Remote DNS Cache Poisoning Attack Lab

Lab Overview

The objective of this lab is for students to gain first-hand experience on the remote DNS cache poisoning attack, also called the Kaminsky DNS attack. DNS (Domain Name System) is the Internet's phone book; it translates hostnames to IP addresses and vice versa.

This translation is through DNS resolution, which happens behind the scenes. DNS attacks manipulate this resolution process in various ways, with an intent to misdirect users to alternative destinations, which are often malicious. This lab focuses on a particular DNS attack technique, called DNS Cache Poisoning attack.

In another SEED Lab, we have designed activities to conduct the same attack in a local network environment, i.e., the attacker and the victim DNS server are on the same network, where packet sniffing is possible. In this remote attack lab, packet sniffing is not possible, so the attack becomes much more challenging than the local attack.

This lab covers the following topics:

- DNS and how it works
- DNS server setup
- DNS cache poisoning attack
- Spoofing DNS responses
- Packet spoofing

Lab Environment Setup

Please download the Labsetup.zip file from the link given below :
https://seedsecuritylabs.org/Labs_20.04/Networking/DNS/DNS_Remote/

Follow the instructions in the lab setup document to set up the lab environment.

The main target for DNS cache poisoning attacks is the local DNS server. Obviously, it is illegal to attack a real server, so we need to set up our own DNS server to conduct the attack experiments. The lab environment needs four separate machines: **one for the victim, one for the DNS server, and two for the attacker.**
The lab environment setup is illustrated in Figure 1. We put all these machines on the same LAN only for the sake of simplicity. Students are not allowed to exploit this fact in their attacks; they should treat the attacker machine as a remote machine, i.e., the attacker cannot sniff packets on the LAN. This is different from the local DNS attack.

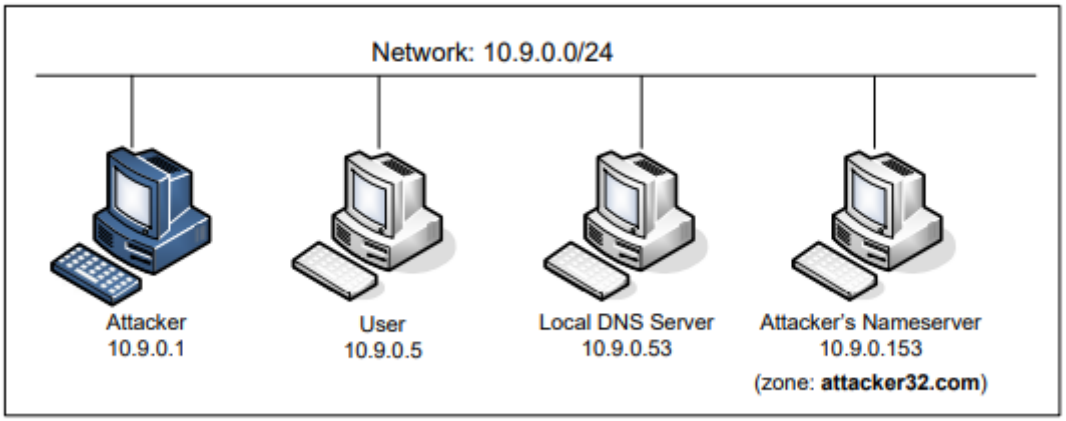


Figure 1 : Environment setup for the experiment

Verification of the DNS setup

From the **User container**, we will run a series of commands to ensure that our lab setup is correct. In your lab report, please document your testing results.

Get the IP address of ns.attacker32.com
When we run the following dig command, the local DNS server will forward the request to the Attacker name server due to the forward zone entry added to the local DNS server's configuration file. Therefore, the answer should come from the zone file (attacker32.com.zone) that we set up on the Attacker nameserver. If this is not what you get, your setup has issues.

Command :

 # dig ns.attacker32.com

Get the IP address of www.example.com
Two nameservers are now hosting the example.com domain, one is the domain's official nameserver, and the other is the Attacker container. We will query these two nameservers and see what response we will get. Please run the following two commands (from the User machine), and describe your observation.

