# Lab 4: DNS Cache Poisoning

## ECE568 – Computer Security

## Lab #4: DNS Cache Poisoning

## Setup

For this lab, we will be using the patched DNS server program called BIND (Berkeley Internet Name Domain). BIND is installed at `/share/copy/ece568f/lab4/bind9` on the ECF machines.

We next describe how to setup and configure BIND on your ECF machine.

**Step 1: Setup configuration files**

There are 2 configuration files you need to set for this lab. They are `rndc.conf` and `named.conf` at `/share/copy/ece568f/lab4/bind9/etc`. RNDC is short for Remote Name Daemon Control, which is useful for dumping BIND's server cache to check whether our attack has been successful or not. You need to specify a few parameters in these files. Therefore, create a local directory and copy them to it:

```
mkdir -p lab4/etc

mkdir -p lab4/var/run/named/

cp /share/copy/ece568f/lab4/bind9/etc/rndc.conf lab4/etc

cp /share/copy/ece568f/lab4/bind9/etc/named.conf lab4/etc
```

**Step 2: Specify BIND configuration parameters**

We will be using two executables from BIND, *named* and *rndc*. *named* executable is the BIND server and *rndc* is the command-line administration tool. *named* uses specific port (specified in `rndc.conf` and `named.conf`) to communicate with *rndc*. Furthermore, *named* uses two ports, *listen-on* port and *query-source* port to communicate with clients and external name servers. Since multiple users can work on ECF workstations, make sure you randomly pick port numbers that would not collide with other students. Also, you must pick port numbers greater than 1024 as ports lower than that are reserved and requires root privileges.

Here are descriptions of parameters that you need to specify before you can run *named* server:

*dump-file*: path where *named* will dump its cache

*listen-on port*: port number *named* will use to listen for DNS queries

*query-source port*: port number BIND will use to send its outgoing queries to external DNS servers

*pid-file*: *named* will write its process id here

*session-keyfile*: *named* will use this file for DNSSEC

*controls port*: port number for *rndc*

*default-port*: port number for *rndc*

Once you have chosen a port number for rndc, modify bold-faced parameters in `rndc.conf` and `named.conf`:

rndc.conf:

```
options {

  default-key “rndc-key”;

  default-server 127.0.0.1;

  default-port <your port number>;

};
```

named.conf:

```
options {

  dump-file “<home directory>/lab4/dump.db”;

  listen-on port <your port number> { any; };

  query-source port <your port number>;

  pid-file “<home directory>/lab4/var/run/named/named.pid”;

  session-keyfile “<home directory>/lab4/var/run/named/session.key”;
};

controls {

  inet 127.0.0.1 port <your port number>

  allow { 127.0.0.1; } keys { “rndc-key”; };

};
```

**Step 3: Start the DNS server**

You can run BIND by running script at `/share/copy/ece568f/lab4/run_bind.sh`:

```
/share/copy/ece568f/lab4/run_bind.sh -c <path to named.conf>
```

**Step 4: Install Scapy**

Scapy is a powerful tool for packet manipulation. To install Python packages locally, we will be using pip. Do the following to install scapy locally:

```
pip install scapy==2.3.3 --user
```

Take care to ensure that the version of scapy being installed is as shown above ie. 2.3.3. You may encounter warnings about `SNIMissingWarning` and `InsecurePlatformWarning`. You can safely ignore them.

Check if scapy is installed correctly:

```
$ python

Python 2.6.6 (r266:84292, Aug  9 2016, 06:11:56)

[GCC 4.4.7 20120313 (Red Hat 4.4.7-17)] on linux2

Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> import scapy

>>>
```

No output and the absence of an import error (ignore any warnings) indicates that we can proceed further.

When using scapy, you may get a lot of following errors:

```
Exception RuntimeError: 'maximum recursion depth exceeded while calling a Python object' in <type 'exceptions.AttributeError'> ignore
d
```

This error should not affect your lab and you can safely ignore them.

Further information on how to use scapy could be found at: [https://scapy.readthedocs.io/en/latest/](https://scapy.readthedocs.io/en/latest/) **(https://scapy.readthedocs.io/en/latest/)**

**Note**: scapy's sending functions such as: `sr1()`, `send()` **will not work** in ECF environment. These functions require sudo privileges which is disabled in ECF lab. Instead, you can use scapy to: a) build packets, b) modify packets. For sending and receiving packets, use Python's socket library. Example for using Python's socket library + scapy for building DNS packet is in `part4_starter.py`.

