Lecture 17: DNS and DNS Cache Poisoning Attacks

Lecture Notes on "Introduction to Computer Security"

by Avi Kak (kak@purdue.edu)

April 3, 2007

Goals:

- The acronym DNS and the multiple things it stands for
- The Domain Name System, Authoritative vs. Recursive Nameservers
- BIND
- Configuring BIND
- Caching Nameservers (and how to install them on your Linux laptop)
- DNS Cache Poisoning Attack

DNS

- The acronym **DNS** stands simultaneously for Domain Name Service, Domain Name Server, Domain Name System, and Domain Name Space.

- The foremost job of DNS is to translate symbolic hostnames into the numerical IP addresses. When you want to send information to another computer, you are likely to designate the destination computer by its symbolic hostname (such as **shay.ecn.purdue.edu**). But the IP protocol running on your computer will need the numerical IP address of the destination machine before it can send any data packets.

- Note that hostnames and IP addresses do not necessarily match on a one-to-one basis. Many hostnames may correspond to a single IP address (this allows a single machine to serve many web sites, a practice referred to as **virtual hosting**). Alternatively, a single hostname may correspond to many IP addresses. This can facilitate fault tolerance and load distribution.

- In addition to translating symbolic hostnames into numerical IP addresses, DNS also lists mail exchange servers that accept email for different domains. MTA's (Mail Transfer Agents) like **send-mail** use DNS to find out where to deliver email for a particular address. The domain to mail exchanger mapping is provided by MX records stored in DNS servers.

- Internet simply would not work without DNS. In fact, one not-so-uncommon reason why your internet connection may not be working is because your ISP's DNS server is down for some reason.

- You will have a lot more fun with your Linux machine (which could be a laptop) if you run your own DNS. I run a **caching DNS** on my Linux laptop. This allows me to talk to the outside world even when the ISP's DNS server happens to be not functioning. (Most of us are creatures of habit. I find myself visiting the same web sites on a regular basis. My email IMAP client talks to the same IMAP server all the time. So if my caching DNS has already stored the IP addresses for such regularly visited sites, it would not need to refer to the ISP's DNS.)

- Someone has said that DNS is one of the largest and most important **distributed databases** that the world depends on for serving billions of DNS requests daily for public IP addresses. What's even more, the DNS is an open database, in the sense anyone can set up a DNS server (for, say, a private computer network) and "plug" it into the network of worldwide network of DNS servers.

- Most DNS servers today are run by larger ISPs and commercial companies. However, there is a place for private DNS servers since they can be useful for giving symbolic hostnames to machines in a private home network.

- Of course, it must be mentioned that if a private home network has just four or five machines in, say, a 192.168.1.0 network, the easiest way to establish a DNS-like naming service for the network is to create a host table (in the `/etc/hosts`) file on each machine. The name resolver would then consult this table to determine the IP address of each machine in the network.

- However, if your private network contains more than a few machines, it might be better to install a DNS server in the network.

- On Linux machines, the file

  ```
  /etc/host.conf
  ```

  tells the system in what order it should search through the following two sources of hostnames-to-ipaddress mappings: `/etc/hosts` and DNS as, for example, provided by a BIND server. On my Linux laptop, this file contains just one line:

```
order hosts,bind
```

This says that we want the name resolver to first check the `/etc/hosts` file and then seek help from DNS.

- The basic idea of DNS was invented by Paul Mockapetris in 1983. (He is also inventor of the SMTP mail server.)

- Finally, if you change any of the network configuration files, such as, say, `/etc/hosts`, you would need to restart the network by

```
/etc/init.d/network restart
```

The Domain Name System

- For the **Domain Name System**, all of the internet is divided into a **tree of zones**.

- Each zone, consisting of a **Domain Name Space**, is served by a DNS nameserver that, in general, consists of two parts:

  - an **Authoritative DNS Nameserver** for the IP addresses for which the zone nameserver directly knows the hostname-to-IP address mappings; and

  - a **Recursive Name Server** for all other IP addresses.

- The authoritative nameserver file that contains the mappings between the hostnames and the IP addresses is known as the **zone file**.

- What distinguishes a **domain name space** is the symbolic domain name that goes with it.

- At the top level of the DNS tree, you have the root server that knows the IP addresses of the nameservers for the **top-level domains** such as 'com', 'org', 'net', 'edu', etc.