Commands :

 # dig www.example.com
 # dig @ns.attacker32.com www.example.com

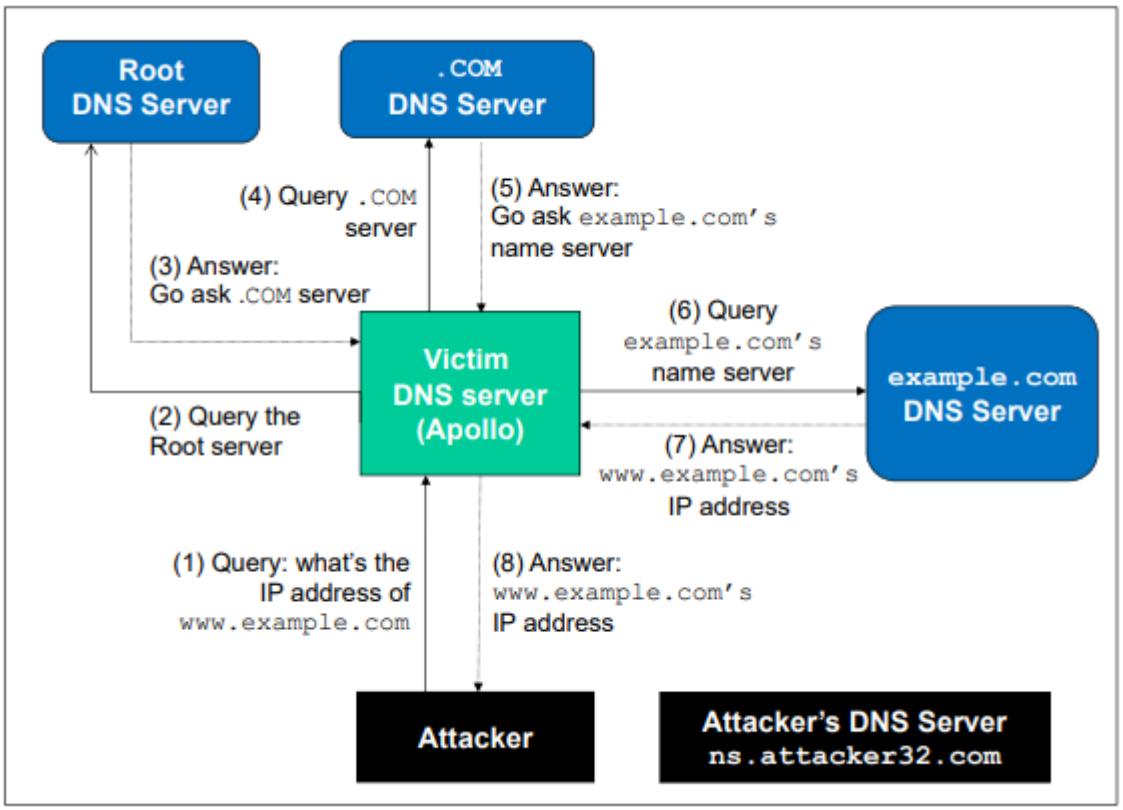
The Attack Tasks

The main objective of DNS attacks is to redirect the user to another machine B when the user tries to get to machine A using A's host name. For example, assuming www.example.com is an online banking site. When the user tries to access this site using the correct URL www.example.com, if the adversaries can redirect the user to a malicious web site that looks very much like www.example.com, the user might be fooled and give away his/her credentials to the attacker.

In this task, we use the domain name www.example.com as our attacking target. It should be noted that the example.com domain name is reserved for use in documentation, not for any real company. The authentic IP address of www.example.com is 93.184.216.34, and its nameserver is managed by the Internet Corporation for Assigned Names and Numbers (ICANN).

When the user runs the dig command on this name or types the name in the browser, the user's machine sends a DNS query to its local DNS server, which will eventually ask for the IP address from example.com's nameserver.

The goal of the attack is to launch the DNS cache poisoning attack on the local DNS server, such that when the user runs the dig command to find out www.example.com's IP address, the local DNS server will end up going to the attacker's name server ns.attacker32.com to get the IP address, so the IP address returned can be any number that is decided by the attacker. As a result, the user will be led to the attacker's web site, instead of to the authentic www.example.com.



