Remote DNS Attack (Kaminsky Attack) Lab

Updated on February 26, 2020

1 Lab Overview

The objective of this lab is for students to gain the 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 scene. 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

Note: This lab was revised on February 26, 2020. The setup part has been modified significantly, and it is now part of lab tasks. As results, the task numbers have changed.

Readings. Detailed coverage of DNS and its attacks can be found in Chapter 18 of the SEED book, *Computer Internet Security: A Hands-on Approach, 2nd Edition*, by Wenliang Du.

Lab environment. This lab has been tested on our pre-built Ubuntu 16.04 VM, which can be downloaded from the SEED website.

Customization. In this lab description, we use the domain `attacker32.com` to refer to the domain controlled by the attacker. When students do this lab, they are not allowed to use this name; instead, they should use a domain name that includes their last names. The objective of this requirement is to differentiate student's work. Since the domain name is only visible inside the lab environment, not to the public, any name, including those already owned by others, can be used safely in this lab.

2 Lab Environment Setup Tasks

The main target for DNS cache poisoning attacks is 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 three separate machines: one for the victim, one for the DNS server, and the other for the attacker.

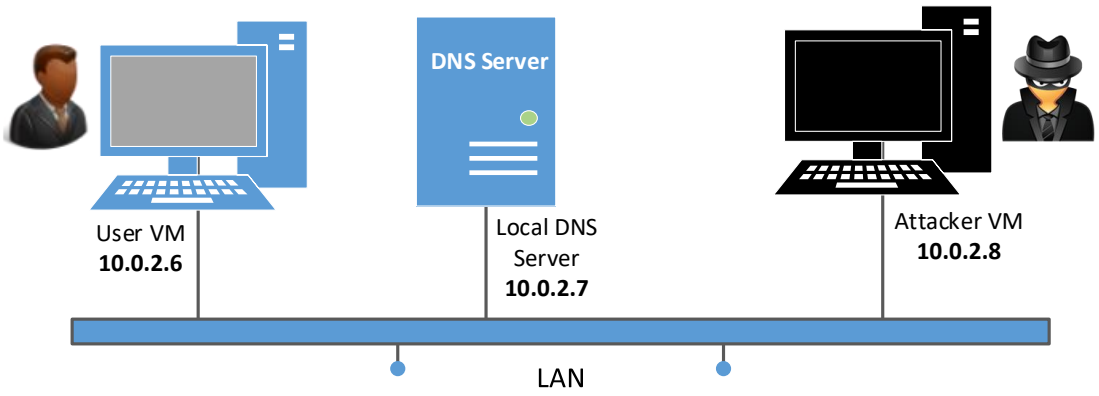

Figure 1: Environment setup for the experiment

We will run these three virtual machines on a single host machine. All these VMs will run our pre-built `Ubuntu` VM image. Figure 1 illustrates the setup of the experiment environment. For the VM network setting, if you are using `VirtualBox`, please use `"NAT Network"` as the network adapter for each VM. If you are using `Vmware`, the default `"NAT"` setting is good enough. We put all these VMs 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.

In the following sections, we assume that the user machine's IP address is `10.0.2.6`, the local DNS Server's IP is `10.0.2.7` and the attacker machine's IP is `10.0.2.8`. We call the Local DNS server `Apollo` in this document.

2.1 Task 1: Configure the User VM

On the user machine `10.0.2.6`, we need to use `10.0.2.7` as the local DNS server. This is achieved by changing the resolver configuration file (`/etc/resolv.conf`) of the user machine, so the server `10.0.2.7` is added as the first `nameserver` entry in the file, i.e., this server will be used as the primary DNS server. Unfortunately, our provided VM uses the Dynamic Host Configuration Protocol (DHCP) to obtain network configuration parameters, such as IP address, local DNS server, etc. DHCP clients will overwrite the `/etc/resolv.conf` file with the information provided by the DHCP server.