- The zone that is in charge of the 'edu' name would contain the mail exchange records for all of the subdomains of 'edu' and the hostnames and IP addresses of the nameservers in each of those subdomains. For example, the root nameserver for the 'edu' domain will contain the hostname and the IP address of the 'purdue.edu' domain and the nameserver for this subdomain.

- For example, if you invoke

  ```
  whois purdue.edu
  ```

  to find the whois server for the 'purdue.edu' domain (which happens to be 'whois.educause.net') and invoke

  ```
  whois -h whois.educause.net purdue.edu
  ```

  you can find out that the zone that corresponds to the 'purdue.edu' domain uses the following nameservers:

  ```
  NS.PURDUE.EDU                 128.210.11.5
  NS1.RICE.EDU
  PENDRAGON.CS.PURDUE.EDU       128.10.2.5
  HARBOR.ECN.PURDUE.EDU         128.46.154.76
  ```

The important thing to note here is that the primary nameservers for the 'purdue.edu' domain are located within the zone that corresponds to this domain.

- When a zone administrator wants to let another administrator control a part of that zone — that is, a part of the domain — that is within his or her zone of authority, he or she can **delegate** control for that subdomain to the other administrator. For example, if I was setting up a separate organization within Purdue for doing research in robotics and wanted to run my own nameserver for the subdomain `robotics.purdue.edu`, I'd need to approach the administrators in charge of the 'purdue.edu' domain and ask them to delegate the subdomain to me. I'd then create a nameserver with a name like `NS.ROBOTICS.PURDUE.EDU.` This nameserver would become the **SOA** (**Start of Authority**) (*which is the thing as the authoritative nameserver*) for all the hostnames within the `robotics.purdue.edu` domain. Subsequently, the main nameservers for `purdue.edu` would be authoritative nameservers for all hostnames within the `purdue.edu` domain but not including the hostnames in `robotics.purdue.edu`. With respect to the hostnames in `robotics.purdue.edu`, the main `purdue.edu` nameservers would be the recursive nameservers.

- Let's now see how someone working on a computer in Gambia can figure out the IP address for the hostname `shay.ecn.purdue.edu`.

The computer in Gambia would first contact the root server (as to how to reach the root server is stored in every network-enabled computer) and will receive from the root server the IP address of the recursive nameserver for the 'edu' top-level domain. The Gambian computer will then access the 'edu' domain nameserver with the same request as before and will receive the IP address of the nameserver for the `purdue.edu` domain. This being the authoritative nameserver for the `purdue.edu` domain will supply the IP address for the requested hostname.

- In case you are wondering about the multiple nameservers in the `purdue.edu` domain — especially the nameserver that is located at Rice — large domains typically have multiple nameservers for redundancy. These nameservers will generally carry identical information. Sometimes, the nameservers may be categorized as **master** and **slave** nameservers. Any changes to the nameserver record for a local domain would be made to the master nameserver and would then *get automatically synced over* to a slave via what is referred to as a **ZONE TRANSFER**.

- One or more of the slave nameservers may be located at geographically separated location for backup in case any man-made or natural disasters impair the operations of the local nameservers.

- **Master** and **slave** nameservers may also be referred to as the **primary** and **secondary** nameservers. Any additional nameservers for a domain would then be referred to as the tertiary nameservers.

- A primary nameserver is the default for a name lookup. A query will failover to the secondary (or to the tertiaries) if the primary is not available.

BIND

- **BIND** (Berkeley Internet Name Domain) is the most commonly used implementation of a domain name server (DNS).

- The BIND software package consists of the following three components

  - a DNS server (the server program itself is called **named**)
  - a DNS name resolver library
  - tools for verifying the proper operation of the DNS server

- BIND was originally written in 1988 by four grad students at the Univ of California, Berkeley. Later, a new version of BIND, BIND 9, was written from scratch by Paul Vixie (then working for DEC) to support DNSSEC (DNS Security Extensions). Other important features of BIND 9 include TSIG (Transaction Signatures), DNS Notify, nsupdate, IPv6, mdc flush, views, multiprocessor support, and an improved portability architecture.

