DNS Security and Threat Mitigation: An Overview of Domain Name System Threats and Strategies for Securing a BIND Name Server

by Jeff S. Drake

Abstract

The Internet is a seemingly limitless source of information. It provides the power of collective knowledge and information to a vast array of users who access innumerable resources for countless reasons. These resources are typically accessed by using a human readable name designed to be easily remembered, thus increasing the usability of the resource. These human readable names, as the very term implies, are for the sake of the human users. Network devices, however, find each other by using a number, referred to as IP (Internet Protocol) addresses. The Domain Name System is the service that maps the human readable names to device specific IP addresses creating the user friendly nature of networked systems.

The Internet and millions of other networks are dependent upon the functionality of the Domain Name System. DNS is a complex, hierarchical system of distributed databases which are dependent upon each other to respond to queries by network users. The failure of this system at any level has crippling effects on network access. An infiltration of the DNS system can lead to disastrous consequences by directing unsuspecting users to network locations that are designed to steal their valuable information. Given the interconnected nature of economic, military and political communications, protecting this DNS structure from threats has taken on a new level of significance. BIND is the standard DNS server used on Linux and Unix systems. This document will first present an overview of the DNS architecture and name resolution process as well as describe common threats to DNS. Finally this document will outline some of the defensive configurations that can be implemented in BIND to help protect against some of these common threats.

Introduction to Name Resolution

In the 1960s the Advanced Research and Projects Agency (ARPA), a part of the US Department of Defense, helped to fund and thus initiate ARPANET. ARPANET was designed to build a network infrastructure connecting research institutes and government contractors (Liu & Albitz, 2006, p.1) Initially the ARPANET connected Stanford Research Institute, the University of Utah in Salt Lake City and two University of California locations (Chappell & Tittel, 2004, p.2-3). In the early days of computing, networking technology was proprietary and incompatible. The government or some institution may own or control several different networks but if each network used differing technologies then interoperability was problematic. (Chappelle &Tittel, 2004, p.3). The ARPANET project was intended at least in part to facilitate the sharing of research information and to promote the sharing of limited computing resources which at that time were mammoth machines typically supporting one or more dumb-terminals. Additionally the DoD wanted to create a network of communication that permitted dissimilar networks to communicate which eventually gave rise to the TCP/IP protocol stack and the desire to create a packet switched network of communication that could continue to function in the event of an atomic or nuclear war (Chappell & Tittel, 2004, p.2). The idea of functionality after nuclear attack may seem odd but the idea here appears to have been that a network designed to switch packets instead of depending on completed circuits would be a more logical choice because in a packet switching network data is divided into pieces called

packets and no two packets have to follow the same path to get to their destinations. Thus, as network segments fail, packets can be routed over different pathways.

This brief history sets the stage then for name resolution. ARPANET, although minuscule compared to the Internet, still required a simple way to access other systems. Computers deal in numbers. Everything to a computer at some level is a number. The computers themselves, like telephones or homes, have numerical numbers or addresses associated with them. However, referring to computers by complex decimal, hexadecimal or binary numbers is not a human friendly way to function. Typically we find it much easier to refer to names and not numbers so a system to resolve a name to a specific number was developed to perform this task. If the concept of name resolution seems a bit esoteric, consider a common usage of name to number resolution used today. Most cellphone owners have a list of names and phone numbers commonly referred to as an address or contacts list stored in their cellphones. The phone owner can simply scroll down a list, select a name (or with some phones simply speak the name) and the phone dials the associated phone number. In the same way, when you access a website over the Internet by a name there is a service that gives your computer the corresponding number, or IP address that corresponds to that computer. Remember, in the end, everything is a number to a computer. Connecting to a remote computer is like making a phone call;you must know the correct number. The name resolution technology places the burden of keeping up with a computers (logical) numerical address, called an IP address, on another technology and not the end user.