One way to get our information into `/etc/resolv.conf` without worrying about the DHCP is to add the following entry to the `/etc/resolvconf/resolv.conf.d/head` file (assuming that `10.0.2.7` is the IP address of the local DNS server):

```
nameserver 10.0.2.7
```

The content of the head file will be prepended to the dynamically generated resolver configuration file. Normally, this is just a comment line (the comment in `/etc/resolv.conf` comes from this head file). After making the change, we need to run the following command for the change to take effect:

```
$ sudo resolvconf -u
```

Testing: After you finish configuring the user machine, use the `dig` command to get an IP address from a hostname of your choice. From the response, please provide evidences to show that the response is indeed from your server. If you cannot find the evidence, your setup is not successful.

2.2 Task 2: Configure the Local DNS Server (the Server VM)

For the local DNS server, we need to run a DNS server program. The most widely used DNS server software is called BIND (Berkeley Internet Name Domain), which, as the name suggests, was originally designed at the University of California Berkeley in the early 1980s. The latest version of BIND is BIND 9, which was first released in 2000. We will show how to configure BIND 9 for our lab environment. The BIND 9 server program is already installed in our pre-built `Ubuntu` VM image and it is automatically started when the system boots up.

BIND 9 gets its configuration from a file called `/etc/bind/named.conf`. This file is the primary configuration file, and it usually contains several `"include"` entries, i.e., the actual configurations are stored in those included files. One of the included files is called `/etc/bind/named.conf.options`. This is where we typically set up the configuration options.

Step 1: Remove the `example.com` Zone. If you did our "Local DNS Attack Lab", you have probably configured the local DNS server `Apollo` to host the `example.com` domain. In this lab, this DNS server will not host that domain, so please remove its corresponding zone from `/etc/bind/named.conf`.

Step 2: Set up a forward zone. The main purpose of the Kaminsky attack in this lab is to get the victim to use `ns.attacker32.com` as the nameserver of the `example.com` domain. Once the attack succeeds, all future queries of the `example.com` domain on the victim DNS server will be sent to `ns.attacker32.com`.

In the real world, the local DNS server needs to find the IP address of `ns.attacker32.com` first. It will go through the root server, the `com` server, and eventually get a response from the actual nameserver that hosts the `attacker32.com` domain. Once the local DNS server gets the IP address, it will send the query to this IP address. That is where students will have problems, because students do not own this `attacker32.com` domain (it is actually owned by Wenliang Du, the author of this lab), so students will not be able to configure the DNS server running on `ns.attacker32.com`.

To solve this problem, each student can purchase his or her own domain, and then use the purchased name in the attack, instead of using `attacker32.com`. This way, students can configure their DNS servers to answer queries. The cost of this approach is too high for students.

Fortunately, BIND9 allows us to add a forward zone in the DNS configuration. Add the following zone entry to the `/etc/bind/named.conf` file. This entry indicates that for all queries of the `attacker32.com` domain, forward the queries to `10.0.2.8`. This is equivalent to using `10.0.2.8` as the nameserver for the `attacker32.com` domain. Therefore, with this entry, the local DNS server will not try to find the IP address of `attacker32.com`'s nameserver; it already has the IP address. Please do not forget to include all those semicolons, or the configuration will be invalid.

```
zone "attacker32.com" {
    type forward;
    forwarders {
        10.0.2.8;
    };
};
```

Step 3: Configure a few options. In our SEED VM, we have already performed all the configuration in this step. We just want to show what we have done; no action is needed from students if the SEED VM is used. The configuration is made in the `/etc/bind/named.conf.options` file.