- BIND 9 is maintained by ISC (Internet Systems Consortium), a not-for-profit US federal organization based in Redmond CA. ISC's principals are Rick Adams and Paul Vixie. (In addition to BIND, ISC has also developed the software for DHCP, INN (InterNetNews, a Usenet news server that incorporates the NNTP functionality), NTP (Network Time Protocol), OpenReg, etc. As an interesting aside, note that ISC also carries out an annual count of the total number of hosts on the internet by polling all the nameservers. The internet had 4,852,200 hosts in January 1995. In a span of ten years, this number has grown 100 fold. The internet had 439,286,364 hosts in July 2006.)

- Microsoft's products for network may or may not use BIND as maintained by ISC. Microsoft uses a DNS called MicrosoftDNS (derived from a WindowsNT port of BIND in early 1990's).

- Other DNS implementations include **djbdns**, **dnsmasq**, **MaraDNS**, etc.

DNS Cache

- The description I gave earlier for how a computer in Gambia might look up the IP address of a hostname is true in theory (but in theory only).

- In practice, if each one of the currently registered over 500 million computers around the world carried out a DNS lookup in the manner previously explained, that would place too great a burden on the root servers. The resulting traffic to the root servers would have the potential of slowing down the name lookup process to the point of its becoming useless.

- This brings us to the subject of caching the name lookups. To understand caching in DNS and where exactly it occurs, let's go back to the business of your computer trying to figure out the IP address associated with a hostname.

- Let's assume that the hostname that your computer is interested in is `www.nyt.com`.

- Do understand that it is usually not your computer as a single entity that is trying to carry out a DNS lookup. On the other hand, it is a client application such as the Internet Explorer, Firefox, a mail client such as sendmail, etc., that sends a query to a **name resolver** utility on a DNS nameserver. For example, when you are within the `purdue.edu` domain and you point your browser to `www.nyt.com`, the browser will send that URL to the name resolver on one of the nameservers of the `purdue.edu` domain. (The nameserver has to be running a program like BIND to be able to process the incoming request for name resolution.) If this is the first request for this URL received by the nameserver for `purdue.edu`, the nameserver will forward the request to the nameserver for the 'com' domain, and the name lookup will proceed in the manner explained previously. However, if this was not the first request for the name resolution of `www.nyt.com`, **it is likely that the local nameserver would be able to resolve the URL by looking into its own cache.**

- If the application software installed on your own computer is smart about domain name resolution, the various client applications (such as mail clients, web browsers, etc.) might maintain their own **DNS caches** usually with very short **caching times** (typically 1 minute but can be as long as 30 minutes) for the information stored.

- Additionally, the operating system always carries out some local name resolution before sending out a name resolution request to the nameserver of the local domain. At the very least, the operating system would be programmed to look up the information in `/etc/hosts` for any direct hostname-to-IP address mappings you might have placed there.

- The operating system may also maintain a local cache for the previously resolved hostnames with relatively short caching times (of the order of 30 minutes) for the information stored.

TTL

- When a DNS query for a given hostname is fielded by a authoritative DNS server, in addition to the IP address the server also sends back a time interval known as the TTL (Time to Live) for the response. The TTL specifies the time interval for which the response can be expected to remain valid. **What is stored in the cache is both the IP address and its associated TTL.** Subsequently, for all DNS queries for the same hostname made within the TTL window, the local name resolver will use the cached entry and the query will not be sent to the remote nameserver.

- The TTL value associated with a hostname is set by the administrator of the authoritative DNS server that returns the IP address along with its TTL. The TTL can be in units of minutes, hours, days, and even weeks. Ordinarily, an ISP nameserver will cache an IP address for a hostname for 48 hours.

- While DNS caching (along with the distributed nature of the DNS architecture) makes the hostname resolution faster, there is a down side to caching: any changes to the DNS do not always take effect immediately and globally.

- Earlier we talked about authoritative nameservers and recursive nameservers. On account of the explanation already provided, we may refer to an authoritative nameserver as a **publishing nameserver** and a recursive nameserver as a **caching nameserver**.

- A DNS query emanating from a nameserver is referred to as a **recursive query** when the local nameserver has to ask another nameserver in order to fulfill a lookup request.

Configuring BIND

- Linux machines most commonly run BIND for DNS.