Task overview

Implementing the Kaminsky attack is quite challenging, so we break it down into several sub-tasks. In Task 1, we construct the DNS request for a random hostname in the example.com domain. In Task 2, we construct a spoofed DNS reply from example.com's nameserver. In Task 3, we put everything together to launch the Kaminsky attack. Finally in Task 4, we verify the impact of the attack.

Task 1: Construct DNS request

This task focuses on sending out DNS requests. In order to complete the attack, attackers need to trigger the target DNS server to send out DNS queries, so they have a chance to spoof DNS replies. Since attackers need to try many times before they can succeed, it is better to automate the process using a program.

The students' job is to demonstrate that their queries can trigger the target DNS server to send out corresponding DNS queries. Show the response packets sent by the nameserver to the local DNS server. Also show the packet that triggers the local DNS server to query the domain's name server.

Before running the command keep wireshark open to view the packets being sent.

On the attacker terminal run the command:

 # python3 generate_dns_query.py

Task 2: Spoof DNS Replies

In this task, we need to spoof DNS replies in the Kaminsky attack. Since our target is example.com, we need to spoof the replies from this domain's nameserver.

We first find the IP addresses of the name servers of the example.com domain. This is done using the dig command as follows :

On the attacker terminal run the command:

 # dig NS example.com

 # dig +short a [example.com name server's name]

These IP addresses are used as the source IP addresses for the spoofed replies.

Since the reply being generated here by itself will not be able to lead to a successful attack, to demonstrate this task, students need to use Wireshark to capture the spoofed DNS replies, and show the contents of the spoofed packets and show that they are valid.

Before running the command keep wireshark open to view the packets being sent.

On the attacker terminal run the command:

 # python3 generate_dns_reply.py

Task 3: Launch the Kaminsky Attack

Now we can put everything together to conduct the Kaminsky attack. In the attack, we need to send out many spoofed DNS replies, hoping one of them hits the correct transaction number and arrives sooner than the legitimate replies.
Therefore, speed is essential: the more packets we can send out, the higher the success rate is. If we use Scapy to send spoofed DNS replies like what we did in the previous task, the success rate is too low.
We introduce a hybrid approach using both Scapy and C (see the SEED book for details). With the hybrid approach, we first use Scapy to generate a DNS packet template, which is stored in a file. We then load this template into a C program, and make small changes to some of the fields, and then send out the packet.

For this task, you should compile the C code inside the host VM, and then run the code inside the container. You can use the "docker cp" command to copy a file from the host VM to a container. See the following example (there is no need to type the docker ID in full):

On the Host VM run the following commands:

```
# gcc -o kaminsky attack.c
# docker ps
// Copy kaminsky executable to the seed-attacker container's /volumes folder
# docker cp kaminsky [Docker container ID]:/volumes
```

On the attacker terminal run the command:

```
# ./kaminsky
```

While the attack is running check if the cache has been poisoned and stop the attack appropriately.

Check the DNS cache
To check whether the attack is successful or not, we need to check the dump.db file to see whether our spoofed DNS response has been successfully accepted by the DNS server. The following commands dump the DNS cache, and search whether the cache contains the word attacker (In our attack, we used attacker32.com as the attacker's domain. If students use a different domain name, they should search for a different word).

On the local DNS server's terminal run the command :

```
# rndc dumpdb -cache && grep attacker /var/cache/bind/dump.db
```

Task 4: Result Verification

If the attack is successful, in the local DNS server's DNS cache, the NS record for example.com will become ns.attacker32.com. When this server receives a DNS query for any hostname inside the example.com domain, it will send a query to ns.attacker32.com, instead of sending to the domain's legitimate nameserver.

To verify whether your attack is successful or not, go to the User machine, run the following two dig commands. In the responses, the IP addresses for www.example.com should be the same for both commands, and it should be whatever you have included in the zone file on the Attacker nameserver.

Please include your observation (screenshots) in the lab report, and explain why you think your attack is successful. In particular, when you run the first dig commands, use Wireshark to capture the network traffic, and point out what packets are triggered by this dig command.

Keep wireshark open before running the commands.

On the victim terminal run the command:

 # dig www.example.com

 # dig @ns.attacker32.com www.example.com

Submission

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits.