Remote DNS Cache Poisoning Attack Lab

Due: March 10th by 11:59 PM

1 Lab Overview

The objective of this lab is to gain first-hand experience on the remote DNS cache poisoning attack, also called the Kaminsky DNS attack [1]. DNS [2] (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 Pharming [4] 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 Pharming attack technique, called *DNS Cache Poisoning attack*. There are two main ways to perform this attack, local (where the attacker and the victim DNS server are on the same network, where packet sniffing is possible) and remote (where packet sniffing is not possible, so the attack becomes much more challenging than the local attack). This lab will focus on the remote attack.

2 Lab Environment

The lab environment involves using one single physical machine, which runs three virtual machines:

1. A computer for the victim user
2. A DNS server
3. The attacker's computer

These will be three different virtual machines, running the provided virtual machine image that you used for the previous lab.

Import three virtual machines as described in `CS528_Lab_0.pdf`. Name one virtual machine `Apollo` (this will be the victim DNS Server), name the second `dns_user`, and the third `dns_attacker` (or feel free to pick your favorite names).

Figure 1 illustrates the setup of the lab environment. For the sake of simplicity, all these virtual machines are on the same LAN, but for your final submission you are not allowed to exploit this fact in your attacks, and the environment should treat the attacker machine as a remote machine, i.e., the attacker cannot sniff the victim DNS server's packets. While testing your attacker feel free to exploit the fact that the victim DNS server is on the same LAN, but for your final submission you are not allowed to exploit this situation. In these instructions, it is assumed that the user machine's IP address is `192.168.15.5`, the DNS Server's IP is `192.168.15.4` and the attacker machine's IP is `192.168.15.6`. However, in your lab, you can use other IP addresses, as long as you make it clear in your report which address is for which machine.

2.1 Configure the Local DNS server `Apollo`

For this lab you will be using the `BIND9` server program for your DNS server. This program is included in the provided virtual machine image.

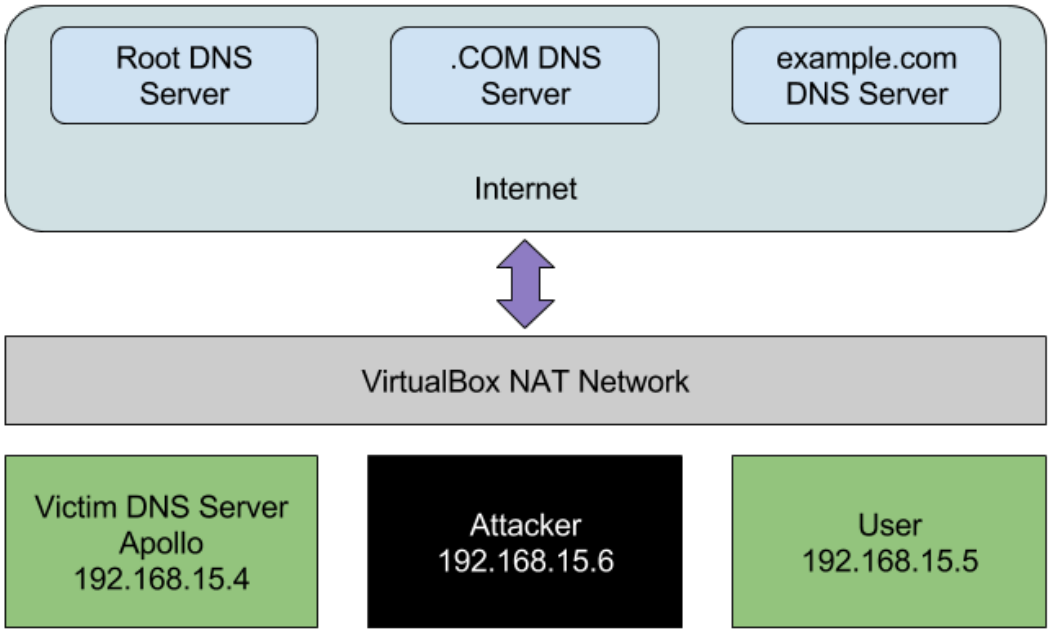

Figure 1: The Lab Environment Setup

Step 1: Create the `named.conf.options` file. The DNS server needs to read a configuration file `/etc/bind/named.conf` to start. This configuration file usually includes an option file, which is called `/etc/bind/named.conf.options`. Please add the following entry to the options (inside the `options{}` block):

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

It should be noted that the file `/var/cache/bind/dump.db` is used to dump DNS server's cache. Here are some related commands that you may find useful:

```
% sudo rndc flush          // Flush the DNS cache
% sudo rndc dumpdb -cache  // Dump the cache to dump.db
```

Step 2: Start DNS server. We can now start the DNS server using the following command:

```
% sudo /etc/init.d/bind9 restart
or
% sudo service bind9 restart
```

2.2 Configure the User Machine

On the user machine `192.168.15.5`, we need to use `192.168.15.4` as the default DNS server. This is achieved by changing the DNS setting file `/etc/resolv.conf` of the user machine:

```
nameserver 192.168.15.4 # the ip of the DNS server you just setup
```

Note: Make sure that this is the only nameserver entry in your `/etc/resolv.conf`. Also note that,

in Lubuntu, /etc/resolv.conf is be overwritten by the DHCP client. As a result you will need to manually edit the resolv.conf file every time you reboot the client machine.

Note: Check that the DNS configuration is working by running:

```
% ping www.google.com -c 10
  PING www.google.com (172.217.4.196) 56(84) bytes of data.
64 bytes from lga15s48-in-f196.1e100.net ...
...
--- www.google.com ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9025ms
rtt min/avg/max/mdev = 9.328/9.922/12.057/0.765 ms
```

If it does not work, wait a few minutes and try again. If it still does not work, you need to restart Apollo and the user machine and reconfigure the user machine.

3 The Wireshark Tool

Wireshark is a very important tool for this lab and is already installed on the virtual machine. It is a sophisticated tool for packet sniffing. Full documentation on Wireshark can be found here:

https://www.wireshark.org/docs/wsug_html_chunked/index.html

4 Lab Tasks

The main objective of "pharming" 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, assume www.example.edu is an online banking site. When the user tries to access this site using the correct URL www.example.edu, if the adversaries can redirect the user to a malicious web site that looks very much like www.example.edu, the user might be fooled and give away his/her credentials to the attacker.

In this task, you will use the domain name www.example.edu as our attacking target. It should be noted that the example.edu domain name is reserved for use in documentation, not for any real company. The authentic IP address of www.example.edu is 93.184.216.34, and it's name server is managed by the Internet Corporation for Assigned Names and Numbers (ICANN). When a user runs the dig command on this name or types the name in the browser, the user's machine sends a DNS query to the local DNS server, which will eventually ask for the IP address from example.edu's name server.

The goal of this lab is to launch a DNS cache poisoning attack on the local DNS server, such that when the user runs the dig command to find out www.example.edu's IP address, the local DNS server will end up going to the attacker's name server ns.dnslabattacker.net 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 the authentic www.example.edu.

There are two tasks in this attack:

1. **Cache poisoning.** In this task, you need to poison the DNS cache of the user's local DNS server Apollo, such that, in Apollo's DNS cache, ns.dnslabattacker.net is set as the name server for the example.edu domain, instead of the domain's registered authoritative name server.

2. **Result verification.** In this task, you need to demonstrate the impact of the attack. More specifically, you need to run the command "`dig www.example.edu`" from the user's machine, and the returned result must be a fake IP address.

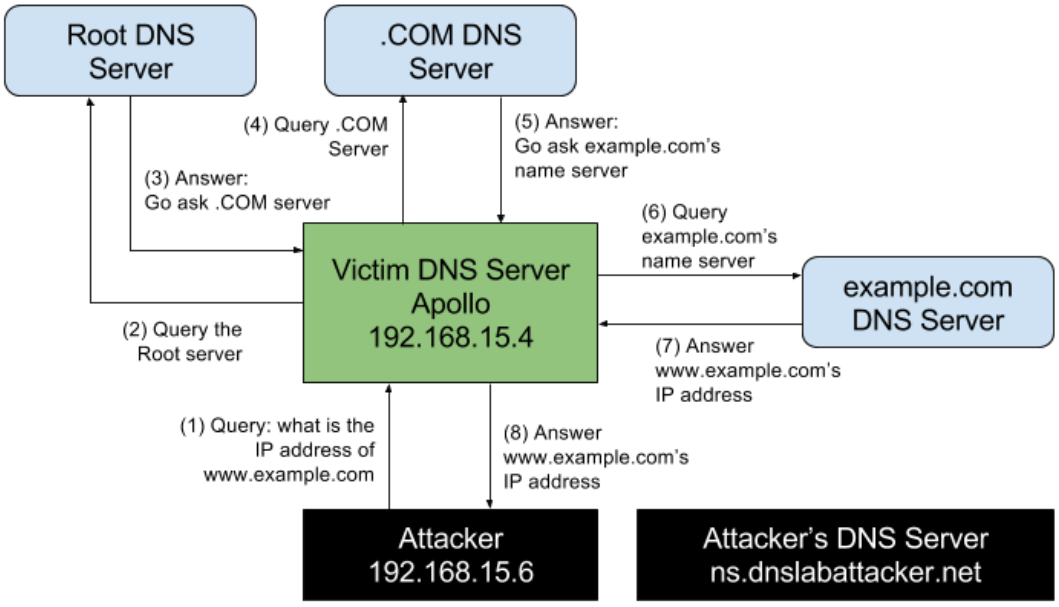

Figure 2: The complete DNS query process

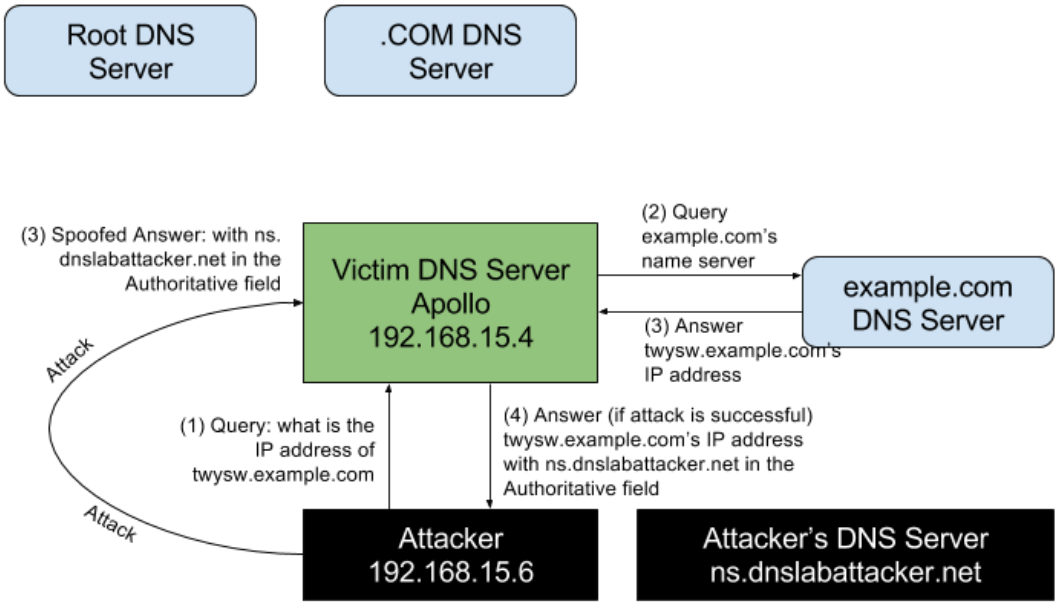

Figure 3: The DNS query process when `example.edu`'s name server is cached

4.1 Task 1: Remote Cache Poisoning (65 points)

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.edu`'s DNS server. This is illustrated in Figure 2. In the case that `example.edu`'s name server information is already cached by `Apollo`, the query will not go through 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.edu`'s name server, the attacker can send forged replies to `Apollo`, pretending that the replies are from `example.edu`'s name server. If the forged replies arrive first, it will be accepted by `Apollo`. The attack will be successful.