- As already mentioned, the actual name of the BIND daemon server is **named**. How this nameserver daemon responds to a query depends much on a configuration file called **named.conf**. If you install BIND in your Linux machine, the pathname to the config file is likely to be `/etc/named.conf`.

- Slide 22 shows an example of a BIND configuration file.

- `named.conf` support C style (/* */) and C++ style (// to the end of line) comments in addition to the Unix style (# to the end of line) comments used in configuration files.

- `named.conf` file begins with what is known as an ACL declaration to define two access control lists. The acl 'dns_slaves' specifies that slave nameservers to be used in the **external view**. And the acl 'lan_hosts' specifies the group of hosts relevant to the **internal view**.

- If you are setting up a DNS server for a private 192.168.1.0 network, the external and the internal views refer to how DNS requests coming from outside the 192.168.1.0 intranet should be processed vis-a-vis the lookup requests emanating from within the 192.168.1.0 intranet.

- Next, the `named.conf` file will contain an **options** clause.

- The declarations made in the 'options' clause are the default values for the various fields. These defaults may be overridden in the individual zone files that will be located in the directory '/etc/namedb/'. Note that the name of this directory is also specified in the 'options' clause. Note the values specified for the listen-on field:

  ```
  listen_on {
      192.168.1.1;
      127.0.0.1;
  };
  ```

  This implies that the machine on which the **named** server is running is 192.168.1.1. This then also becomes the IP address of the interface on which named will be listening on. Note that the loopback address in IPv4 is 127.0.0.1 and the same in IPv6 is :: 1.

- Let's now talk about the 'controls' clause. To understand this clause, note that BIND makes available port 953 for remote administration of the nameserver. (BTW, **named** listens on port 53 for UDP requests for DNS service.) The 'controls' clause:

  ```
  controls {
      inet 127.0.0.1 allow {localhost;}
      keys { rndc-key; }
  }
  ```

  generates a TCP listen on port 953 (the default control port). If remote administration will not be used, this control interface can be disabled by defining an empty 'controls' clause:

  ```
  controls {}
  ```

- The acronym 'rndc' in the 'controls' clause stands for **remote name daemon controller** that is used for remote administration. We may think of 'rndc' as the remote administration utility whose operation is controlled by a secret key defined in the file `/etc/rndc.key`. The various parameters of this key are defined in `/etc/rndc.conf` configuration file. A new key can be generated by executing '`rndc-confgen -a`'.

- The 'inet' statement within the 'controls' clause specifies the IP address of the local server interface on which rndc connections will be accepted. If instead of 127.0.0.1, we had used the wildcard `""`, that would allow for the rndc connections to be accepted on all of the server machine's interfaces, including the loopback interface. The IP address that follows 'inet' can accept a port number if the default port 953 is not available. What follows 'allow' is the list of hosts that can connect to the rndc channel.

An Example of the `/etc/named.conf` Configuration File

```
acl "dns_slaves" {
    xxx.xxx.xxx.xxx;            # IP of the slave DNS nameserver
    xxx.xxx.xxx.xxx;            # same as above
};

acl "lan_hosts" {
    192.168.1.0/24;             # network address of your local LAN
    127.0.0.1;                  # allow loop back
};

options {                       # this section sets the default options
    directory "/etc/namedb";    # directory where the zone files will reside
    listen-on {
        192.168.1.1;            # IP address of the local interface to listen
     127.0.0.1;
    };
    auth-nxdomain no;           # conform to RFC1035
    allow-query { any; };       # allow anyone to issue queries
    recursion no;               # disallow recursive queries unless
                                # overridden below
};

key "rndc-key" {
    algorithm hmac-md5;
    secret "XXXXXXXXXXXXXXXXXXXXX";
};

controls {
    inet 127.0.0.1 allow { localhost; }
    keys { rndc-key; };
};

view "internal" {
    match-clients { lan_hosts; };    # match hosts in acl "lan_hosts" above
    recursion yes;                   # allow recursive queries
    notify no;                       # disable AA notifies
    // location of the zone file for DNS root servers
    zone "." {
        type hint;
        file "zone.root";
    };
    // be AUTHORITATIVE for forward and reverse lookup inside LAN:
    zone "localhost" {
        type master;
        file "example.local";
    };
    zone "0.0.0.127.in-addr.arpa" {
        type master;
        file "example.local.reverse";
    };
    zone "example.com" {
        type master;
        file "example.com.zone";
    };
    zone "0.1.168.192.in-addr.arpa" {
        type master;
        file "example.com.reverse";
    };

};

view "external" {
    // "!" means to negate
    match-clients { !lan_hosts; };
    recursion no;                       # disallow recursive queries
    allow-transfer { dns_slaves; };
                # allow "hosts in act "dns_slaves" to transfer zones
    zone "example.com" {
        type master;
        file "external_example.com.zone";
    };
};
```

- Every **zone** statement in the config file specifies a domain that it refers to. Here the "." is the root level domain for DNS.