The ARPANET was originally made up of less than a thousand computers. As previously stated, this is minuscule relative to the number of devices on the Internet today. However, it still required a name resolution solution to make it reasonable to reference remote systems.The host file, as it is commonly called (although the name is actually 'hosts'), was the widely used mechanism for name resolution in the early stages of ARPANET. (Dean, 2004, p.183) Figure 1 shows a sample host file. Specifically, this sample host file is from a Red Hat Linux system. In Linux and Unix systems the full path to the file is /etc/hosts. Please note that the line numbers are not in the file; they are supplied here to make lines easy to reference.

```
1)  # Do not remove the following line, or various programs
2)  # that require network functionality will fail.
3)  127.0.0.1              localhost.localdomain localhost
4)  10.100.100.30      server1.linuxclass.edu      server1
5)  10.100.100.31      server2.linuxclass.edu      server2
6)  10.100.100.26      olddog.linuxclass.edu       olddog
7)  10.100.5.120       mail.linuxclass.edu          mail
8)  10.100.5.120       dogct
9)  10.100.90.30       server3.linuxclass.edu      server3
10)206.81.220.22       www.rei.com                  rei
```

Figure 1

Lines one and two are merely comments as indicated by the # symbol whereas line three is the fist line read by the system. This line is a reference to the machine itself and the IP address is referred to as the loopback address. Lines four through ten provide name resolution. The first column is the IP address, the numerical address assigned to the computer. The second column is the name of the system and the third column an alias. The alias column is not mandatory; it is simply a way of creating a shorter nickname for the system referenced on the line so that a user does not have to type in olddog.linuxclass.edu. The user using this particular /etc/hosts file can simply type in

olddog instead.

The original host file used on ARPANET was obviously much larger but from a computer's standpoint searching through a few hundred lines in a simple text file is not an overly laborious task. Periodically network administrators would download a copy of the host file from a central location on ARPANET maintained by Stanford Research Institute's Network Information Center. As the Internet grew, this centrally maintained database was found to be problematic.

First, this global host file was dependent upon network administrators sending in timely updates for the file. It was dependent upon administrators ever more frequently downloading the updated file. And perhaps the larger of the problems was that it did not scale well. It is not difficult to see that as ARPANET morphed into what is now the Internet which is made of millions and millions of computers with continuous additions, the host file became obsolete. Even if a text file with millions of entries in it were feasible for a computer to search through it would be impossible to keep up to date (Liu & Albitz, 2006, p. 3). A solution was needed for name resolution which gave birth to the Domain Name System or DNS.

What is DNS?

The host file was a single growing text document that had to be centrally maintained. DNS is a distributed database. DNS data is distributed across literally thousands of DNS servers where zone data maintenance, the job of updating, adding and deleting resolution information, is distributed as well. A DNS server is a computer running a program referred to generically as a name server. The name server is responsible for the name resolution process. The administrator is responsible for maintaining the DNS data on that name server. However, unlike the centralized administration where a single entity was making changes for the entire ARPANET, now the name server administrator only makes changes for his own part of the DNS name space. In distributing the information that was formerly maintained in a host file it also gave rise to the more robust feature set in modern day DNS.

A host file was a single document requiring uniqueness on the part of every host name in the file (Liu & Albitz, 2006, p. 3). It was a flat data structure much like MAC addressing for those familiar with MAC versus IP addressing comparisons. The Internet population explosion made it even more important to have a naming resolution system that scaled well and obviously a flat text file does not scale well. DNS provides this scalability by implementing levels in the naming scheme referred to as domains (Brain, 2006, section 1). To understand the concept of domains, remember that DNS is a distributed database. It is a single, monstrous, data set that has been chopped into pieces and these pieces have been placed on thousands of different name servers. Each domain can potentially have dozens of child domains. Figures 2 and 3 give graphical depictions of parts of the DNS structure.