In a local DNS attack, 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 fortunate 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 domain name, because the DNS server will not send out another DNS query for this domain name before the cache times out. To forge another response on the same domain name, the attacker has to wait for another DNS query on this domain 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 techique to defeat the caching effect [1]. 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]. In this task, you will use 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.edu`, such as `twysw.example.edu`, 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.edu` domain.
3. While `Apollo` waits for the reply, the attacker floods `Apollo` with a stream of spoofed DNS response [6], 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.edu`, the attacker also provides an "Authoritative Nameservers" record, indicating `ns.dnslabattacker.net` as the name server for the `example.edu` 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 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 attacker another chance to do the spoofing attack. This effectively defeats the caching effect.

5. If the attack succeeds, in Apollo's DNS cache, the name server for example.edu will be replaced by the attacker's name server ns.dnslabattacker.net. To demonstrate the success of this attack, you need to show that such a record is in Apollo's DNS cache. Figure 5 shows an example of poisoned DNS cache.

Attack Configuration. You need to make the following configuration for this task:

1. *Configure the Attack Machine.* You need to configure the attack machine, so it uses the targeted DNS server (i.e., Apollo) as its default DNS server. Please see Section 2.2 for the instruction.

2. *Source Ports.* Some DNS servers now randomize the source port number in the 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 on Apollo:

```
query-source port 33333;
```

3. *DNSSEC.* Most DNS servers now adopt a protection scheme called "DNSSEC", which is designed to defeat the DNS cache poisoning attack. If you do not turn it off, your attack would be extremely difficult, if possible at all. In this lab, you will turn it off by changing the file /etc/bind/named.conf.options on Apollo. Please find the line "dnssec-validation auto", comment it out, and then add a new line. See the following:

```
//dnssec-validation auto;
  dnssec-enable no;
```

4. *Flush the Cache.* Flush Apollo's DNS cache, and restart its DNS server.

Forge DNS Response Packets. In order to complete the attack, the attacker first needs to send DNS queries to Apollo for some random host names in the example.edu domain. Right after each query is sent out, the attacker needs to forge a large number of DNS response packets in a very short time window, hoping that one of them has the correct transaction ID and it reaches the target before the authentic response does. It is better to write C code to achieve this. To make your life easier, a sample code file called "udp.c" has been provided. This program can send a large number of DNS packets. Feel free to use this sample code when writing your attack programs.

1. When modifying the udp.c program, you need to fill each DNS field with the correct value. To understand the value in each field, you can use Wireshark to capture a few DNS query and response packets.

2. DNS response packet details: it is not easy to construct a correct DNS response packet. A sample packet to help you is provided. Figure 4 is the screen shot of an example response packet: 10.0.2.6 is the local DNS server address, and 199.43.132.53 is the real name server for example.edu. The IP address in the example is old, in your case you should figure out the IP address of the name

