# Remote DNS Cache Poisoning Attack Lab

## 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 Pharming 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*. 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 sniffing and spoofing

**Readings and related topics.** Detailed coverage of DNS and its attacks can be found in Chapter 15 of the SEED book, *Computer Security: A Hands-on Approach*, 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.

## 2 Lab Environment

The main purpose of this lab is on DNS attacks, and our attacking target is a local DNS server. Obviously, it is illegal to attack a real machine, 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. We will run these three virtual machines on one physical 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, and they should treat the attacker machine as a remote machine, i.e., the attacker cannot sniff victim DNS server's packets.

In the following sections, we assume that the user machine's IP address is `10.0.2.18`, the local DNS Server's IP is `10.0.2.16` and the attacker machine's IP is `10.0.2.17`. We call the Local DNS server `Apollo` in this document.

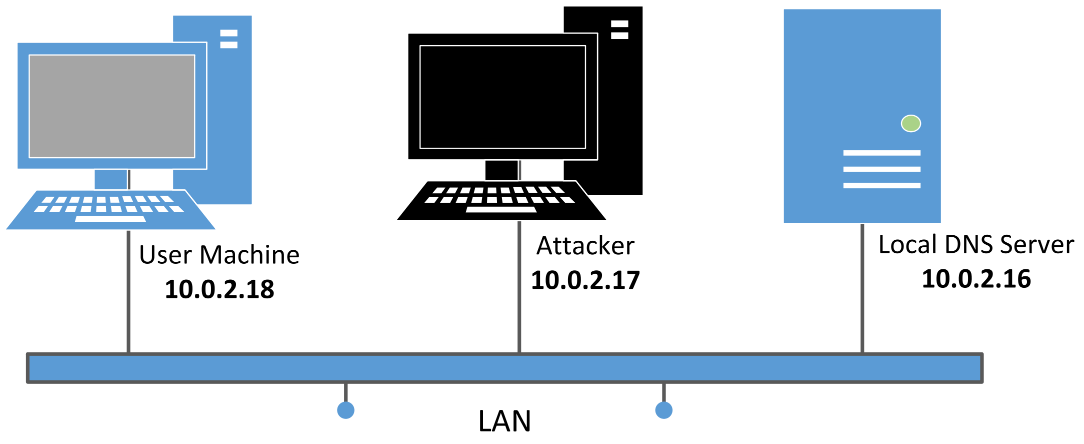


Figure 1: Environment setup for the experiment

## 2.1 Configure the Local DNS Server `Apollo`

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.

**Step 1: Configure the BIND 9 server.** 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. Let us first set up an option related to DNS cache by adding a `dump-file` entry to the `options` block:

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

The above 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`. The two commands shown below 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
```

**Step 2: 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;
  };
```

**Step 3: 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: 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 5: Start DNS server.** We can now start 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
```

## 2.2 Configure the User Machine

On the user machine `10.0.2.18`, we need to use `10.0.2.16` as the local DNS server (by default, the DNS server program is already running in the SEED VM). This is achieved by changing the resolver configuration file (`/etc/resolv.conf`) of the user machine, so the server `10.0.2.16` 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:

```
Add the following entry to /etc/resolvconf/resolv.conf.d/head
  nameserver 10.0.2.16