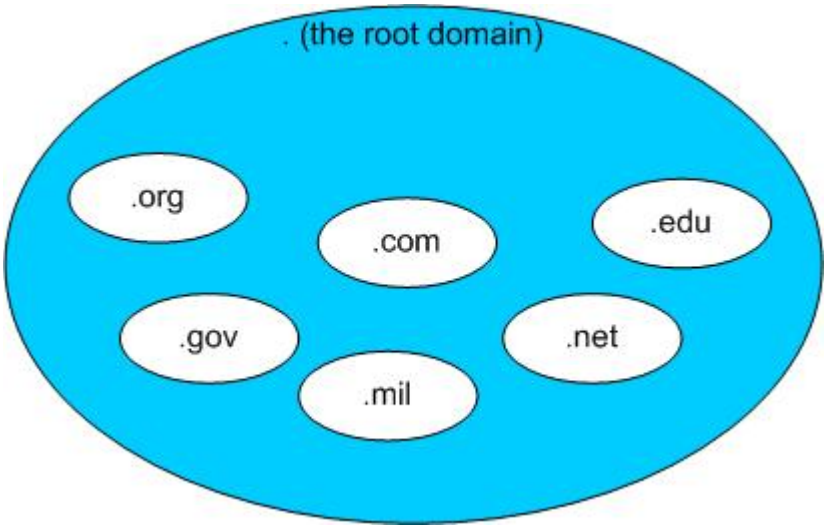

Figure 2

Figure 2 refers to the root domain and some of the top-level domains or TLDs. The root domain as in a file system is the starting point, or root of the DNS database. Information about the root domain is maintained by root DNS servers. Circles are used here to help the reader understand that what is represented by a domain is not a computer but a group of computers or other devices, or in the case of the root domain, other sub-groups. As Liu and Albitz as well as others have described it, the DNS database is much like a file system. File systems are made up of directors (a.k.a. folders) and files. Folders allow us to organize files. In the same way, domains are like directories in that they allow us to create and organize sub-domains (like sub-directories) and hosts names (like files). And, like in a file system, only the names in a domain have to be unique. So I can have a host named 'mail' in the ubuntu.com domain as well as one named 'mail' in the redhat.com domain. Figure 3 depicts this principle as well as that of a top-level domain having second-level domains beneath it.

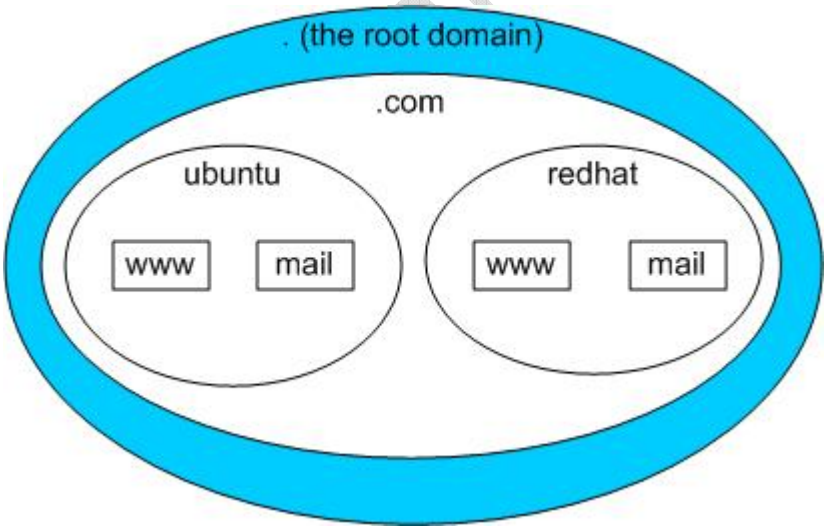

Figure 3

Figure 4 shows a common web address that a user might enter and labels each section. This is called a full qualified domain name or FQDN. A fully qualified name contains at least a top-level

domain, second-level domain and host name. (http://webopedia.com/TERM/F/FQDN.html) There can be more than two levels in a domain name. So there could be a web address of ftp.devel.ubuntu.com where 'devel' represents a sub-domain of ubuntu and ubuntu is a sub-domain of 'com' and finally 'com' is a sub-domain of the root domain typically noted as simply a '.' that most normal web users would never need to reference. It is theoretically possible to have up to 127 domain levels in a name (Brain, 2006, section 1).