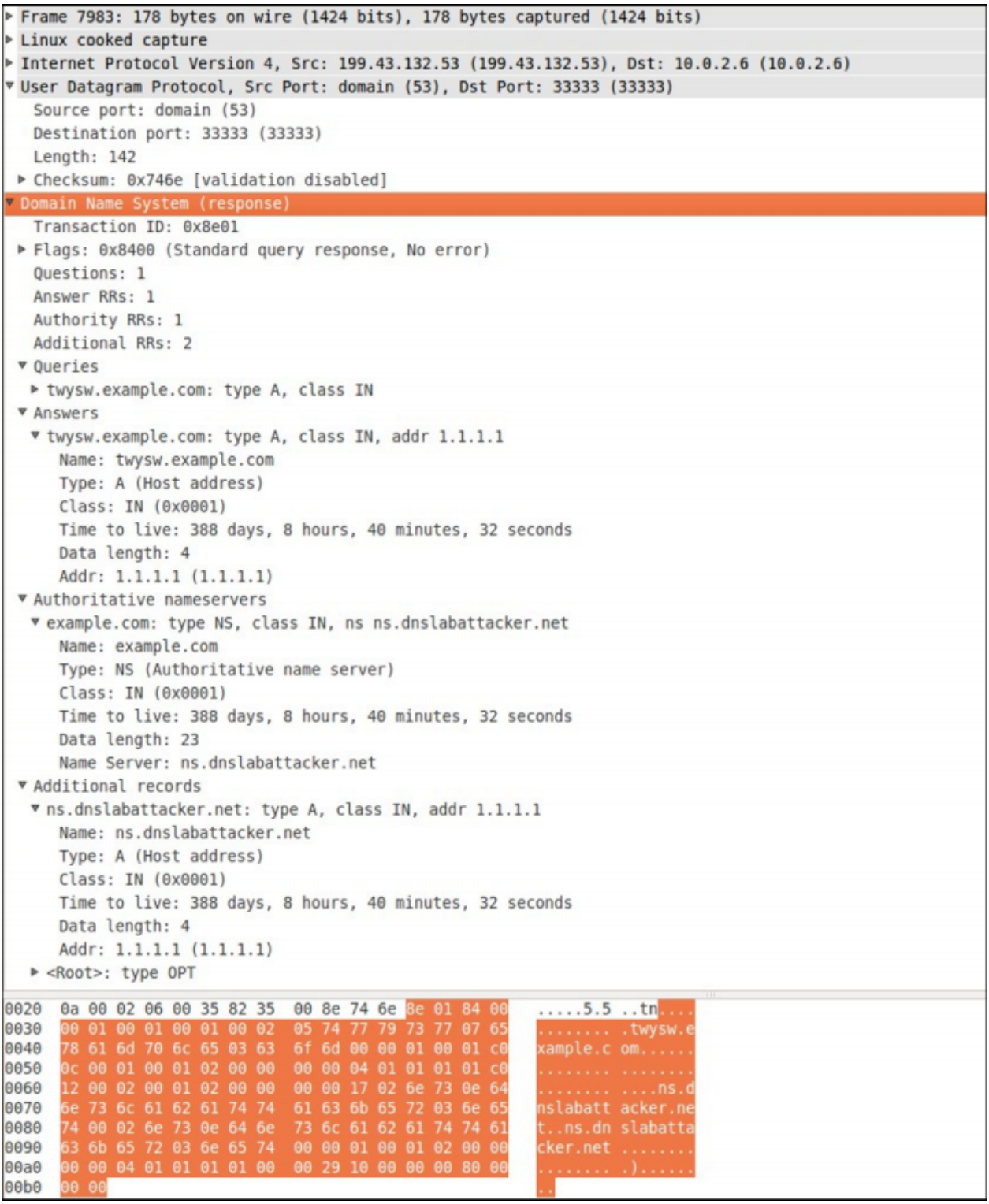

Figure 4: A Sample DNS Response Packet

server by running `dig www.example.edu`. The highlighted bytes are the raw UDP payload data, and you need to figure out what they are. The details about how each byte works are explained clearly

in Appendix A. There are several techniques used in the response packet, such as the string pointer offset to shorten the packet length. You may not have to use that technique but it is very common in real packets.

Check the `dump.db` file to see whether your spoofed DNS response has been successfully accepted by the DNS server. See an example in Figure 5.