Run the following command for the change to take effect
  $ sudo resolvconf -u
```

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 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.

## 3 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, 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 name server 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 name server.

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.dnslabattacker.net` 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 the authentic `www.example.com`.

There are two tasks in this attack: cache poisoning and result verification. In the first task, students 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.com` domain, instead of the domain's registered authoritative name server. In the second task, students need to demonstrate the impact of the attack. More specifically, they need to run the command `"dig www.example.com"` from the user's machine, and the returned result must be a fake IP address.

### 3.1 Task 1: Remote Cache Poisoning

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 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.com`'s name server, the attacker can send forged replies to `Apollo`, pretending that the replies are from `example.com`'s name server. 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.

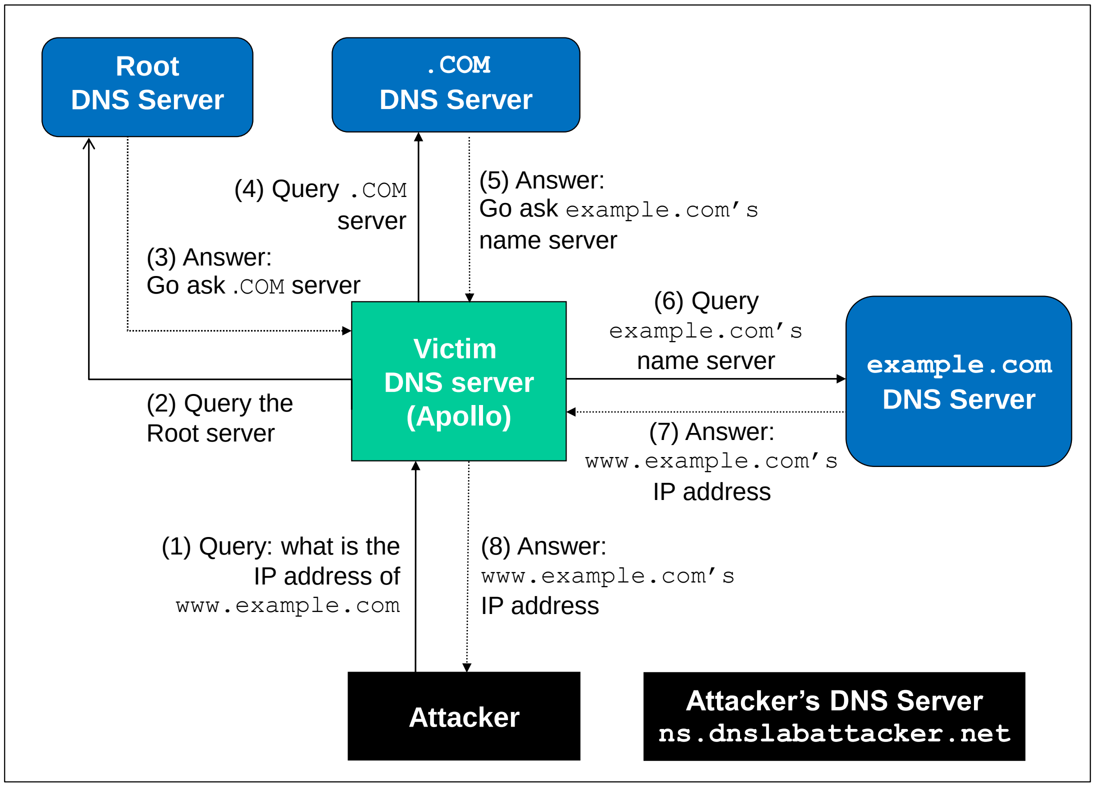

Root DNS Server
.COM DNS Server
(4) Query .COM server
(5) Answer: Go ask example.com's name server
(3) Answer: Go ask .COM server
(6) Query example.com's name server
Victim DNS server (Apollo)
example.com DNS Server
(2) Query the Root server
(7) Answer: www.example.com's IP address
(1) Query: what is the IP address of www.example.com
(8) Answer: www.example.com's IP address
Attacker
Attacker's DNS Server ns.dnslabattacker.net

Figure 2: The complete DNS query process

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 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 technique 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, 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.