- When 'type' in a 'zone' declaration is 'master' that means that our DNS server will be a primary server for that zone. Our DNS will also be authoritative for these zones. When the 'type' is 'hint', then the file named contains information on the root servers that will be accessed should DNS query not be answerable from the information in any of the zone files or from the cache.

- The zone file for a domain name like `127.in-addr.arpa` is for the `in-addr.arpa` domain names that are needed for **reverse DNS lookup**. Reverse lookup means that we want to know the symbolic hostname associated with a numerical IP address in the dotted-quad notation. An IP address such as 123.45.67.89 would be associated with an `in-addr.arpa` domain name of `89.67.45.123.in-addr.arpa`. The symbolic hostname associated with the IP address could be listed in a zone file whose name is something like `0.0.0.123.in-addr.arpa`.

- Note the 'match-clients' line in the 'internal' and the 'external' views. The internal view is for the LAN clients and the external view for clients outside the LAN.

- Note also the definition of `lan_hosts` at the beginning of the config file. The notation `192.168.1.0/24` is the **prefix length** representation for specifying a range of IP addresses. Our example notation says that the first 24 bits of the 32 bit IP address are supposed to remain constant for all the hosts in this LAN. In other words, the subnet mask for this LAN consists of 24 ones followed by eight zeros, that is 255.255.255.0. This implies that the network address for our LAN is 192.168.1.0 and the host addresses span the range 192.168.1.1 through 192.168.1.255. *The subnet mask tells you which portion of an IP address is the network address and which portion is reserved for the host addresses in a LAN.*

- Read the manpage on 'named.conf' for further information.

Setting up a Caching Nameserver on Your Linux Machine

A caching nameserver forwards queries to an upstream nameserver and caches the results. Here are the steps for installing a caching nameserver on your personal machine:

- Make sure you have the following five packages installed:

1. **bind** (includes the famous DNS server '**named**')
2. **bind-utils** (utilities for querying DNS servers about host information)
3. **bind-libs** (libraries used by the bind server '**named**' and the **bind-utils** package)
4. **bind-chroot** (tree of files which can be used as a chroot jail for bind) [See Slide 30 for what is meant by 'chroot jail'.]
5. **caching-nameserver** (config files for a simple caching nameserver)

- You can check whether or not each of the above packages is present by

```
rpm -q  packagename_listed_above
```

- Now edit the file

```
/var/named/chroot/etc/named.conf
```

  and add the following two lines to the global options section:

```
forwarders { xxx.xxx.xxx.xxx; xxx.xxx.xxx.xxx; };
forward only;
```

  where 'xxx.xxx.xxx.xxx' is meant to be replaced by the actual IP address of an upstream nameserver. The second line above causes the nameserver to forward all DNS resolve it is not able to resolve from its cache to the IP addresses listed.

- Make sure that the above file has permission 644:

```
chmod 644 /var/named/chroot/etc/named.conf
```

- Now check the syntax of the changes you made to the chrooted `named.conf` file by

```
named-checkconf named.conf
```

while you are in the `/var/named/chroot/etc/` directory. Shannon Hughes says that you should also check for error messages in `/var/log/messages`. (See the Shannon Hughes citation at the end of this document.)

- Now edit the `/etc/resolv.conf` file. Make sure it contains only the following string

  ```
  nameserver 127.0.0.1
  ```

  Remember this file gets automatically written over by DHCP. We obviously want to prevent that. See next step.

- To prevent `/etc/resolv.conf` from getting written over by DHCP, edit the following files

  ```
      /etc/sysconfig/network-scripts/ifcfg-ath0
      /etc/sysconfig/network-scripts/ifcfg-eth0
  ```

  Make sure that the following three lines are in each of these files

  ```
  BOOTPROTO=dhcp
  PEERDNS=no
  TYPE=Ethernet
  ```

  It is the 'PEERDNS=no' line that prevents DHCP from writing into the `resolv.conf` file.

