

Remote DNS Cache Poisoning Attack Lab

Copyright © 2014 Wenliang Du, Syracuse University.

The development of this document is/was funded by the following grants from the US National Science Foundation: No. 1303306 and 1318814. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation. A copy of the license can be found at <http://www.gnu.org/licenses/fdl.html>.

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 [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*. 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.

2 Lab Environment

We setup the lab environment using one single physical machine, which runs three virtual machines. The lab environment actually needs three separate machines, including a computer for the victim user, a DNS server, and the attacker's computer. These machines are three different virtual machines, running our provided Ubuntu VM image. For the VM network setting, if you are using VirtualBox, please use "NAT-Network" as the only network adapter for each VM. If you are using Vmware, the default "NAT" setting is good enough.

Figure 1 illustrates the setup of the lab environment. For the sake of simplicity, we do put all these VMs on the same LAN, but 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 this lab description, we assume that the user machine's IP address is 192.168.0.100, the DNS Server's IP is 192.168.0.10 and the attacker machine's IP is 192.168.0.200. However, in your lab, you can use other IP addresses, as long as you make it clear in your reports which address is for which machine.

Note for Instructors: For this lab, a lab session is desirable, especially if students are not familiar with the tools and the environments. If an instructor plans to hold a lab session (by himself/herself or by a TA), it is suggested that the following topics are covered in the lab session ¹:

1. The use of the virtual machine software.
2. The use of Wireshark.
3. Configuration of BIND 9 DNS server [3].

¹We assume that the instructor has already covered the concepts of the attacks in the lecture, so we do not include them in the lab session.

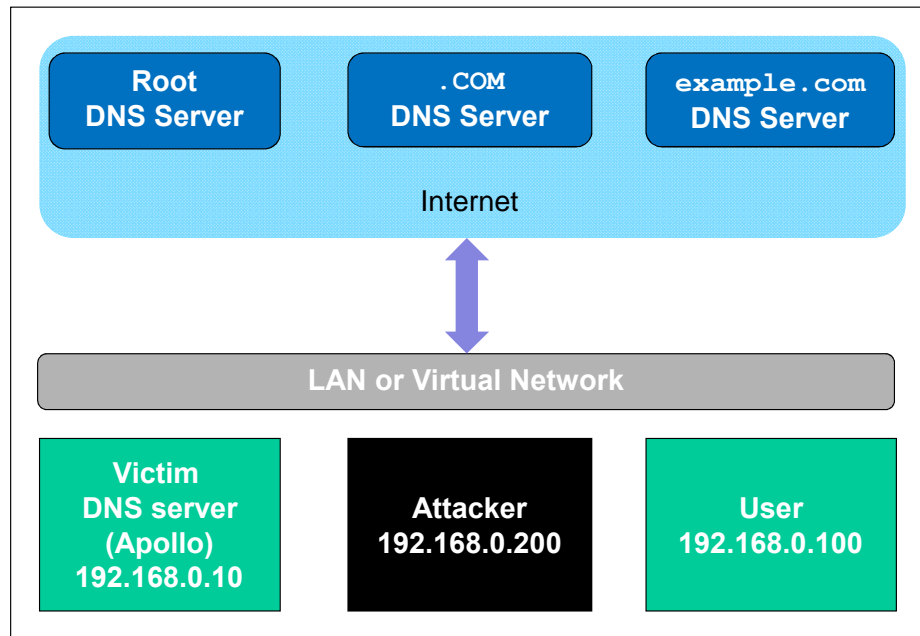


Figure 1: The Lab Environment Setup

2.1 Configure the Local DNS server Apollo

Step 1: Install the BIND 9 DNS server. The BIND 9 server program is already installed in our pre-built Ubuntu VM image. The BIND 9 software can be installed using the following command:

```
# sudo apt-get install bind9
```

Step 2: 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 contents to the option file:

```
options {  
    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 3: 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 4: Start DNS server. We can now start the DNS server using the following commands:

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

2.2 Configure the User Machine

On the user machine 192.168.0.100, we need to use 192.168.0.10 as the default DNS server. This is achieved by changing the DNS setting file `/etc/resolv.conf` of the user machine:

```
nameserver 192.168.0.10 # 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 Ubuntu, `/etc/resolv.conf` may be overwritten by the DHCP client. To avoid this, disable DHCP by doing the following (in Ubuntu 12.04):

```
Click "System Settings" -> "Network",  
Click "Options" in "Wired" Tab,  
Select "IPv4 Settings" -> "Method" -> "Automatic(DHCP) Addresses Only"  
and update only "DNS Servers" entry with IP address of BIND DNS Server.
```

```
Now Click the "Network Icon" on the top right corner and Select  
"Auto eth0". This will refresh the wired network connection and  
updates the changes.
```

You should restart your Ubuntu machine for the modified settings to take effect.

2.3 The Wireshark Tool

Wireshark is a very important tool for this lab, and you probably need it to learn how exactly DNS works, as well as debugging your attacks. This tool is already installed in our pre-built VM.

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.