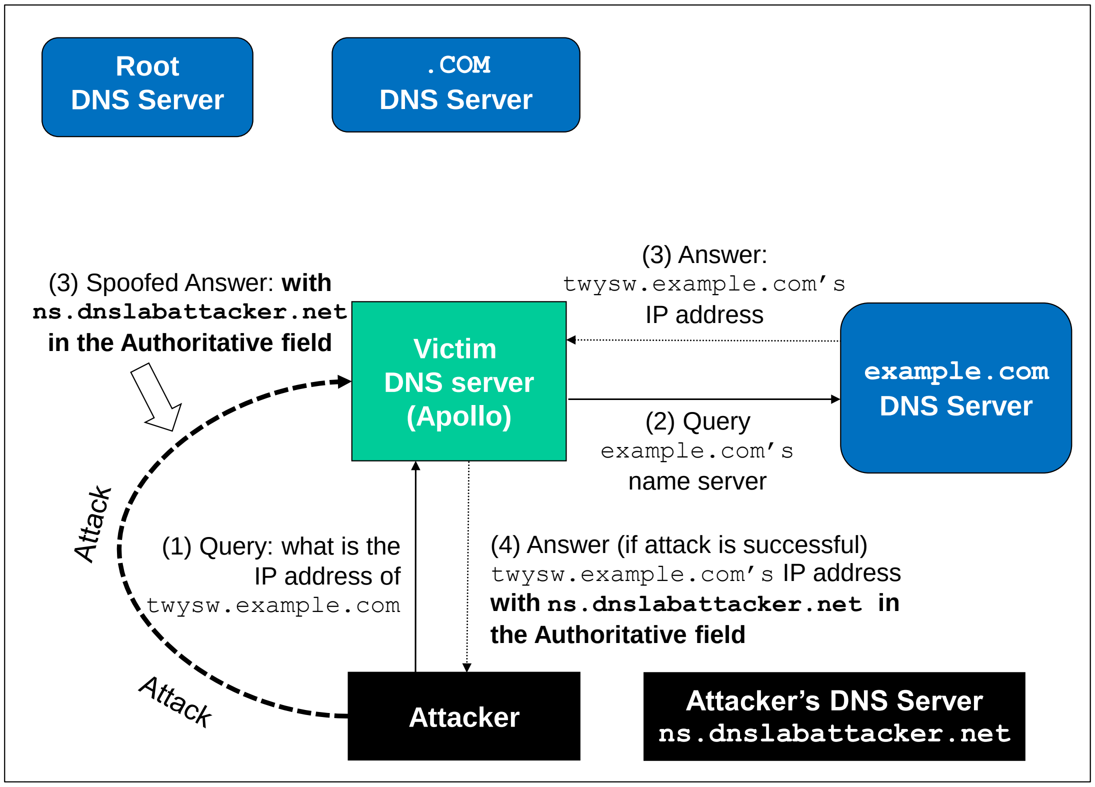

Root DNS Server
.COM DNS Server
(3) Spoofed Answer: with ns.dnslabattacker.net in the Authoritative field
(3) Answer: twysw.example.com's IP address
Victim DNS server (Apollo)
example.com DNS Server
(2) Query example.com's name server
Attack
(1) Query: what is the IP address of twysw.example.com
(4) Answer (if attack is successful) twysw.example.com's IP address with ns.dnslabattacker.net in the Authoritative field
Attack
Attacker
Attacker's DNS Server ns.dnslabattacker.net

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

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.dnslabattacker.net` as the name server 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 name server for `example.com` will be replaced by the attacker's name server `ns.dnslabattacker.net`. To demonstrate the success of this attack, students need to show that such a record is in `Apollo`'s DNS cache. Figure 4 shows an example of poisoned DNS cache.

**Attack Configuration.** We 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. Make sure that the network configuration for this VM is `"NAT Network"`.

**Task 1.1: Spoofing DNS request.** Implementing the Kaminsky attack is quite challenging, so we break it down into three sub-tasks. This subtask focuses on spoofing DNS requests. In order to complete the attack, we (as attackers) need to trigger the target DNS server to send out DNS queries, so we have a chance to spoof DNS replies. Since we need to try many times before we can succeed, we have to automate the process. The first step is to write a program to send out DNS queries to the target DNS server, each time with a different hostname in the question field. Your job is to write this program and demonstrate using Wireshark that your queries can trigger the target DNS server to send out DNS queries on behalf of you.

Because we do not need to send out DNS queries very fast, we can afford to use an external program to do that for us, instead of implementing everything in C programs. For example, you can use `system()` to invoke dig:

```
system("dig xyz.example.net"}
```

First needs to send DNS queries to `Apollo` for some random host names in the `example.com` 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.

**Task 1.2: Spoofing DNS Replies.** We need to spoof DNS replies in the Kaminsky attack. Performance is essential for the success of the attack, so it is better to write the program in C. To make your life easier, we have provided a sample code called `spoofdns.c`, which shows how to construct DNS reply packets.

1. When modifying the `spoofdns.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. Explain the code ...

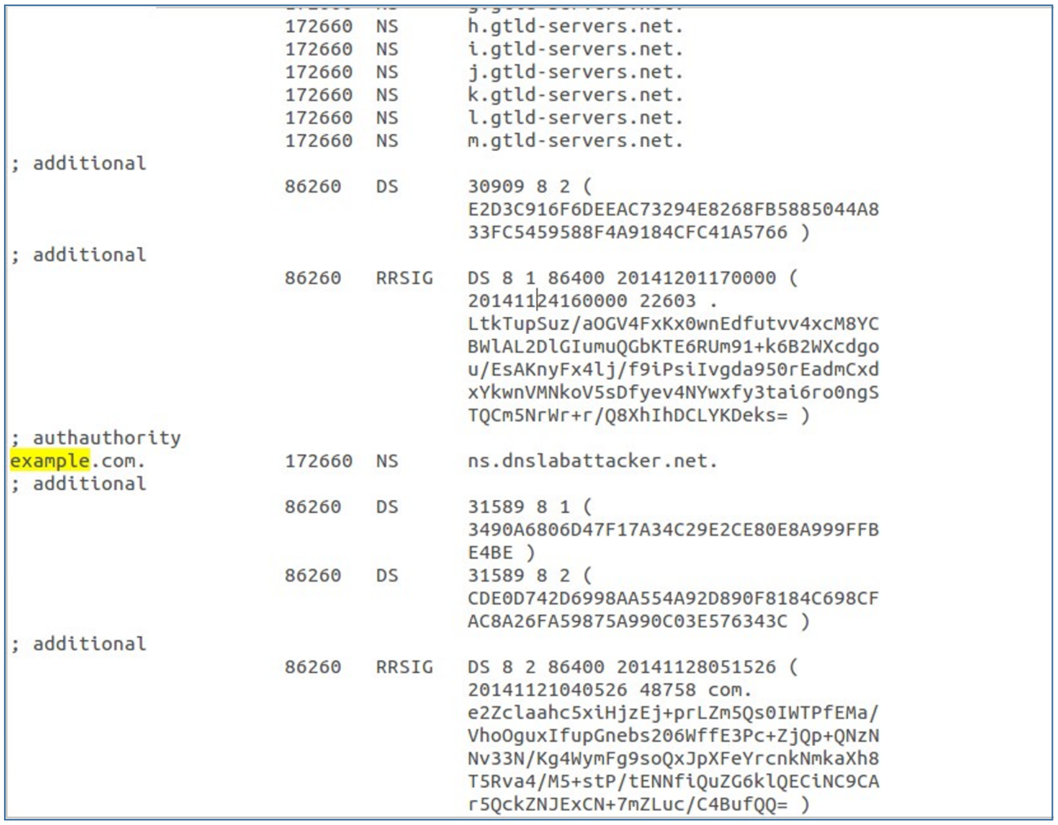
Please show what kind of replies you are spoofing and explain why. You should use Wireshark to capture your spoofed DNS replies, and show the details of some selected replies in the report.

**Task 1.3: The Kaminsky Attack.** Now we can put everything together to conduct the Kaminsky attack. Launch your attack and then 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 4.

## 3.2 Task 2: Result Verification

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

When `Apollo` receives the DNS query, it searches for `example.com`'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 separate DNS query. That is where we get into trouble.