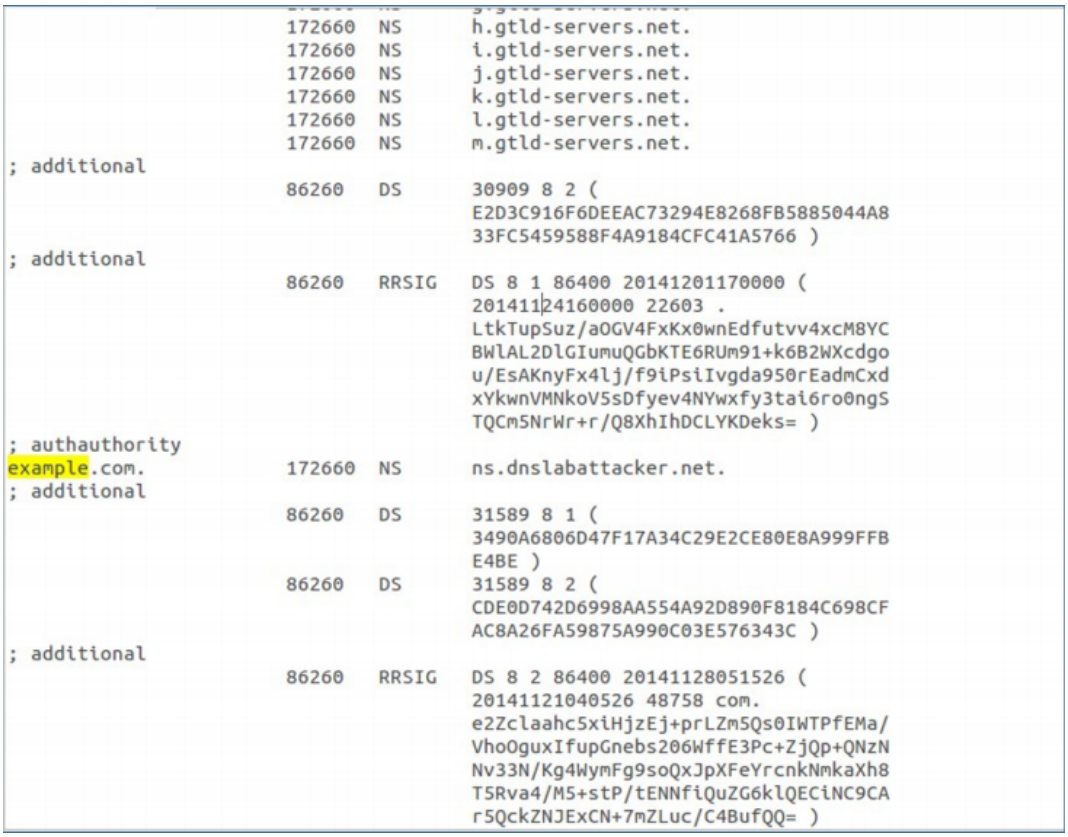

Figure 5: A Sample of Successfully Poisoned DNS Cache

Tasks. Implement and report the following:

1. Poison the cache of Apollo
2. Show the poisoned DNS cache (in `dump.db`)
3. Write the steps needed to achieve the above (Writing for the entire lab earns 15 points)

4.2 Task 2: Result Verification (30 points)

If your attack is successful, `Apollo`'s DNS cache will look like that in Figure 5, i.e., the `NS` record for `example.edu` becomes `ns.dnslabattacker.net`. To make sure that the attack is indeed successful, run the `dig` command on the user machine (see Figure 1) to ask for `www.example.edu`'s IP address.

When `Apollo` receives the DNS query, it searches for `example.edu`'s `NS` record in its cache, and finds `ns.dnslabattacker.net`. It will therefore send a DNS query to `ns.dnslabattacker.net`.

However, before sending the query, it needs to know the IP address of `ns.dnslabattacker.net`. This is done by issuing a seperate DNS query. That is where we get into trouble.

The domain name `dnslabattacker.net` does not exist in reality. It was created for the purpose of this lab. `Apollo` will soon find out about that, and mark the `NS` entry invalid, essentially recovering from the poisoned cache. One may say that when forging the DNS response, we can use an additional record to provide the IP address for `ns.dnslabattacker.net`. The sample response packet in Figure 4 actually does that. Unfortunately, this additional record will not be accepted by `Apollo`. Please think about why and give your explanation in your lab report (hint: think about the *zones*).

To solve this requires some extra configuration on `Apollo` to demonstrate the impact of a successful cache-poisoning attack (the attack is indeed successful, the problem is that we cannot show it):

Use A Fake Domain Name. This step requires some extra configuration on `Apollo`, so it recognizes `dnslabattacker.net` as a real domain. We basically add the `ns.dnslabattacker.net`'s IP address to `Apollo`'s DNS configuration, so `Apollo` does not need to go out asking for the IP address of this hostname from a non-existing domain. The instructions are provided in the following.

We first configure the victim DNS server `Apollo`. Find the file `named.conf.default-zones` in the `/etc/bind/` folder, and add the following entry to it:

```
zone "ns.dnslabattacker.net" {
                type master;
                file "/etc/bind/db.attacker";
};
```

Create the file `/etc/bind/db.attacker`, and place the following contents in it. We let the attacker's machine and `ns.dnslabattacker.net` share the same IP (e.g `192.168.15.6`). Be aware that the format of the following contents can be messed up in the PDF file if you copy and paste, and make sure that the IP address in the file has to match the IP address of the attacker machine. We have linked the file `db.attacker` in the lab's web site.

```
$TTL 604800
@ IN SOA localhost. root.localhost. (
                2; Serial
                604800 ; Refresh
                86400 ; Retry
                2419200 ; Expire
                604800 ) ; Negative Cache TTL;
@ IN NS ns.dnslabattacker.net.
@ IN A 192.168.15.6
@ IN AAAA ::1
```

Once the setup is finished, if your cache poisoning attack is successful, any DNS query sent to `Apollo` for the hostnames in `example.edu` will be sent to `192.168.15.6`, which is attacker's machine.