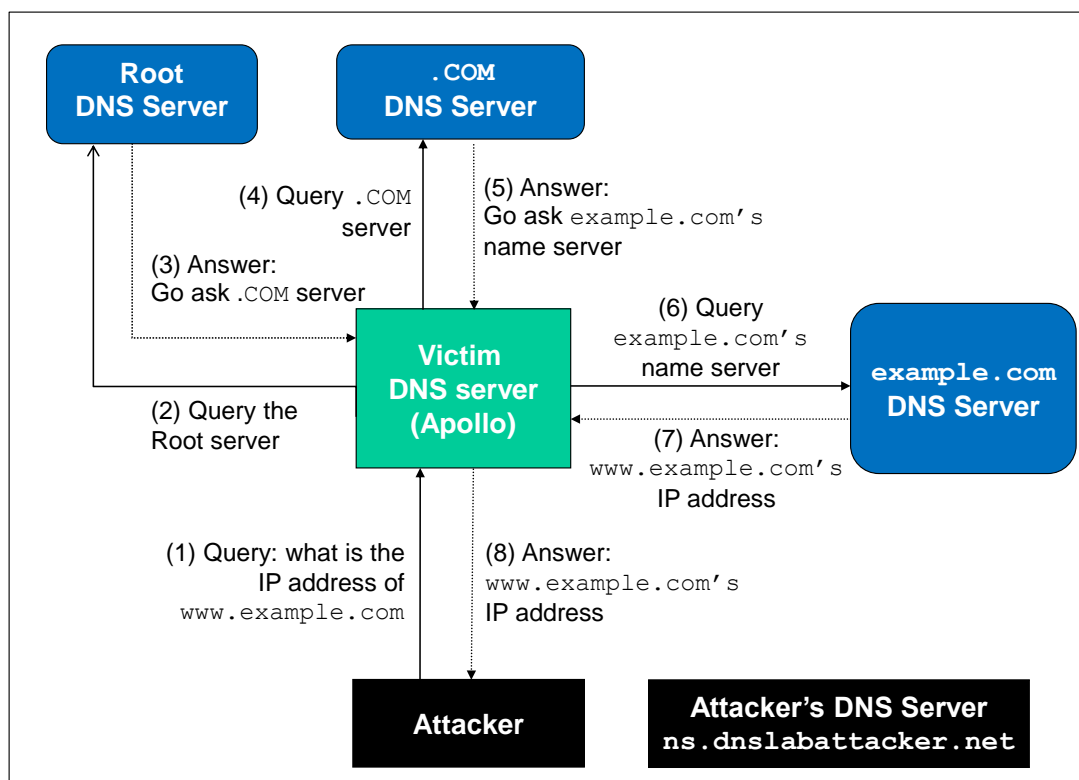


Figure 2: The complete DNS query process

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.

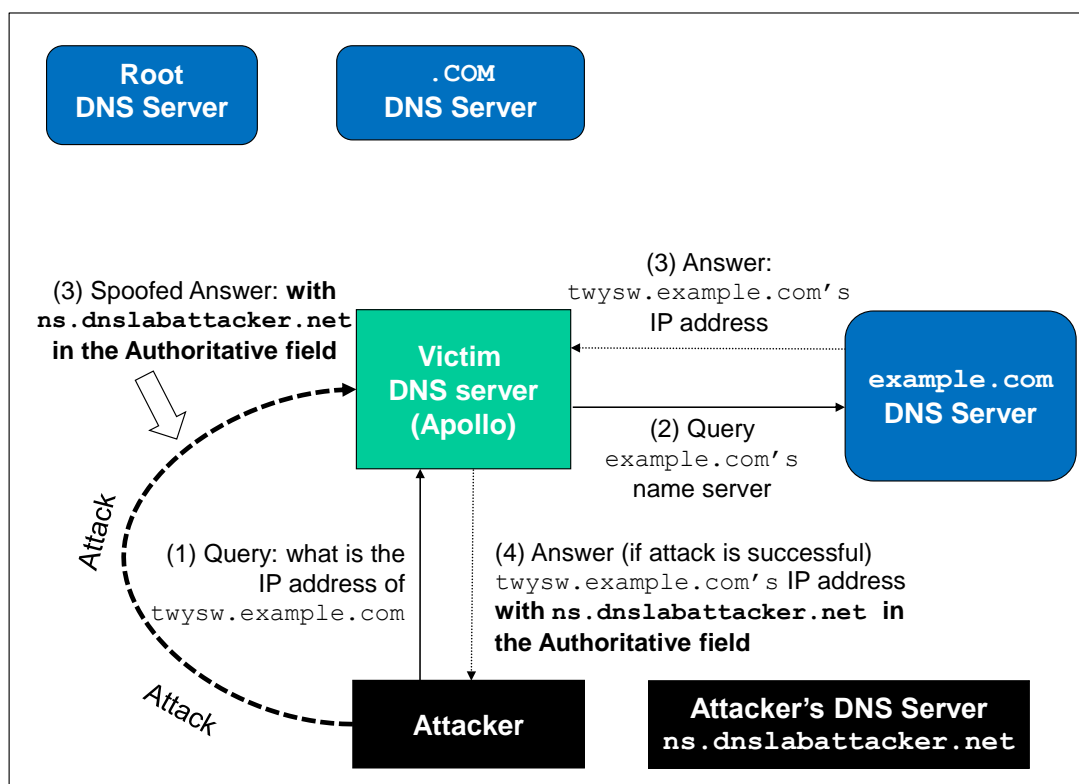


Figure 3: The DNS query process when `example.com`'s name server is cached

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 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]. 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 [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.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 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 5 shows an example of poisoned DNS cache.

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

1. *Configure the Attack Machine.* 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".
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, we will turn it off. This can be done 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.

```

▶ Frame 7983: 178 bytes on wire (1424 bits), 178 bytes captured (1424 bits)
▶ Linux cooked capture
▶ Internet Protocol Version 4, Src: 199.43.132.53 (199.43.132.53), Dst: 10.0.2.6 (10.0.2.6)
▼ User Datagram Protocol, Src Port: domain (53), Dst Port: 33333 (33333)
  Source port: domain (53)
  Destination port: 33333 (33333)
  Length: 142
  ▶ Checksum: 0x746e [validation disabled]
▼ Domain Name System (response)
  Transaction ID: 0x8e01
  ▶ Flags: 0x8400 (Standard query response, No error)
  Questions: 1
  Answer RRs: 1
  Authority RRs: 1
  Additional RRs: 2
  ▼ Queries
    ▶ twysw.example.com: type A, class IN
  ▼ Answers
    ▼ twysw.example.com: type A, class IN, addr 1.1.1.1
      Name: twysw.example.com
      Type: A (Host address)
      Class: IN (0x0001)
      Time to live: 388 days, 8 hours, 40 minutes, 32 seconds
      Data length: 4
      Addr: 1.1.1.1 (1.1.1.1)
    ▼ Authoritative nameservers
      ▼ example.com: type NS, class IN, ns ns.dnslabattacker.net
        Name: example.com
        Type: NS (Authoritative name server)
        Class: IN (0x0001)
        Time to live: 388 days, 8 hours, 40 minutes, 32 seconds
        Data length: 23
        Name Server: ns.dnslabattacker.net
    ▼ Additional records
      ▼ ns.dnslabattacker.net: type A, class IN, addr 1.1.1.1
        Name: ns.dnslabattacker.net
        Type: A (Host address)
        Class: IN (0x0001)
        Time to live: 388 days, 8 hours, 40 minutes, 32 seconds
        Data length: 4
        Addr: 1.1.1.1 (1.1.1.1)
      ▶ <Root>: type OPT
0020  0a 00 02 06 00 35 82 35 00 8e 74 6e 8e 01 84 00  ....5.5 ..tn....
0030  00 01 00 01 00 01 00 02 05 74 77 79 73 77 07 65  ....twysw.e
0040  78 61 6d 70 6c 65 03 63 6f 6d 00 00 01 00 01 c0  xample.c om.....
0050  0c 00 01 00 01 02 00 00 00 00 04 01 01 01 01 c0  .....
0060  12 00 02 00 01 02 00 00 00 00 17 02 6e 73 0e 64  .....ns.d
0070  6e 73 6c 61 62 61 74 74 61 63 6b 65 72 03 6e 65  nslabatt acker.ne
0080  74 00 02 6e 73 0e 64 6e 73 6c 61 62 61 74 74 61  t..ns.dn slabatta
0090  63 6b 65 72 03 6e 65 74 00 00 01 00 01 02 00 00  cker.net .....
00a0  00 00 04 01 01 01 01 00 00 29 10 00 00 00 80 00  .....).
00b0  00 00

```

Figure 4: A Sample DNS Response Packet

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.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. It is better to write C code to achieve this. To make your life easier, we have provided a sample code called `udp.c`. 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. We made a sample packet to help you. 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.com`. 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.

| | | | |
|---------------------------------|--------|-------|--|
| | 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/a0GV4FxKx0wnEdfutvv4xcM8YC BWLAL2DLGIumuQGbkTE6RUM91+k6B2WXcdgo u/EsAKnyFx4lj/f9iPsiIvgda950rEadmCxd xYkwnVMNkoV5sDfyev4NYwxfy3tai6ro0ngS TQcm5Nrw+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/ Vho0guxIfupGnebs206WffE3Pc+ZjQp+QNzN Nv33N/Kg4WymFg9soQxJpXFeYrcnkNmkaXh8 T5Rva4/M5+stP/tENNfiQuZG6klQECiNC9CA r5QckZNJEXCN+7mZLUC/C4BufQQ=) |

Figure 5: A Sample of Successfully Poisoned DNS Cache

3.2 Task 2: Result Verification

If your attack is successful, Apollo's DNS cache will look like that in Figure 5, 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 1) 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.

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`. 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*).

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 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] RFC 1035 Domain Names - Implementation and Specification : <http://www.rfc-base.org/rfc-1035.html>
- [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: http://www.secureworks.com/resources/articles/other_articles/dns-cache-poisoning/
- [6] DNS Client Spoof: http://evan.stasis.org/odds/dns-client_spoofing.txt

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.com: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.com: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
```