```
                         172660  NS      h.gtld-servers.net.
                         172660  NS      i.gtld-servers.net.
                         172660  NS      j.gtld-servers.net.
                         172660  NS      k.gtld-servers.net.
                         172660  NS      l.gtld-servers.net.
                         172660  NS      m.gtld-servers.net.
; additional
                         86260   DS      30909 8 2 (
                                         E2D3C916F6DEEAC73294E8268FB5885044A8
                                         33FC5459588F4A9184CFC41A5766 )
; additional
                         86260   RRSIG   DS 8 1 86400 20141201170000 (
                                         20141124160000 22603 .
                                         LtkTupSuz/aOGV4FxKx0wnEdfutvv4xcM8YC
                                         BWlAL2DlGIumuQGbKTE6RUm91+k6B2WXcdgo
                                         u/EsAKnyFx4lj/f9iPsiIvgda950rEadmCxd
                                         xYkwnVMNkoV5sDfyev4NYwxfy3tai6ro0ngS
                                         TQCm5NrWr+r/Q8XhIhDCLYKDeks= )
; authauthority
example.com.             172660  NS      ns.dnslabattacker.net.
; additional
                         86260   DS      31589 8 1 (
                                         3490A6806D47F17A34C29E2CE80E8A999FFB
                                         E4BE )
                         86260   DS      31589 8 2 (
                                         CDE0D742D6998AA554A92D890F8184C698CF
                                         AC8A26FA59875A990C03E576343C )
; additional
                         86260   RRSIG   DS 8 2 86400 20141128051526 (
                                         20141121040526 48758 com.
                                         e2Zclaahc5xiHjzEj+prLZm5Qs0IWTPfEMa/
                                         VhoOguxIfupGnebs206WffE3Pc+ZjQp+QNzN
                                         Nv33N/Kg4WymFg9soQxJpXFeYrcnkNmkaXh8
                                         T5Rva4/M5+stP/tENNfiQuZG6klQECiNC9CA
                                         r5QckZNJExCN+7mZLuc/C4BufQQ= )
```

Figure 4: A Sample of Successfully Poisoned DNS Cache

The domain name `dnslabattacker.net` does not exist in reality. We created this name 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`. 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*).

There are two ways to solve the problem, so we can demonstrate the impact of our success cache-poisoning attack (the attack is indeed successful, the problem is that we cannot show it):

**Use a Real Domain Name.** If you own a real domain and you can configure its DNS, your job is easy. Just use your own domain name in the `NS` record, instead of `dnslabattacker.net`. Please refer to the setup section in our "Local DNS Attack Lab" to configure your domain's DNS server, so it can answer the queries for the `example.com` domain.

**Use A Fake Domain Name.** If you do not own a real domain name, you can still do the demonstration using our fake domain name `ns.dnslabattacker.net`. We just need to do 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's 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 machine (`192.168.0.200`). Be aware that the format of the following contents can be messed up in the PDF file if you copy and paste. 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 192.168.0.200
@ 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.com` will be sent to `192.168.0.200`, which is attacker's machine.

We need to configure the DNS server on `192.168.0.200`, so it answers the queries for the domain `example.com`. Add the following entry in `/etc/bind/named.conf.local` on `192.168.0.200`:

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

Create a file called `/etc/bind/example.com.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.com.db` file from the lab's web site.

```
$TTL 3D
@               IN         SOA ns.example.com. admin.example.com. (
                2008111001
                8H
                2H
                4W
                1D)
@               IN          NS          ns.dnslabattacker.net.
@               IN          MX          10 mail.example.com.
www             IN          A           1.1.1.1
mail            IN          A           1.1.1.2
*.example.com   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.com` on the user machine. The reply would be `1.1.1.1`, which is exactly what we put in the above file.

## 4 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.

**Note:** Please do not forget to answer the question asked in Task 2, regarding why the IP address for `ns.dnslabattacker.net` in the additional field is not accepted by the victim DNS server.

## 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] Du, Wenliang. Computer Security: A Hands-on Approach. *CreateSpace Independent Publishing Platform*, 2017 ISBN-10: 154836794X, ISBN-13: 978-1548367947 `https://www.handsonsecurity.net/`