We need to configure the DNS server on `192.168.15.6`, so it answers the queries for the domain `example.edu`. Add the following entry in `/etc/bind/named.conf.local` on `192.168.15.6`:

```
zone "example.edu" {
                type master;
```

```
                file "/etc/bind/example.edu.db";
};
```

Create a file called `/etc/bind/example.edu.db`, and fill it with the following contents. Please do not directly copy and paste from the PDF file, as the format may be messed up. You can download the `example.edu.db` file from the lab's web site.

```
$TTL 3D
@               IN          SOA ns.example.edu. admin.example.edu. (
                2008111001
                8H
                2H
                4W
                1D)
@               IN           NS          ns.dnslabattacker.net.
@               IN           MX          10 mail.example.edu.
www             IN           A           1.1.1.1
mail            IN           A           1.1.1.2
*.example.edu.  IN           A           1.1.1.100
```

When the configurations are finished, do not forget to restart both `Apollo`'s and the attacker's DNS servers; otherwise, the modification will not take effect. If everything is done properly, you can use the command like "`dig www.example.edu`" on the user machine. The reply would be `1.1.1.1`, which is exactly we put in the above file.

Tasks. Implement and report the following:

1. Configure the attacker and Apollo correctly
2. Show the correct output of `dig` on a user machine
3. Write the steps needed to achieve the above (Writing for the entire lab earns 15 points)

4.3 Task 3: Attacking users (30 points)

So far we have seen the internals of the attack from the attacker and the vulnerable server's perspective. In this task we will launch an attack on the unsuspecting user.

The attacker's machine has an in-built `apache` server running. The source code for the web-page hosted on this webpage can be found at `/var/www/`. You will have to modify the `index.html` file to show anything other than the standard webpage on `dns attacker` when querying `wget localhost`. The objective of this task is to make the `dns user` open the web-page hosted by the attacker. Concretely, when the `dns user` runs the following:

```
% wget http://test.example.edu
```

it must open the custom web page served by the attacker.

Based on your understanding of the assignment so far, you will need to modify the DNS record(s) used by `Apollo` to get this working.

Tasks. Implement and report the following:

1. Modify the default web-page hosted by `dns attacker` and show it on the same machine.
2. Configure `Apollo`, so that `dns user` sees this custom webpage on running `wget`

5 Importance of Reporting Results

The results from this lab are very binary in nature. Either your attacker is going to be able to successfully poison the DNS cache, or it will not. Therefore, thorough documentation specifics of your process for creating your attacker are very important. A poorly documented attack program (even if it is able to successfully poison the DNS cache) will not receive full credit.

Although you do not need to, you can install any additional libraries in the VM using [7].

6 Submission

Total points earnable: 140 points

6.1 What to Submit

Your submission directory should contain:

- All source files written by you to perform the lab tasks.
- A README text file describing how to compile and execute your source code.
- **Report (Writing quality - 15points)**: A detailed report containing an explanation of the observations. Name this report *Analysis-lab2.pdf*. Be sure to put your name in your report.

6.2 How to submit

Submit every time you have a stable version of any part of the functionality. Make sure to submit early to make sure you do not miss the deadline due to any last minute congestion.

1. Login or ssh to your mcXX machine, e.g., mc19.cs.purdue.edu
2. In the parent directory of your submission directory, type the command:

   turnin -c cs528 -p lab2 <submission-dir-name>

   where <submission-dir-name> is the name of the directory containing your files to be submitted. For example, if your program is in a directory `/homes/abc/assignment/src`, make sure you cd to the directory `/homes/abc/assignment` and type:

   turnin -c cs528 -p lab2 src

3. If you wish to, you can verify your submission by typing the following command:

   turnin -v -c cs528 -p lab2

   Do not forget the **-v** above, as otherwise your earlier submission will be erased (it is overwritten by a blank submission).