- Now make the `/etc/resolv.conf` file only readable by

```
chmod 444 /etc/resolv.conf
```

- What is interesting is that some system processes are capable of overriding the above "read-only" permissions. To prevent any system driven process from overwriting `/etc/resolv.conf`, you may have to provide it with 'i' attribute by

```
chattr +i  /etc/resolv.conf
```

  To drop the 'i' attribute, invoke `chattr -i` on the file. See the manpage on 'chattr' for further details. [In my case, whenever I connected wirelessly to Purdue Air Link and started up the VPN service, my `/etc/resolv.conf` would get overwritten until I made the above fix.]

- Now start the chrooted nameserver by

```
service named start
chkconfig named on
```

  See Slide 30 for what is meant by chrooting a process.

- Now test the caching nameserver. This you can do by two consecutive applications of 'dig' to a domain name:

```
    dig nyt.com                 (The first time the answer will
                                 come back in, say, 100 msec.)


    dig nyt.com                 (The second time around, the same
                                 answer will come back in
                                 practically no time because now
                                 the answer will be supplied from
                                 the DNS cache.)
```

`dig`, which stands for Domain Information Groper, is a commonly used utility for interrogating and troubleshooting DNS nameservers.

- If instead of installing a caching nameserver on your Linux laptop, you want to install it on a desktop in, say, a `192.168.15.0/24` private network that is protected by a firewall, you would need to open up port 53 on the nameserver machine so the other machines in your private network can request DNS service from caching nameserver. You could do this with the following `iptables` commands:

```
iptables -A  INPUT  -s  192.168.15.0/24  -p udp  --dport  53  -j  ACCEPT
service iptables save
```

What does it Mean to Run a Process in a `chroot` Jail

- Ordinarily, when you run an executable on a Linux machine, it is run with the permissions of the user that started up the executable. **This fact has major ramifications with regard to computer security.**

- Consider, for example, a web server daemon that is fired up by a sysadmin as root. Unless some care is taken in how the child processes are spawned by the web server, all of the server's interaction with the machine on which it is running would be as root. A web server must obviously be able to write to local files and to also execute them (such as when you are uploading a form or such as when a remote client's interaction with the server causes a CGI script on the server to be executed). Therefore, a web server process running as root could create major security holes. **It is for this reason that even when the main HTTPD process starts up as being owned by root, it may spawn child processes as 'nobody'.** It is the child processes that interact with the browsers. More technically speaking, we say that the child HTTPD processes spawned by the main HTTPD server process are **`setuid`** to the user 'nobody'. The user 'nobody' has no permissions at all. (Because 'nobody' has no permissions at all, the permissions on the pages to be served out must be set to 755. Purdue ECN sets the permissions of public-web directory in user accounts to 750. That works because the HTTPD processes dishing out the pages are runs as 'www'.)

- Some people think that running a server process as 'nobody' does not provide sufficient security. They prefer to run the server in what is commonly referred to as the **chroot jail**.

- This is done with the 'chroot' command. This command allows the sysadmin to force the program to run in a specified directory and without allowing access from that directory to any other part of the file system.

- For example, if you wanted to run HTTPD in a chroot jail at the node '/www' in the actual directory tree in a file system, you would invoke HTTPD as

  ```
  chroot /www httpd
  ```

  All pathnames to any resources called upon by HTTPD would now be with respect to the node `/www`. The node `/www` now becomes the new '/' for the httpd executable. Anything not under `/www` will not be accessible to HTTPD.

- Note that, ordinarily, when an executing program tries to access a file, its pathname is with respect to the root '/'. But when the same program is run when chrooted to a specific node in the directory tree, all pathnames are interpreted with respect to that node.

- Therefore, you can say that 'chroot' changes the default interpretation of a pathname to a file. The default interpretation is with respect to the root '/' of the directory tree. But for a 'chrooted' program, it is with respect to the second argument supplied to 'chroot'. As a result, a 'chrooted' program cannot access any nodes outside of what the program got chrooted to.

