nn # to see the arp table of the device
ifconfig # to see the interface, ip and MAC address
```

Client network info:

```
IP: 10.1.1.2
MAC: a0:36:9f:0a:5d:83
```

Cache server network info:

```
IP: 10.1.1.3
MAC: a0:36:9f:0a:0e:1e
```

Auth server network info:

```
IP: 10.1.2.3
MAC: 00:25:90:67:97:57
```

Attacker network info:

```
IP: 10.1.2.4
MAC: a0:36:9f:0a:5c:3a
```

The packets does not pass through the attacker in any case. This can be see by performing on the attacker

```
sudo tcpdump -nnti ethX
```

Where `ethX` is the interface with which the attacker is connected to the cache server. By performing a dns request from the client is possible to notice as no packet is intercepted by the `tcpdump` command.

PART 3

Perform a MitM attack using ettercap

```
sudo ettercap -T -i ethX -M arp /10.1.2.2// /10.1.2.3//

-T is used to have a text only interface
-i is used to specify on which interface should ettercap look
-M to start a MitM attack
arp is the type of MitM attack
10.1.2.2 and 10.1.2.3 are the target.
```

Now, thanks to the MITM attack the traffic is passing through the attacker. That because the attacker has poisoned the arp table of the cache server and of the authorative server gaining a privilege position between the two. Now the attacker is acting as the auth server for the cache server and as the cache server for the auth server. All the traffic on the connection between the two server can be read, dropped or tampered by the attacker. The poisoning can be seen by performing on the cache server the following command:

```
arp -nn
```

The output will be something like the following:

```
Address                  HWtype  HWaddress           Flags Mask            Iface
10.1.2.3                 ether   a0:36:9f:0a:5c:3a   C                     eth5
192.168.1.254            ether   00:1b:21:cd:de:b1   C                     eth0
10.1.1.2                 ether   a0:36:9f:0a:5d:83   C                     eth4
10.1.2.4                 ether   a0:36:9f:0a:5c:3a   C                     eth5
```

Is possible to notice how the attacker (`10.1.2.4`) and the auth server (`10.1.2.3`) share the same MAC address which is the one of the attacker.

Is now possible to perform a DNS cache poisoning trickying the cache server to save as IP associated with `www.google.com` one choice by the attacker, let's assume the attacker's one (`10.1.2.4`). First of all is needed to update the file inside the attacker located at `/etc/ettercap/etter.dns` to include the malicious association between domain and IP, so in that file add:

```
www.google.com A 10.1.2.4
```

This will create the malicious association between the domain `www.google.com` and the IP of the attacker. Now by restarting `ettercap` is possible to use the plug-in `dns_spoof` (first start `ettercap`, then pres `p` and write `dns_spoof`). By performing a `dig` request from the client is possible to notice how the answer will associate the reqeusted domain with the IP of the attacker, meaning that the spoof is working as expected.

PART 4

Implementing DNSSEC

First of all is needed to configure the `auth` server. So connect throw ssh and perform the following step.

Add the following options for DNSsec in `/etc/bind/named.conf.options`, this are for enabling the dnssec feature.

```
dnssec-enable yes;
dnssec-validation yes; # requires manually-configured trust anchors using trusted-keys or managed-keys.
```

Now is time to generate the needed ZSK key. Create a directory called `keys` in `/etc/bind/` and, inside the just created folder type:

```
sudo dnssec-keygen -r /dev/urandom -a RSASHA256 -b 1024 -n ZONE google.com
```

This will create both the private and the public key.

Now is time to sign the domain `google.com` in `/etc/bind/`.

```
sudo dnssec-signzone -S -K /etc/bind/keys/ -P -g -a -o google.com google.com
```

This will generate a signed version of the dns record for `google.com` called `google.com.signed`.

The last step is to update the `named.conf.local` updating the zone `google.com` by specifing to use the `.signed` one.

Now restart bind.

```
sudo service bind9 restart
```

Configuring the Cache server

Now is time to update the cache server to start using DNSsec.

Add the following options for DNSsec in `/etc/bind/named.conf.options`, this are for enabling the dnssec feature.

```
dnssec-enable yes;
dnssec-validation yes;
```

Edit the file `bind.keys` for adding the ZSK key generated in the auth server. Is possible to see it by typing `cat Kgoogle...key` in the auth server. Remember to use the same structure as follow.

```
google.com. initial-key 256 3 8
"AwEAAbfjp9HRf9gR7DQqc5fQ5n6nek/cqoBgMoKJoMpeo8GubS20Ftt4
fGeKSm8QNSZ9qw91QFfyxj7biovH5d1X3CIVT3/KOB7Khuo5MwU/6WhT
wBnH2yp/HfHI/5gsK7JlE3HTVjOGb2VeGHZn0PXTj/eVIVV9JO+vAUTO mM2Cy6Yt";
```

Last step is to update the `named.conf` file including the `bind.keys` file just modified.

```
include "/etc/bind/bind.keys";
```

Now again restart the server.

```
sudo service bind9 restart
```

Is now possible to see how, if performing a `dig` request from the client with the `+dnssec` option added, the response will include a new `ad` flags, meaning that the request has been validated.