Note that resubmitting overwrites any earlier submission and erases any record of the date/time of any such earlier submission.

NOTE: Do not submit your virtual machine disk image. Copy the files for submission off of your virtual machine and into a directory on the mcXX machine.

References

[1] 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`

[2] RFC 1035 Domain Names - Implementation and Specification : https://tools.ietf.org/html/rfc1035

[3] DNS HOWTO : http://www.tldp.org/HOWTO/DNS-HOWTO.html

[4] Pharming Guide : http://www.technicalinfo.net/papers/Pharming.html

[5] DNS Cache Poisoning: https://www.secureworks.com/blog/dns-cache-poisoning

[6] DNS Client Spoof: http://evan.stasis.org/odds/dns-client_spoofing.txt

[7] Update repositories in the VM: https://www.stephenrlang.com/2017/03/setting-up-the-old-releases-repo-for-ubuntu/

A Details of DNS Response Packet

```
0x8e 0x01 transaction ID
0x84 0x00 flags:means a no-error answer
0x00 0x01 Questions No.      (1 question session)
0x00 0x01 Answer No.         (1 answer session)
0x00 0x01 Authority No.      (1 authority session)
0x00 0x02 Additional No.     (2 additional sessions)
query session: eggdd.example.edu:type A, class IN
0x05 5 characters follow
0x74 t
0x77 w
0x79 y
0x73 s
0x77 w
0x07 7 characters follow
0x65 e
0x78 x
0x61 a
0x6d m
0x70 p
0x6c l
0x65 e
0x03 3 characters
0x63 c
0x6f o
0x6d m
0x00 end of the string
0x00 0x01 type:A(address)
0x00 0x01 Class:IN
the Answer session:
0xc0 first two bits set to 1 to notify this is a pointer for a name string,
not a standard
string as before
0x0c the offset of the start point: here from transaction ID field to the
name string
12 bytes. The string will shows from the offset point to the end of the
string
0x00 0x01 type:A
0x00 0x01 Class:IN
0x02 0x00 0x00 0x00 time to live
0x00 0x04 DataLength:4 bytes
0x01 0x01 x01 0x01 1.1.1.1
Authoritative Nameservers session:
0xC0 first two bits set to 1 to notify this is a pointer for a name string,
not a standard
string as before
0x12 Offset 18 the string should be "/7example/3com/0"
0x00 0x02 type:NS
0x00 0x01 Class:IN
0x02 0x00 0x00 0x00 time to live
0x00 0x17 DataLength:23 bytes
The string represent "/2ns/14dnslabattacker/3net"
0x02 2 characters follow
0x6e n
0x73 s
0x0e 14 characters
0x64 d
```

```
0x6e n
0x73 s
0x6c l
0x61 a
0x62 b
0x61 a
0x74 t
0x74 t
0x61 a
0x63 c
0x6b k
0x65 e
0x72 r
0x03 3 characters
0x6e n
0x65 e
0x74 t
0x00 end of the string
****************************additional session*************************************
first session :example.edu:type NS,class IN ns ns.dnslabattacker.net

notice: you can use the same pointer technique we
talked before to shorten
the packet, this is just to show you both ways work.
The string represent "/2ns/14dnslabattacker/3net"
0x02 2 characters follow
0x6e n
0x73 s
0x0e 14 characters
0x64 d
0x6e n
0x73 s
0x6c l
0x61 a
0x62 b
0x61 a
0x74 t
0x74 t
0x61 a
0x63 c
0x6b k
0x65 e
0x72 r
0x03 3 characters
0x6e n
0x65 e
0x74 t
0x00 end of the string
0x00 0x01 type:A
0x00 0x01 Class:IN
0x02 0x00 0x00 0x00 time to live
0x00 0x04 DataLength:4 bytes
0x01 0x01 0x01 0x01 1.1.1.1
second session: not related to the lab. Just set a rule that during the DNS
communication, the server won't accept
the packet which is larger than a certain size
0x00 0x00 0x29 0x10 0x00 0x00 0x00 0x88 0x00 0x00 0x00
```