- **Configure where to dump the DNS cache.** The following option specifies where the cache content should be dumped to if BIND is asked to dump its cache. If this option is not specified, BIND dumps the cache to a default file called `/var/cache/bind/named_dump.db`.

```
options {
    dump-file "/var/cache/bind/dump.db";
};
```

  The following two commands are related to DNS cache. The first command dumps the content of the cache to the file specified above, and the second command clears the cache.

```
$ sudo rndc dumpdb -cache    // Dump the cache to the sepcified file
$ sudo rndc flush            // Flush the DNS cache
```

- **Turn off DNSSEC.** DNSSEC is introduced to protect against spoofing attacks on DNS servers. To show how attacks work without this protection mechanism, we need to turn the protection off. This is done by modifying the `named.conf.options` file: comment out the `dnssec-validation` entry, and add a `dnssec-enable` entry.

```
options {
    # dnssec-validation auto;
    dnssec-enable no;
};
```

- **Fix the Source Ports.** DNS servers now randomize the source port number in their DNS queries; this makes the attacks much more difficult. Unfortunately, many DNS servers still use predictable source port number. For the sake of simplicity in this lab, we assume that the source port number is a fixed number. We can set the source port for all DNS queries to `33333`. This can be done by adding the following option to the file `/etc/bind/named.conf.options`:

```
 query-source port 33333
```

Step 4: Restart DNS server. We can now restart the DNS server using the following command. Every time a modification is made to the DNS configuration, the DNS server needs to be restarted. The following command will start or restart the `BIND 9` DNS server.

```
$ sudo service bind9 restart
```

2.3 Task 3: Configure the Attacker VM

On the Attacker VM, we will host two zones. One is the attacker's legitimate zone `attacker32.com`, and the other is the fake `example.com` zone.

- Step 1: Download the `attacker32.com.zone` and `example.com.zone` files from the lab's website.
- Step 2: Modify these files accordingly based on students' actual network setup (e.g., some IP addresses need to be changed).
- Step 3: Copy these two files to the `/etc/bind` folder.

- Step 4: Add the following entries to `/etc/bind/named.conf`:

```
zone "attacker32.com" {
        type master;
        file "/etc/bind/attacker32.com.zone";
};

zone "example.com" {
        type master;
        file "/etc/bind/example.com.zone";
};
```

- Step 5: Restart the DNS server.

2.4 Task 4: Testing the Setup

From the User VM, we will run a series of commands to ensure that our setup is correct.

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 VM due to the `forward` zone entry added to the local DNS server's configuration file. Therefore, the answer should come from the `attacker32.com.zone` file that we set up on the Attacker VM. If this is not what you get, your setup has an issue. Please describe your observation in your lab report.

```
$ 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 VM. We will query these two nameservers and see what response we will get. Please run the following two commands (from the User VM), and describe your observation.

```
// Send the query to our local DNS server, which will send the query
// to example.com's official nameserver.
$ dig www.example.com

// Send the query directly to ns.attacker32.com
$ dig @ns.attacker32.com www.example.com
```

Obviously, nobody is going to ask `ns.attacker32.com` for the IP address of `www.example.com`; they will always ask the `example.com` domain's official nameserver for answers. The objective of the DNS cache poisoning attack is to get the victims to ask `ns.attacker32.com` for the IP address of `www.example.com`. Namely, if our attack is successful, if we just run the first `dig` command, the one without the `@` option, we should get the fake result from the attacker, instead of getting the authentic one from the domain's legitimate nameserver.

3 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 nameserver `ns.attacker32.com` to get the IP address, so the IP address returned can be any number that is decided by the attacker. As results, the user will be led to the attacker's web site, instead of to the authentic `www.example.com`.

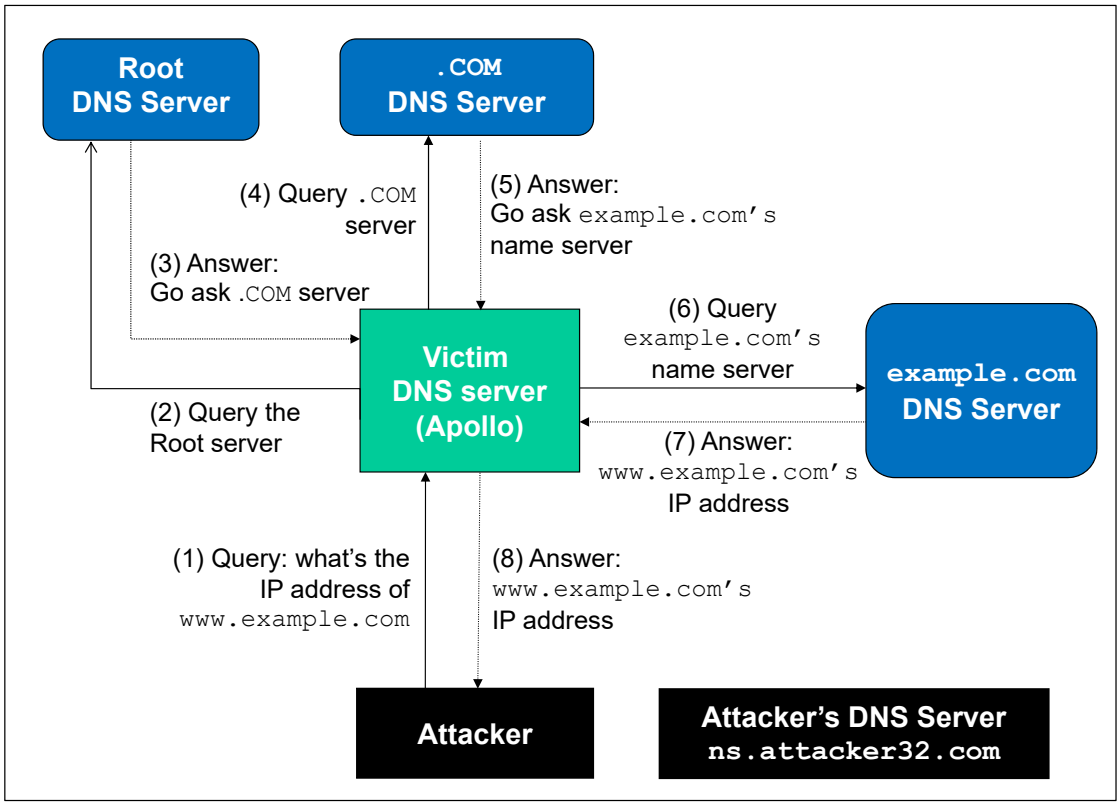

Figure 2: The complete DNS query process

3.1 How Kaminsky attack works

In this task, the attacker sends a DNS query request to the victim DNS server (`Apollo`), triggering a DNS query from `Apollo`. The query may go through one of the root DNS servers, the `.COM` DNS server, and the final result will come back from `example.com`'s DNS server. This is illustrated in Figure 2. In case that `example.com`'s nameserver information is already cached by `Apollo`, the query will not go through

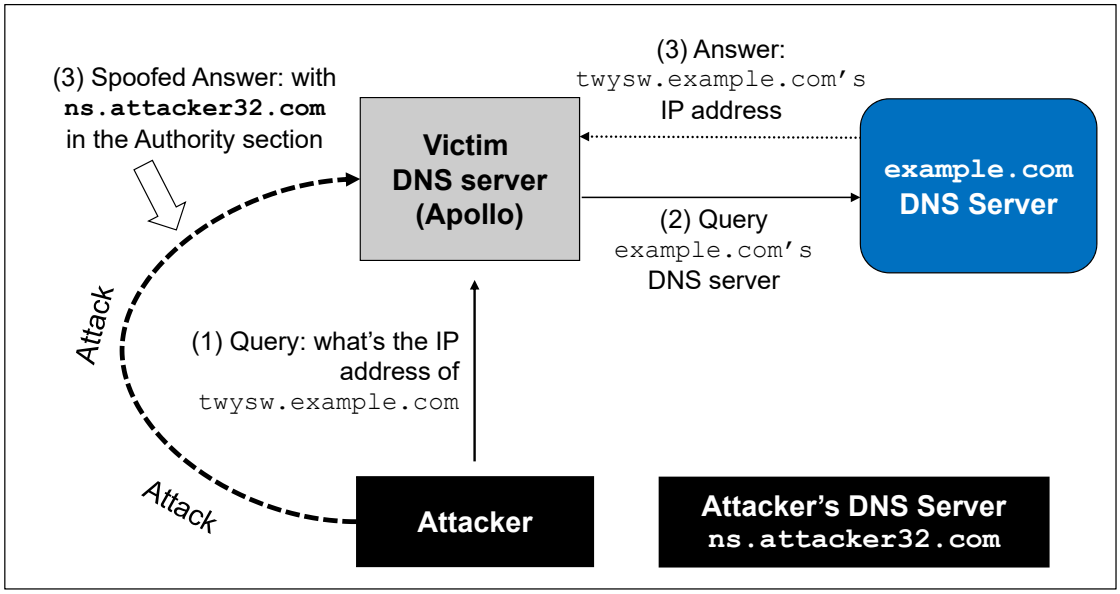

Figure 3: The Kaminsky Attack

the root or the `.COM` server; this is illustrated in Figure 3. In this lab, the situation depicted in Figure 3 is more common, so we will use this figure as the basis to describe the attack mechanism.

While `Apollo` waits for the DNS reply from `example.com`'s name server, the attacker can send forged replies to `Apollo`, pretending that the replies are from `example.com`'s nameserver. If the forged replies arrive first, it will be accepted by `Apollo`. The attack will be successful.

If you have done our local DNS attack lab, you should realize that those attacks assume that the attacker and the DNS server are on the same LAN, i.e., the attacker can observe the DNS query message. When the attacker and the DNS server are not on the same LAN, the cache poisoning attack becomes more difficult. The difficulty is mainly caused by the fact that the transaction ID in the DNS response packet must match with that in the query packet. Because the transaction ID in the query is usually randomly generated, without seeing the query packet, it is not easy for the attacker to know the correct ID.

Obviously, the attacker can guess the transaction ID. Since the size of the ID is only 16 bits, if the attacker can forge K responses within the attack window (i.e. before the legitimate response arrives), the probability of success is K over 2^{16}. Sending out hundreds of forged responses is not impractical, so it will not take too many tries before the attacker can succeed.

However, the above hypothetical attack has overlooked the cache effect. In reality, if the attacker is not fortunately enough to make a correct guess before the real response packet arrives, correct information will be cached by the DNS server for a while. This caching effect makes it impossible for the attacker to forge another response regarding the same name, because the DNS server will not send out another DNS query for this name before the cache times out. To forge another response on the same name, the attacker has to wait for another DNS query on this name, which means he/she has to wait for the cache to time out. The waiting period can be hours or days.

The Kaminsky Attack. Dan Kaminsky came up with an elegant technique to defeat the caching effect [2]. With the Kaminsky attack, attackers will be able to continuously attack a DNS server on a domain name, without the need for waiting, so attacks can succeed within a very short period of time. Details of the attacks are described in [1, 2]. In this task, we will try this attack method. The following steps with reference to

Figure 3 outlines the attack.

1. The attacker queries the DNS Server `Apollo` for a non-existing name in `example.com`, such as `twysw.example.com`, where `twysw` is a random name.

2. Since the mapping is unavailable in `Apollo`'s DNS cache, `Apollo` sends a DNS query to the name-server of the `example.com` domain.

3. While `Apollo` waits for the reply, the attacker floods `Apollo` with a stream of spoofed DNS response, each trying a different transaction ID, hoping one is correct. In the response, not only does the attacker provide an IP resolution for `twysw.example.com`, the attacker also provides an "Authoritative Nameservers" record, indicating `ns.attacker32.com` as the nameserver for the `example.com` domain. If the spoofed response beats the actual responses and the transaction ID matches with that in the query, `Apollo` will accept and cache the spoofed answer, and and thus `Apollo`'s DNS cache is poisoned.

4. Even if the spoofed DNS response fails (e.g. the transaction ID does not match or it comes too late), it does not matter, because the next time, the attacker will query a different name, so `Apollo` has to send out another query, giving the attack another chance to do the spoofing attack. This effectively defeats the caching effect.

5. If the attack succeeds, in `Apollo`'s DNS cache, the nameserver for `example.com` will be replaced by the attacker's nameserver `ns.attacker32.com`. To demonstrate the success of this attack, students need to show that such a record is in `Apollo`'s DNS cache.

Task overview. Implementing the Kaminsky attack is quite challenging, so we break it down into several sub-tasks. In Task 4, we construct the DNS request for a random hostname in the `example.com` domain. In Task 5, we construct a spoofed DNS reply from `example.com`'s nameserver. In Task 6, we put everything together to launch the Kaminsky attack. Finally in Task 7, we verify the impact of the attack.

3.2 Task 4: 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.

Students need to write a program to send out DNS queries to the target DNS server (i.e., the local DNS server in our setup). Students' job is to write this program and demonstrate (using Wireshark) that their queries can trigger the target DNS server to send out corresponding DNS queries. The performance requirement for this task is not high, so students can use C or Python (using Scapy) to write this code. A Python code snippet is provided in the following (the +++'s are placeholders; students need to replace them with actual values):

```
Qdsec  = DNSQR(qname='www.example.com')
dns    = DNS(id=0xAAAA, qr=0, qdcount=1, ancount=0, nscount=0,
             arcount=0, qd=Qdsec)

ip  = IP(dst='+++', src='+++')
udp = UDP(dport=+++, sport=+++, chksum=0)
request = ip/udp/dns
```

Scapy. If you use Python3, the version of the SEED VM may not have Scapy installed. You can use the following command to install Scapy for Pyhon3.

```
$ sudo pip3 install scapy
```

3.3 Task 5: 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. Students first need to find out the IP addresses of `example.com`'s legitimate nameservers (it should be noted that there are multiple nameservers for this domain).

Students can use Scapy to implement this task. The following code snippet constructs an DNS response packet that includes a question section, an answer section, and an NS section. In the sample code, we use +++ as placeholders; students need to replace them with the correct values that are needed in the Kaminsky attack. Students need to explain why they pick those values.

```
name   = '+++'
domain = '+++'
ns     = '+++'

Qdsec  = DNSQR(qname=name)
Anssec = DNSRR(rrname=name,   type='A',  rdata='1.2.3.4', ttl=259200)
NSsec  = DNSRR(rrname=domain, type='NS', rdata=ns, ttl=259200)
dns    = DNS(id=0xAAAA, aa=1, rd=1, qr=1,
             qdcount=1, ancount=1, nscount=1, arcount=0,
             qd=Qdsec, an=Anssec, ns=NSsec)

ip    = IP(dst='+++', src='+++')
udp   = UDP(dport=+++, sport=+++, chksum=0)
reply = ip/udp/dns
```

Since this reply 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 that the spoofed packets are valid.

3.4 Task 6: 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 the spoofed DNS replies like what we did in the previous task, the success rate is too low. Students can use C, but constructing DNS packets in C is non-trivial. We have introduced a hybrid approach using both Scapy and C.

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. We have include a code skeleton for the C code on the lab's website (`attack.c`). Students can make changes in the marked areas. Detailed explanation of the code is given in the guideline section.

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 shell script 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).

```
#!/bin/bash

sudo rndc dumpdb -cache
cat /var/cache/bind/dump.db | grep attacker
```

3.5 Task 7: 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 VM, 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 VM.