Phishing vs. Pharming

- **Phishing** is online fraud that attempts to steal sensitive information such as usernames, passwords, and credit card numbers. A common way to do this is to display familiar strings like `www.amazon.com` or `www.paypal.com` in the browser window while their actual URL links are to questionable web servers in some country with weak cyber security laws. (You can check this out by letting your screen pointer linger on such hyperlinked strings in your spam email and seeing the URL that is displayed at the bottom of the browser.)

- In **pharming**, a user's browser is redirected to a malicious web site after an attacker corrupts a domain nameserver (DNS) with illegitimate IP addresses for certain hostnames. This can be done with a **DNS cache poisoning attack**.

- DNS serves that run BIND whose versions predate that of BIND 9 are vulnerable to DNS cache poisoning attacks.

- More commonly, it is the out-of-date BIND software running on old Windows based nameservers that is highly vulnerable to DNS cache poisoning.

DNS Cache Poisoning

- As mentioned already, by the poisoning of a DNS cache is meant entering in the cache a fake IP address for a hostname.

- What makes DNS cache poisoning a difficult (or, in some cases, not so difficult) exploit is the use of a 16-bit **transaction ID** integer that is sent with every DNS query. **This integer is supposed to be randomly generated.**

- That is, when an application running on your computer needs to resolve a symbolic hostname for a remote host, it sends out a DNS query along with the 16-bit transaction ID integer.

- If the nameserver to which the DNS query is sent does not contain the IP address either in its cache or in its zones for which its has authority, it will forward the query to nameservers higher up in the tree of nameservers. **Each such query will be accompanied with its own 16-bit transaction ID number.**

- When a nameserver is able to respond to a DNS query with the IP address, it returns the answer along with the transaction number so that the recipient of the response can identify the corresponding query. As long as the TCP or UDP port number, the IP address and the transaction ID from the remote host are correct, the reply to the query is considered to be legitimate.

- The DNS cache poisoning attack proceeds as follows:

1. The attacker identifies a **vulnerable recursive nameserver** for the attack. We will refer to this nameserver as the **target nameserver** — the target of the attack.

2. The attacker identifies the authoritative nameserver for the name whose IP address the attacker wants to hijack.

3. The attacker tries to slow down the authoritative nameserver for the name in question by some sort of a DoS attack.

4. The attacker creates a fake nameserver in the domain he controls. It could be running on the same machine that the attacker uses to mount the attack. The fake nameserver has

the same name (but obviously an IP address controlled by the attacker) as the authoritative nameserver for the name in question. For example, for the `purdue.edu`, one of the authoritative nameservers is `ns.purdue.edu`. So the attacker will create a host with the name `ns.purdue.edu` but with an illegitimate IP address.

5. The attacker sends a few hundred queries to target nameserver asking for the IP address of the domain name to be hijacked. The query for a domain like `purdue.edu` might look like

   ```
   purdue.edu. in A
   ```

   where 'A' stands for the authoritative zone. Such a query will, for example, be sent out by a command like

   ```
   nslookup purdue.edu
   ```

   To send a DNS query to a specific name server, you can invoke 'nslookup' as follows:

   ```
   nslookup  purdue.edu  harbor.ecn.purdue.edu
   ```

   where the second argument is the name of the nameserver to be used. (See the manpage for 'nslookup' for the various options with which it can be called.) (Another way to query DNS servers would be by issuing commands like '`host purdue.edu`' or '`host -v purdue.edu`'. The `host` utility is a simpler version of `nslookup`.)

6. Assume that the target nameserver receiving the queries is not the authoritative nameserver for the name in question and assume that the target nameserver does not possess a cache entry for the name in question. The target nameserver will therefore forward the request to either peer nameservers or nameservers higher up in the nameserver tree. *The important thing to note here is that each forwarded query generated by the target nameserver will carry its own Transaction ID. Any legitimate replies to that query much include that same Transaction ID.*

7. At the same time as the attacker is sending out the queries, the attacker's nameserver sends an equal number of phony replies to the target nameserver for the query. These replies are formulated to look as if they were sent from the authoritative nameserver. A phony reply may look like

```
Server: xxx.xxx.xxx.xxx
Address: xxx.xxx.xxx.xxx#53

Authority Section:
purdue.edu. IN NS
NS.PURDUE.EDU

Additional Section:
NS.PURDUE.EDU IN A
yyy.yyy.yyy.yyy
```

The phony reply includes an illegal IP address (yyy.yyy.yyy.yyy) in its last line for the legitimate nameserver for the `purdue.edu` domain. **Subsequently, the target nameserver will send all queries related to the `purdue.edu` domain to this IP address.**

8. Each phony reply packet will use a random Transaction ID. In order for the attack to be successful, for one of the spoofed reply packets, the Transaction ID, the source and destination IP addresses, and the source and the destination ports must match the legitimate recursive query packet from the target nameserver.