# Resources

In this lab, you will be performing a DNS Cache Poisoning attack as described in Lecture <lecture_number>. More details on the attack can be found here:

- **An Illustrated Guide to the Kaminsky DNS Vulnerability (http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html)**
- Analysis of the Cache Poisoning attack **(http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.172.5025&rep=rep1&type=pdf)**

# **Part 1:** Getting familiar with dig

*dig* is a useful tool for sending DNS queries. Unless otherwise specified, *dig* will by default, try each of servers listed in `/etc/resolv.conf`.

Direct *dig* to the default DNS server on the ECF machines (not the BIND server) and figure out the following:

1. What is the IPv4 address of ecf.utoronto.ca?
2. What are the names of *name servers* of ecf.utoronto.ca and their IPv4 addresses?
3. What are the names of *mail servers* of ecf.utoronto.ca and their IPv4 addresses?
4. Now direct dig to your local BIND server and repeat the above queries. Check to see if the output matches your results from 1-3.

Refer to the man page **(https://linux.die.net/man/1/dig)** to learn how to call *dig* with the appropriate options in order to point it to the BIND server. For question 1-3, create a file called `part1.txt` and write to it in the following format:

```
1. <ipv4 address of ecf.utoronto.ca>
2. <name of name server #1>:<ipv4 address of name server #1>
2. <name of name server #2>:<ipv4 address of name server #2>
2. ....
3. <name of mail server #1>:<ipv4 address of mail server #1>
3. ....
```

Note - there may be more than 1 name/mail server for ecf.utoronto.ca

# **Part 2:** Write a DNS proxy that sits between dig and local DNS server

If we can inspect DNS queries and their replies, it is possible to forge them to attack users. Imagine redirecting a DNS request of *google.com* to a malicious server.

Due to security concerns in ECF, packet sniffing is disabled. However, we can simulate it by forcing DNS queries to go through proxies. Your goal is to write a proxy server that accepts DNS queries from *dig* and forwards them to the BIND server we setup earlier. It should also receive a DNS reply from the BIND server and forward it back to *dig* (unmodified).

Refer to the [man page](https://linux.die.net/man/1/dig) **(https://linux.die.net/man/1/dig)** to learn how to call *dig* with the appropriate options in order to point it to the proxy instead of the BIND server. Check to see if you receive the same output as when you point *dig* directly to the BIND server.

We have provided some starter code that can be used to build the proxy for Parts 2 and 3 in `dnsproxy_starter.py`

Copy this into your local directory as follows

```
cp /share/copy/ece568f/lab4/dnsproxy_starter.py <path to lab4 directory>
```

Your proxy should be run as `python dnsproxy_starter.py --port <port #> --dns_port <dns port #>`

# **Part 3:** Spoof DNS reply using the DNS proxy

Your goal for this exercise is to use the proxy created in Part 2 to intercept and forge DNS replies. Look up example.com by sending:

```
dig example.com <options>
```

Spoof the DNS reply such that example.com's IPv4 address is `5.6.6.8` instead and change its name servers to `ns1.dnsattacker.net`, `ns2.dnsattacker.net`.

**Furthermore, remove all data from additional sections.**

We suggest using scapy to manipulate DNS packets.

Information about DNS packet format can be found at: [http://www.networksorcery.com/enp/protocol/dns.htm](http://www.networksorcery.com/enp/protocol/dns.htm) **(http://www.networksorcery.com/enp/protocol/dns.htm)**

Your proxy should be run as `python dnsproxy_starter.py --port <port #> --dns_port <port #> --spoof_response`

# **Part 4:** DNS cache poisoning attack

We will be implementing a DNS cache poisoning attack, also known as the Kaminsky attack.

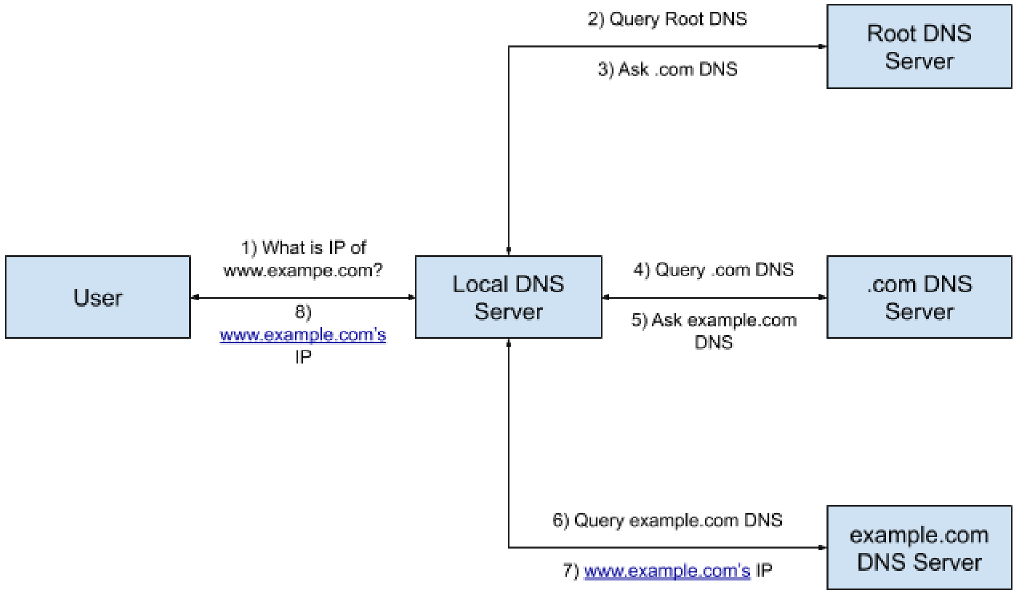


**Figure 1: DNS query process**

Figure 1 illustrates a complete DNS query process when a user submits a DNS query for *example.com*'s IP address. Aside from returning *www.example.com*'s IP address, the local DNS server (i.e. BIND) will also cache *example.com*'s name server address. Therefore, when the user sends a DNS query for another address in example.com domain (e.g. *mail.example.com*), the local BIND server will directly ask for its IP from the cached example.com's name server, illustrated in Figure 2.

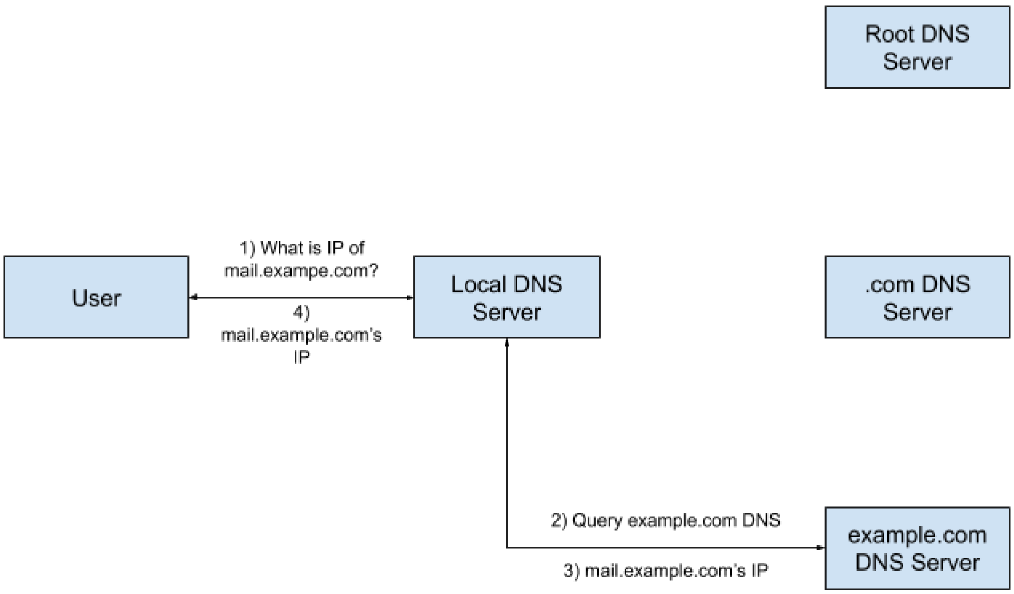


**Figure 2: DNS query process when example.com DNS Server is cached**

Imagine what would happen if an attacker forges DNS reply from example.com name server before the actual reply (i.e. message #3) reaches the local BIND server. The attacker can return arbitrary IP address of mail.example.com and furthermore overwrite example.com name server address that is cached at the local BIND server to a new fake address! (e.g. *ns.dnsattacker.net*)

However, this is more difficult than it sounds because DNS replies include a transaction ID which must match with the ID from the corresponding DNS query. Transaction IDs for DNS queries are randomly generated and not visible to attackers. Furthermore, sending DNS queries to resolve the same domain name again will not be effective because the first result will be cached. For example, if you send DNS query for *www.example.com* twice, the 2nd DNS query will not trigger DNS query to *example.com* name server. This makes it impossible for the attacker to forge another response for the same domain name until the cache entry expires and thus, makes the attack impractical.

