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 [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 this remote attack lab, packet sniffing is not possible, so the attack becomes much more challenging than if it was conducted on the local network.

2 Lab Environment

We will setup the lab environment using one single physical machine, which runs three virtual machines. The lab environment actually needs three seperate machines, including a computer for the victim user, a DNS server, and the attacker's computer. These three VMs will run the provided `Ubuntu` image at http://www-internal.eecs.usma.edu/courses/cs482/setup/s3.ova. NOTE: Modern bind is robust against this attack so you need to use this specific VM image.

```
VM 1 (Attacker)         VM 2 (Victim)       VM 3 (Observer)
   10.172.x.16            10.172.x.14         10.172.x.15
       |                       |                       |
       |_______________________|_______________________|
       |                Virtual Switch                 |
       |_______________________________________________|
```

The figure above 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 `10.172.xxx.15`, the DNS Server's IP is `10.172.xxx.14` and the attacker machine's IP is `10.172.xxx.16`. However, in your lab, you will use your IP addresses, making it clear in your reports which address is for which machine.

- Client User IP ________________
- DNS Server IP ________________
- Attacer IP ________________

2.1 Configure the Local DNS server `Target`

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 is 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`. This file should already be present on your DNS server from Lab 2 setup (VM1, .4 IP). Please confirm the following option is present in 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. In this lab, this DNS server will not host the `example.com` domain, so please remove its corresponding zone from `/etc/bind/named.conf`. We recommend just commenting out both the blocks rather than deleting. NOTE: We will work solely with `example.net` for this lab.

Step 4: Configure a Fake Domain Name In order for the attack to work, the attacker needs their own domain name (reasons for this will become clearer after you see the explanation below). Since we do not own a real domain name, we can demonstrate the attack using our fake domain name `ns.dnslabattacker.net` and some extra configuration on `Target`. We will basically add the `ns.dnslabattacker.net`'s IP address to `Target`'s DNS configuration, so `Target` does not need to go out asking for the IP address of this hostname from a non-existing domain. In a real-world setting, the `Target`'s query would resolve to the attacker's server, which would be registered with a DNS registrar.

We first configure the victim's DNS server. 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";
};
```

****Note: Order of these zone entries matters! Please place this zone at the bottom of the file.

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 (`10.172.xxx.16`). Be aware that the format of the following contents can be messed up in the PDF file if you copy and paste.

```
$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 10.172.xxx.16
@ IN AAAA ::1
```

Once the setup is finished, if your cache poisoning attack is successful, any DNS query sent to `Target` for the hostnames in `example.net` will be sent to `10.172.xxx.16`, which is attacker's machine.

Step 5: 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 Attacker MAchine

We need to configure a malicious DNS server on `10.172.xxx.16`, so it answers the queries for the domain `example.net` once the attack is executed. Add the following entry in `/etc/bind/named.conf.local` on `10.172.xxx.16`:

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

Create a file called `/etc/bind/example.net.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.

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

2.3 Configure the User Machine

On the user machine `10.172.xxx.15`, we need to use `10.172.xxx.14` as the default DNS server. This is achieved by changing the DNS setting file `/etc/resolv.conf` of the user machine:

```
nameserver 10.172.xxx.14 # the ip of the DNS server you just setup
```

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

In this task, we use the domain name `www.example.net` as our attacking target. It should be noted that the `example.net` domain name is reserved for use in documentation, not for any real company. The authentic IP address of `www.example.net` is `93.184.216.34`, and it is 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.net`'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.net`'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.net`.

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 `Target`'s DNS cache, `ns.dnslabattacker.net` is set as the name server for the `example.net` 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.net"` from the user's machine, and the returned result must be a fake IP address.

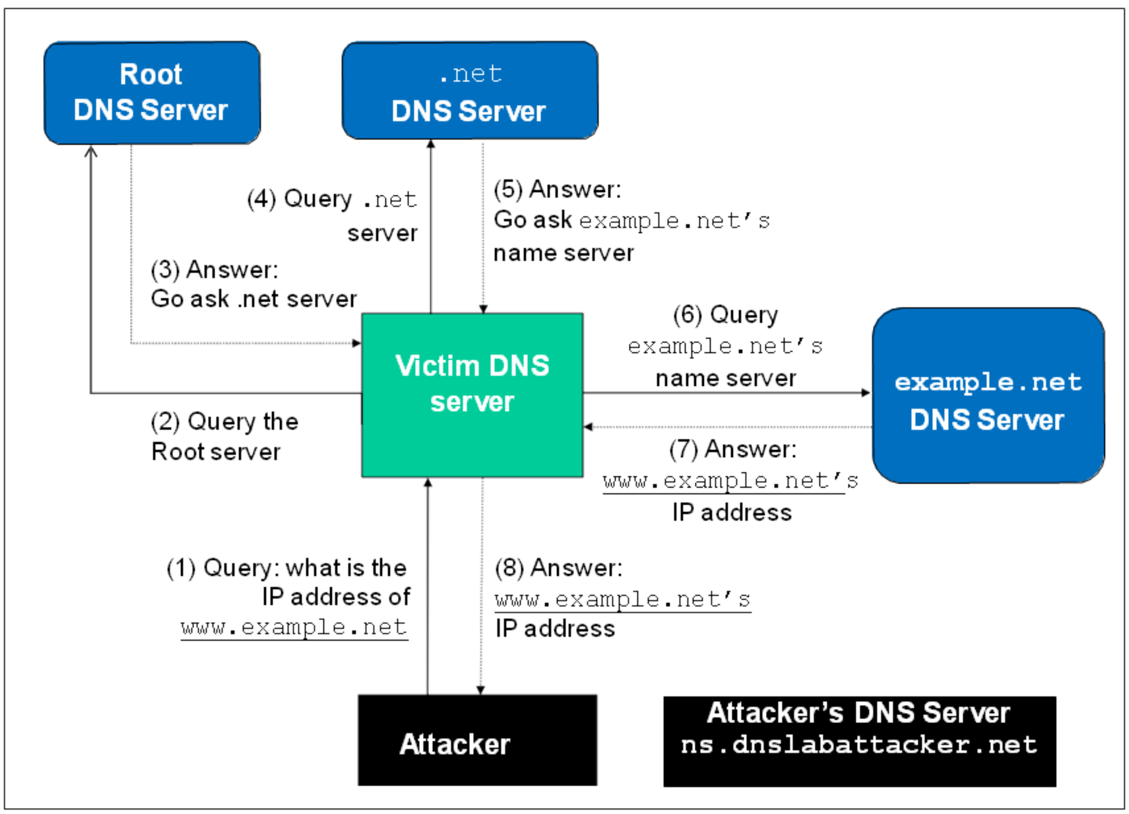

Figure 1: The complete DNS query process

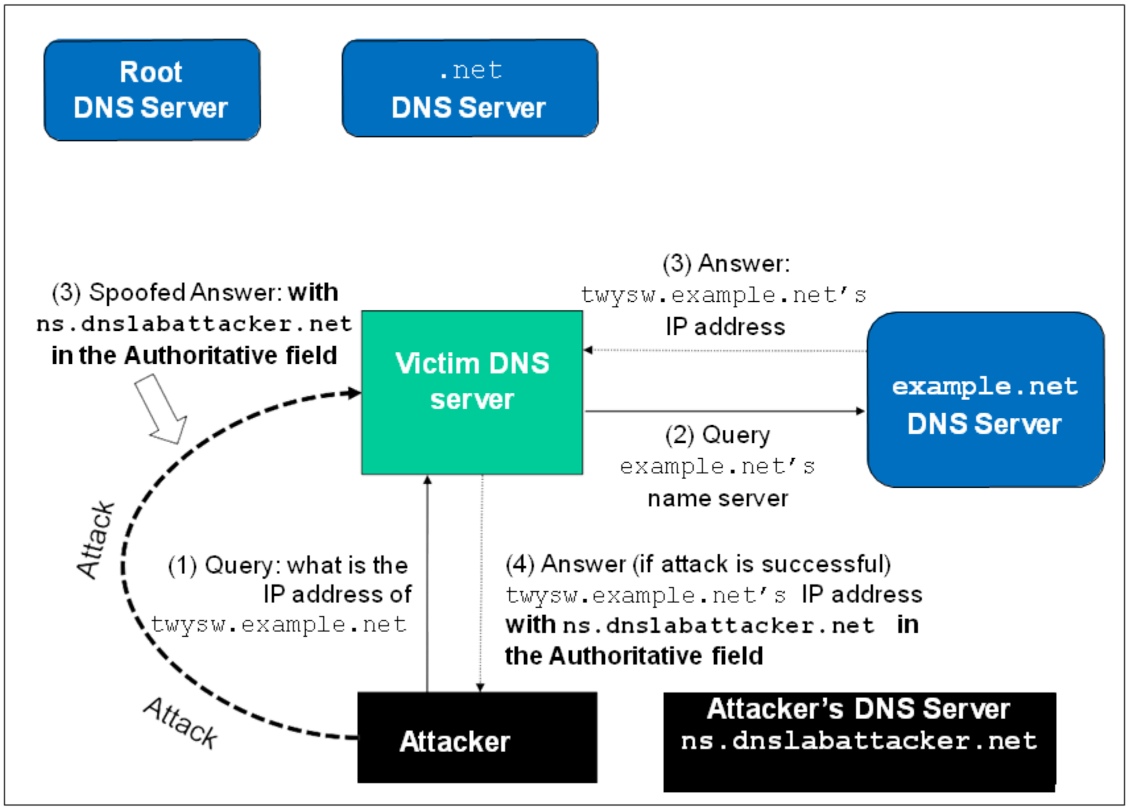

Figure 2: The DNS query process when `example.net`'s name server is cached

3.1 Task 1: Remote Cache Poisoning

In this task, the attacker sends a DNS query request to the victim DNS server, triggering a DNS query from `Target`. The query may go through one of the root DNS servers, the `.COM` DNS server, and the final result will come back from `example.net`'s DNS server. This is illustrated in Figure 1. In case that `example.net`'s name server information is already cached by `Target`, the query will not go through the root or the `.COM` server; this is illustrated in Figure 2. In this lab, the situation depicted in Figure 2 is more common, so we will use this figure as the basis to describe the attack mechanism.

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

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, we will try this attack method. The following steps with reference to Figure 2 outlines the attack.

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

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

3. While `Target` waits for the reply, the attacker floods `Target` 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.net`, the attacker also provides an "Authoritative Nameservers" record, indicating `ns.dnslabattacker.net` as the name server for the `example.net` domain. If the spoofed response beats the actual responses and the transaction ID matches with that in the query, `Target` will accept and cache the spoofed answer, and and thus `Target`'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 `Target` 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 `Target`'s DNS cache, the name server for `example.net` 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 `Target`'s DNS cache. Figure 4 shows an example of poisoned DNS cache.

Why did we have to create an additional DNS entry on `Target`*?* When `Target` receives the DNS query, it searches for `example.net`'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. This seperate query is why we created a DNS entry on the `Target` server. The domain name `dnslabattacker.net` does not exist in reality. We created this name for the purpose of this lab. If we did not create that entry `Target` will soon find out that the name does not exist, and mark the `NS` entry invalid, essentially recovering from the poisoned 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., `Target`) as its default DNS server. Please refer back to Section 2.3 for the instructions on how to do this. 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 `Target`:

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

   *****Note: This line should be added to the bottom of the `named.conf.options` file. Order matters!

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 `Target`. 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 `Target`'s DNS cache, and restart its DNS server. NOTE: Failure to this step will result in not getting the correct results. BONUS: write a detailed explanation why you must do this, see https://www.blackhat.com/presentations/bh-dc-09/Kaminsky/BlackHat-DC-09-Kaminsky-DNS-Critical-Infrastructure.pdf for further details.

Forge DNS Response Packets. In order to complete the attack, the attacker first needs to send DNS queries to `Target` for some random host names in the `example.net` 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. To make your life easier, we have provid code called `udp.c`. This program can send a large number of DNS packets. This program will work without modification, but feel free to modify this sample code to practice different variations against your `Target` DNS server.

1. To run the `udp.c` program:

   (a) Compile the program! Note: you should run this from wherever you saved the udp.c file.

   ```
   gcc -lpcap udp.c -o udp
   ```

   (b) Form the command line arguments

   ```
   sudo ./udp 10.172.XXX.15 10.172.XXX.14 10.172.XXX.16 199.43.135.53
   ```

   where,

   i. The first IP is the spoofed query source ip
   ii. The second IP is the victim DNS server
   iii. The third IP is the spoofed answer IP (malicious server); this could be whatever we want to host, i.e you could use 10.172.XXX.16 or another IP the attacker controls

Figure 3: A Sample DNS Response Packet

iv. The fourth IP is the spoofed response source IP, i.e. the IP of the DNS server to which the `Target` DNS server forwards requests. Here `199.43.135.53` is an instantce of a root server.

Check the `dump.db` file on the `Target` 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, `Target`'s DNS cache will look like that in Figure 4, i.e., the `NS` record for `example.net` becomes `ns.dnslabattacker.net`. To make sure that the attack is indeed successful, we run the `dig` command on the user machine (VM2) to ask for `www.example.net`'s IP address:`dig www.example.net`. NOTE: if you fail to clear the cache before launch the attack, you will notice that the attack will hijack the domain,`example.net` , but not the `www.example.net` subdomain. In your lab report, please provide an explanation why. See the following source:
`https://www.blackhat.com/presentations/bh-dc-09/Kaminsky/BlackHat-DC-09-Kaminsky-DNS-Critical-Infrastructure.pdf`

```
; additional
                        86400   DS      35886 8 2 (
                                        7862B27F5F516EBE19680444D4CE5E762981
                                        931842C465F00236401D8BD973EE )
; additional
                        86400   RRSIG   DS 8 1 86400 20150902050000 (
                                        20150823040000 1518 .
                                        odTrA1gfIetsQs9kaCuvGQBJgqLpoJJGP+Ig
                                        6f5qF+YeSUNj97U3lXDVuYDX7TzuKldZuPTZ
                                        DZ+kz+BkvvKkhueSOPXq9XzXq4+pEBaWkNMR
                                        jlQ3/N287Gog8sAjkEU5FzMZ6Kk9ly6xTMJd
                                        2UybcdaIX2rhmKquiE2suEfYo9M= )
; additional
ns.dnslabattacker.net.  604787  A       1.1.1.1
; authauthority
example.net.            604787  NS      ns.dnslabattacker.net.
; additional
\000\242\000\247\010.example.net. 0 \-ANY ;-$NXDOMAIN
; example.net. SOA sns.dns.icann.org. noc.dns.icann.org. 2015082218 7200 3600 1209600 3600
; example.net. RRSIG SOA ...
; example.net. RRSIG NSEC ...
; example.net. NSEC www.example.net. A NS SOA TXT AAAA RRSIG NSEC DNSKEY
; additional
\000\242\001\247\010.example.net. 0 \-ANY ;-$NXDOMAIN
; example.net. SOA sns.dns.icann.org. noc.dns.icann.org. 2015082218 7200 3600 1209600 3600
; example.net. RRSIG SOA ...
; example.net. RRSIG NSEC ...
; example.net. NSEC www.example.net. A NS SOA TXT AAAA RRSIG NSEC DNSKEY
; additional
\000\242\001\248\010.example.net. 0 \-ANY ;-$NXDOMAIN
; example.net. SOA sns.dns.icann.org. noc.dns.icann.org. 2015082218 7200 3600 1209600 3600
```

Figure 4: A Sample of Successfully Poisoned DNS Cache

4 Submission requirements

4.1 Partner Submission

Each team will provide one written lab report, answering each question, and providing evidence for each step taken to include tests. Be sure to include the time spent on the lab and document any external resources used.

4.2 Individual Submission

Each member needs to submit a detailed lab reflection. This includes

- How could you use this attack in a practical setting? I.E. if you wanted to steal someone's banking information, how would this attack help?
- List some of the challenges with this attack. Identify at least three major considerations (two were mentioned already). *HINT:* Consider the DNS server we spoofed in the response packets from when we ran the `udp.c` program. What would happen if we used a different DNS server further from the local area network?
- any challenging points or thoughts on what you found interesting during the lab
- time spent you personally spent and how much effort you put forth
- time your partner spent, and how much effort they put forth
- be sure document any external resources used.

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