9. If the number of queries dispatched by the attacker to the target nameserver and the corresponding number of phony replies (each with a different random number as the Transaction ID) is large enough, the statistical odds of the target nameserver accepting a phony reply as one that is legitimate also go up.

10. As Stewart has stated so eloquently, "at this point all the attacker has to do is to win the race between the first successful collision of his spoofed transactions and the legitimate answer from the true authoritative nameserver. The race is already slanted in favor of the attacker; however, he could utilize other methods to gain an even bigger edge, such as flooding the true

authoritative nameserver with bogus packets in order to slow down its response." (See my Step 3 above.)

- Whether or not the attacker would succeed with a DNS cache poisoning attack depends on how deep an understanding the attack has of the pseudo-random number generator used by the target nameserver used for generating the Transaction ID numbers.

- Earlier versions of BIND did not randomize the transaction IDs; the numbers used were purely sequential. If a target nameserver is still running one of those versions of BIND, it would be trivial to construct a candidate set of Transaction IDs and to then send fake replies to the target nameserver's query about the name in question. Obviously, when the target nameserver randomizes its transaction IDs, the attacker would need to be smarter about constructing the packet flood that would constitute answers to the target nameserver's query.

- What increases the odds in attacker's favor is that BIND's implementation of the DNS protocol actually sends multiple simultaneous recursive queries for the same symbolic name that needs to be resolved. As mentioned by Stewart, this brings into play a mathematical phenomenon known as the Birthday Paradox that significantly increases the probability (to near 100%) of getting the target nameserver to accept one of the phony answers to its query with only a few hundred packets (instead of the tens of thousands previously believed to be needed). **(See Lecture 16 slides for a discussion of the Birthday Paradox.)**

- The harm caused by the Birthday Paradox is exacerbated by any weaknesses in the pseudo-random number generator used by the target nameserver. If you already know the transaction IDs that the target nameserver has produced so far and you can predict with a high probability the next transaction ID that the nameserver will use, you are making the life of the attacker that much easier.

- In addition to the transaction ID, as already mentioned, there is one more piece of information that the attacker needs when sending phony replies to the target nameserver: **the source port that the target nameserver uses when sending out its queries about the domain name in question.**

- The attacker can safely assume that the destination address used in the query packets issued by the target nameserver is 53 since that is the standard port monitored by nameservers. However, the source port at the target nameserver machine is another matter altogether. As Stewart has mentioned, "it turns out that more often than not BIND reuses the same port for queries on behalf of the same client. So if the attacker is working from an authoritative nameserver, he can first issue a request for a DNS lookup of a hostname on his server. When the recursive packet arrives, he can look at the source port. Chances are this will be the same source port used when the victim sends the queries for the domain to be hijacked..”

For Further Reading .....

- Cricket Liu and Paul Albitz, "DNS and BIND", 5^{th} edition, O'Reilly, May 2006.

- Joe Stewart, "DNS Cache Poisoning — The Next Generation," `http://www.lurhq.com/dnscache.pdf`.

- S. M. Bellovin, "Using the domain name system for system break-ins," Proceedings of the Fifth Usenix Unix Security Symposium, pp. 199-208, June 1995.

- Michal Zalewski, "Strange Attractors and TCP/IP Sequence Number Analysis," `http://www.bindview.com/Services/Razor/Papers/2001/tcpseq.cfm`, December 2002.

- Shannon Hughes, "How to Set Up a Home DNS Server," `http://www.redhat.com/magazine/025nov06/features/dns/`, November 16, 2006. [*The information regarding the caching nameservers was drawn primarily from this source.*]

- D. Atkins and R. Austein, "Threat Analysis of the Domain Name System (DNS)," RFC 3833, 2004.