Figure 4 Fully Qualified Domain Name

Each domain represents a piece of the database and each piece, or domain has its own name server which is responsible for maintaining correct entries that resolve the names to IP addresses. These entries are called resource records, which will be shown later. Each name server is also responsible for answering queries regarding names in the domain or domains it is responsible for maintaining. One name server may maintain information for more than one domain. And, one domain may have more than one designated name server for load balancing and security reasons discussed later.

The name servers that resolve queries sent to the "." domain are managed by ICANN which stands for Internet Corporations for Assigned Names and Numbers (Kato & Murai, 2001, p.1). The root name servers are at the top of the DNS hierarchical distributed database. They act as the proverbial "wise man on the mountain" that all other name servers reference to ask a question about a name resolution for some other domain. The root name servers then refer the requesting name server to the appropriate top-level name servers which are also managed by ICANN ("What is ICANN", 2004, section 1). Second level domain name servers are managed by the organizations or individuals who purchase the domain names or someone they delegate such as their ISP. The name resolution process will be reviewed after a brief description of the dominant name server software.

What is BIND?

When we refer to a name server it is easy to think about the "box", that is the computer doing the work. The name server, more specifically, is the piece of software running on the computer, on top of the operating system, that is responsible for resolving queries for a particular domain. The most commonly used name server on the Internet is BIND (Berkeley Internet Name Domain) ("ISC BIND", 2004, section 1). BIND is primarily run from UNIX or Linux platforms but it has been ported to Windows ("ISC Bind", 2004, section 1). Microsoft has its own name server product which is tightly integrated with its Active Directory product. A newer name server packages has been written called 'djbdns' named after its author Daniel J. Bernstein who also authored the email product 'qmail' (Langfeldt, 2001, p.4). The name server this paper is concerned with specifically is BIND version 9.3 but many of the principles and configuration examples will be applicable to earlier

versions of BIND or other name server products.

As stated earlier name servers and their administrators have several responsibilities which include but are not limited to the following:

1. Resolving queries for name resolution for a particular domain that it maintains.
2. Maintaining up to date resource record information for hosts in the domain.
3. Distributing copies of the domain information to slave name servers which help to spread the name resolution workload around and help in security.
4. Tell resolvers (those machines issuing the queries) how long they should consider the answer to be good information, specifically how long they can cache the answer to the query.

The most commonly associated responsibility is that of resolving queries. Figure 5 helps to illustrate the name resolution process.