```
// Ask the local DNS server to do the query
$ dig www.example.com

// Directly query the attacker32 nameserver
$ dig @ns.attacker32.com www.example.com
```

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. Use the packet trace to prove that your attack is successful.

4 Guidelines

To implement the Kaminsky attack, we can use Scapy to do the packet spoofing. Unfortunately, the speed of Python is too slow; the number of packets generated per second is too low to make the attack successful. It is better to use a C program. This could be quite challenging to many students, because constructing DNS packets using C is not very easy. I have developed a hybrid method, and have experimented with it in my own class. Using this approach, students' time spent on coding can be significantly reduced, so they can spend more time focusing on the actual attack.

The idea is to leverage the strength of both Scapy and C: Scapy is much more convenient in creating DNS packets than C, but C is much faster. Therefore we simply use Scapy to create the spoofed DNS packet, and save it to a file. We then load the packet into a C program. Even though we need to send a lot of different DNS packets during the Kaminsky attack, these packets are mostly the same, except for a few fields. Therefore, we can use the packet generated from Scapy as the basis, find the offsets where changes need to be made (e.g., the transaction ID field), and directly make changes. This will be much easier than creating the entire DNS packets in C. After the changes are made, we can use the raw socket to send out the packets. Details of such a hybrid method are provided in the Packet Sniffing and Spoofing chapter of the SEED book [1]. The following Scapy program creates a simple DNS reply packet, which is saved into a file.

Listing 1: `generate_dns_reply.py`

```
#!/usr/bin/python3
from scapy.all import *

# Construct the DNS header and payload
name   = 'twysw.example.com'
Qdsec  = DNSQR(qname=name)
Anssec = DNSRR(rrname=name, type='A', rdata='1.1.2.2', ttl=259200)
dns    = DNS(id=0xAAAA, aa=1, rd=0, qr=1, qdcount=1, ancount=1,
             nscount=0, arcount=0, qd=Qdsec, an=Anssec)

# Construct the IP, UDP headers, and the entire packet
ip  = IP(dst='10.0.2.7', src='1.2.3.4', chksum=0)
udp = UDP(dport=33333, sport=53, chksum=0)
pkt = ip/udp/dns

# Save the packet to a file
with open('ip.bin', 'wb') as f:
  f.write(bytes(pkt))
```

In a C program, we load the packet from the file `ip.bin`, and use it as our packet template, based on which we create many similar packets, and flood the target local DNS servers with these spoofed replies. For each reply, we change three places: the transaction ID and the name `twysw` occurred in two places (the question section and the answer section). The transaction ID is at a fixed place (offset `28` from the beginning of our IP packet), but the offset for the name `twysw` depends on the length of the domain name. We can use a binary editor program, such as `bless`, to view the binary file `ip.bin` and find the two offsets of `twysw`. In our packet, they are at offsets `41` and `64`.

The following code snippet shows how we make change to these fields. We change the name in our reply to `bbbbb.example.com`, and then send out a spoofed DNS replies, with transaction ID being `1000`. In the code, the variable `ip` points to the beginning of the IP packet.

```
  // Modify the name in the question field (offset=41)
  memcpy(ip+41, "bbbbb" , 5);

  // Modify the name in the answer field (offset=64)
  memcpy(ip+64, "bbbbb" , 5);

  // Modify the transaction ID field (offset=28)
  unsigned short id = 1000;
  unsigned short id_net_order = htons(id);
  memcpy(ip+28, &id_net_order, 2);
```

Generate random names. In the Kaminsky attack, we need to generate random hostnames. There are many ways to do so. The following code snippet shows how to generate a random name consisting of 5 characters.

```
char a[26]="abcdefghijklmnopqrstuvwxyz";

// Generate a random name of length 5
char name[5];
for (int k=0; k<5; k++)
```

```
    name[k] = a[rand() % 26];
```

5 Submission

Students need to submit a detailed lab report to describe what they have done and what they have observed. Report should include the evidences to support the observations. Evidences include packet traces, screen dumps, etc.

References

[1] Wenliang Du. *Computer & Internet Security: A Hands-on Approach, 2nd Edition*. Self publishing, May 2019. ISBN: 978-1733003933. URL: https://www.handsonsecurity.net.

[2] D. Schneider. Fresh phish, how a recently discovered flaw in the internet's domain name system makes it easy for scammers to lure you to fake web sites. *IEEE Spectrum*, 2008. http://spectrum.ieee.org/computing/software/fresh-phish.