**Attack Procedure:**

Here are outline for the attack procedure, to successfully poison your BIND server:

1. Query the BIND server for a non-existing name in example.com, such as *twysw.example.com*, where *twysw* is a random name.
   A. Since the mapping is not available in the BIND server's cache, it will send out a DNS query to the name server of the *example.com* domain.
2. While the BIND server waits for the reply from *example.com*'s name server, flood it with a stream of spoofed DNS replies, each with a different transaction ID, hoping one is correct.
   A. Even if the spoofed DNS reply fails, it does not matter because the next time, the attacker will query a different name.
3. Repeat steps 1-2 until the attack succeeds!

**Goal**:

Spoof DNS replies to overwrite example.com's name server's address to `ns1.dnsattacker.net`, `ns2.dnsattacker.net`. You need to add fake NS (i.e. name server) record in your spoofed DNS reply to overwrite example.com's name server address. Use starter code `part4_starter.py`.

Copy this into your local directory as follows

```
cp /share/copy/ece568f/lab4/part4_starter.py <path to lab4 directory>
```

You can run it with:

```
./part4_starter.py --dns_port <listen-on port> --query_port <query-source port>
```

**Notes:**

1. BIND is patched such that you do not need to fake IP address of the spoofed packet to match with the IP address of the remote name server (i.e. *example.com*). Spoofing IP address requirees sudo privilege which is not allowed in ECF workstations.
2. BIND is patched such that the transaction IDs are 8 bits instead of 16 bits. This greatly increases the chance of guessing the correct transaction ID. For the attack to work with 16 bit IDs, the attacking script need to use other techniques to send spoofed messages much faster, such as programming in faster languages, use multi-threading.
3. There are several ways to verify that the attack worked. **DO NOT** send DNS query to BIND asking what is the **NS** for *example.com*, before the attack has worked. It will make BIND cache **NS** for *example.com* and your attacking script will no longer work (BIND will simply disregard your spoofed message to change NS for *example.com*). It is fine to send the DNS query after you have verified that the attack has worked. Use the method we describe in the next section.
4. You can send a command through RNDC to enable logging queries received by your BIND server:

```
/share/copy/ece568f/lab4/bind9/sbin/rndc -c etc/rndc.conf querylog
```

**Attack Verification:**

Inspect the BIND server's cache to determine if it is poisoned:

```
/share/copy/ece568f/lab4/bind9/sbin/rndc -c etc/rndc.conf dumpdb -cache

less /tmp/dump.db // should be your cache dump location
```

A successful attack output should contain following:

```
; authauthority
example.com.                608379  NS      ns1.dnsattacker.net.
                            608379  NS      ns2.dnsattacker.net.
```

# Submission Instruction

You may work on this lab individually or in groups of two. Your code must be entirely your own original work. The assignment should be submitted by 11:59:59pm on Thursday, December 10th. Please submit a `README` file that contains your name(s), student number(s), and email(s) at the top, along with explanations for each part. You should submit `part1.txt`, `README`, `dnsproxy_starter.py` (for part2 and 3) and `part4_starter.py`.

```
submitece568f 4 README part1.txt dnsproxy_starter.py part4_starter.py
```

README format:

```
#first1 last1, studentnum1, e-mail1
#first2 last2, studentnum2, e-mail2

Part 1 Explanation:

...
```

Note: Please put your explanations in the `README`, not in the part#.txt files. If your files are not formatted as instructed, you may lose marks.

We have provided a submission checking script to help ensure that the automarker will process your files correctly:

```
/share/copy/ece568f/bin/check568_lab4 <directory of files>
```

Your solutions will be tested on the ECF machines. Please ensure that you test your solutions on these machines prior to submission.