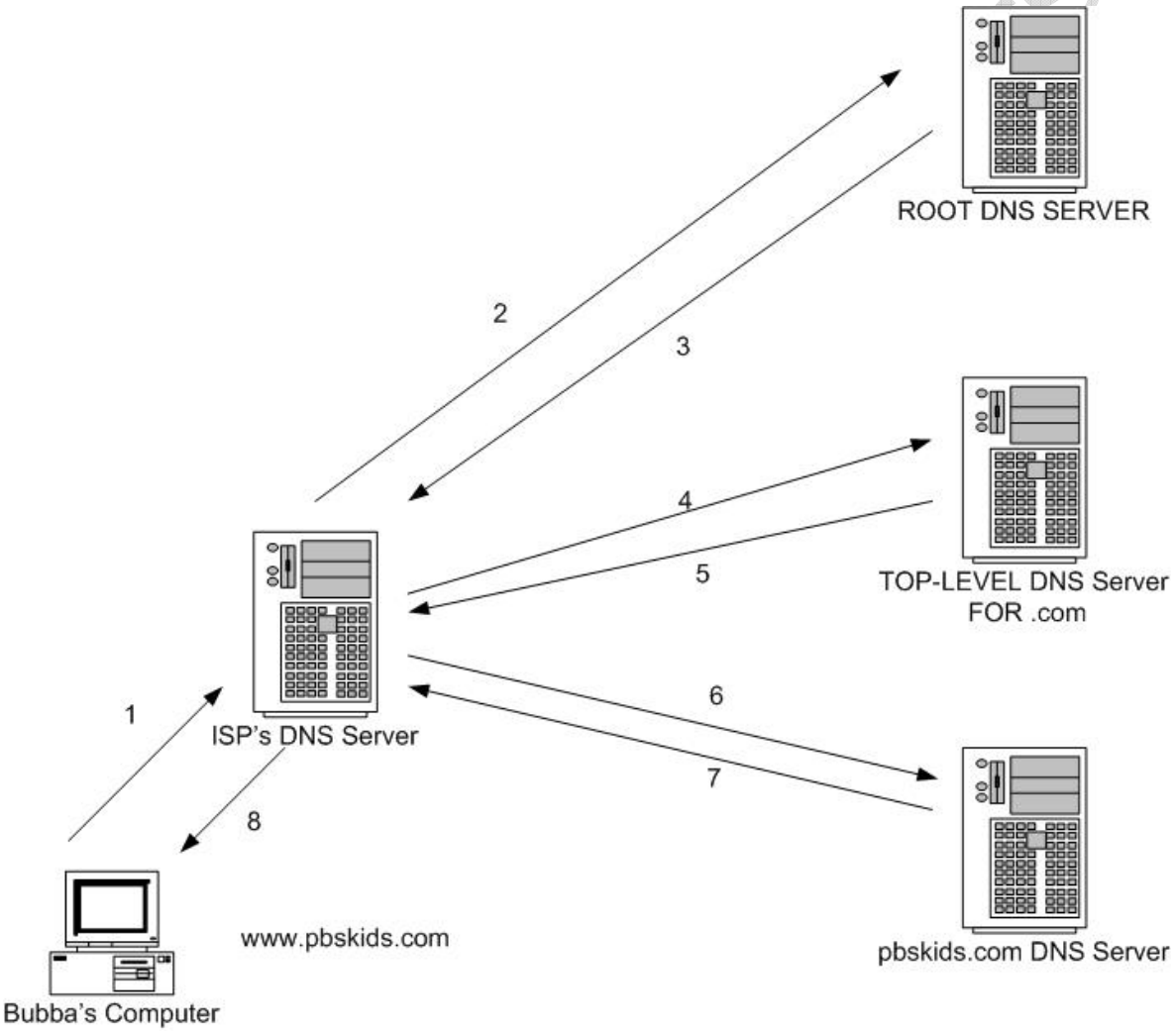

Figure 5 Sample DNS query diagram

In step one, little Bubba sits down at his home computer. He wants to go to 'www.pbskids.com' where he can play some kid-friendly games on line. So he brings up his favorite web browser

Firefox 2.0 and types in 'www.pbskids.com' and hits the enter key. At this point the journey begins. Little Bubba's computer needs the IP address of the web server 'www.pbskids.com' so it can actually get to the website, so it checks its own DNS cache, its own memory, to see if it knows the answer to its own question. Apparently Bubba hasn't been to this website recently so his computer does not have that information cached. So, it sends a query to his ISP's name server, also referred to as a DNS server. The ISP's DNS server checks its own cache but does not find the answer there either so in step 2 it sends a query to a root dns server. Every name server has a file that contains a list of all of the root nameservers. The root nameserver does not maintain the information about the pbskids.org domain but it does maintain information about where a .com level (top-level) nameserver is located. So, in step 3 the root DNS server sends the IP address of a .com top-level DNS server. In step 4 the ISP's DNS server then sends a message to the top-level DNS server for .com which does not know about www.pbskids.com but it does know the IP address of the DNS server that maintains the pbskids.org domain. So, in step 5 the .com top-level DNS server sends the IP address for the pbskids.org domain nameserver. The ISP's DNS server, depleted at this point, sends a message in step 6 to the nameserver that actually has information specific to the pbskids.org domain and it returns the ip address of the host www.pbskids.com in step 7 to the ISP's DNS server. Finally, the ISP's DNS server sends little Bubba's computer the IP address and he can now visit www.pbskids.org, portal to cool kid stuff. The IPS's DNS server is now exhausted and it goes out for a cheese danish and glass of pineapple-orange juice amazed that this whole process works. The process is described by numerous authors such as Liu and Albitz 2006, Dean 2004, Olzak 2006 and a host of other resources. Now comes a shift in focus from explaining DNS operation to securing the DNS process, specifically on a BIND nameserver running on a Linux system. But first, a review of basic BIND configuration files structure.

/etc/named.conf and Zone Data Files

The two primary configuration files for BIND are the named.conf configuration file and the zone data files which store the actual names and their corresponding IP addresses. Here are sections of a named.conf file from BIND running on a Red Hat Enterprise Linux 4 server. Keep in mind our purpose is not to define every line of the named.conf file but to give an overview of key sections often configured by a BIND administrator. Each section has added comment lines noted by the leading // with the text in bold for effect.

```
//Sample subsections of an /etc/named.conf file


//Defines access control lists by name.
//The names can be referred to in global or zone specific configuration sections
acl "labs" { 10.100.80.0/24;10.100.90.0/24;10.100.100.0/24;10.100.60.0/24;10.100.170.0/24;
 };
acl "nonlabs" { 10.100.70.0/24;10.100.120.0/24;
};

//Defines global options that affect all zones that this nameserver services
//For instance, queries are being allowed to the name server only from the IP addresses
//identified as part of the "labs" acl
options {
   directory "/var/named";  //this line is the working directory for the BIND service
   allow-query { labs; };     //this line indicates that only those subnets defined above will receive responses to
queries
   allow-recursion { labs; };
   allow-transfer {labs; };
```

```
    dump-file "/var/named/data/cache_dump.db";
        statistics-file "/var/named/data/named_stats.txt";
    /*
     * If there is a firewall between you and name servers you want
     * to talk to, you might need to uncomment the query-source
     * directive below.  Previous versions of BIND always asked
     * questions using port 53, but BIND 8.1 uses an unprivileged
     * port by default.
     */
     // query-source address * port 53;
    forwarders { 10.100.50.210; };
};

//The zone section identifies the name of a zone this particular nameserver
//services, indicates that it is the master and not a slave nameserver for this zone
//and identifies the name of the file where the resource records for the zone are stored.
zone "linuxclass.edu" IN {
    type master;
    file "linuxclass.zone";
    allow-transfer { 10.90.90.5;10.80.90.5;
    };
};

//This oddly named zone is the reverse lookup zone.  Reverse lookup zones
//allow the nameserver to resolve IP address to names instead of the traditional
//name to IP address resolution that is commonly associated with DNS
zone "100.100.10.in-addr.arpa" IN {
    type master;
    file "100.100.10.in-addr.arpa";
};

//This zone declaration indicates that the server is a slave server for
//the online.linuxclass.edu domain
//notice that this is a sub-domain of linuxclass.edu in hierarchy
//but they represent two different data sets which are only logically related

zone "online.linuxclass.edu" IN {
    type slave;
    file "dogpound.zone.sl";
    masters { 10.150.990.5;
    };
}
```

This is a sample zone file. Specifically, we will say that this is the linuxclass.zone file referred to in the above sample named.conf file zone declaration. Note that the explanation comments are preceded by ';' which is the standard comment indicator for the zone file.

```
;The TTL value is the length of time a host querying this nameserver can cache the
;DNS information it gets from this server.  This is a global configuration
;and can be over ridden for different zones in the named.conf file
$TTL    12h

;This is the Start of Authority information section
;The SOA is considered the "best source of information for the data within this zone" (Lui and Albitz, 2006 p. 57)
;The other information here is primarily for slave nameservers for this zone.  The time here is in seconds
;but could be noted using letter abbreviations for day, week, hour and minute like the $TTL value
```

```
linuxclass.edu.   IN    SOA    rha-server.linuxclass.edu. root.rha-server.linuxclass.edu.    (
25              ; serial
28800       ; refresh
14400       ; retry
3600000  ; Expire
43200       ; Negative
)

;This line indicates the name server for the zone in question
linuxclass.edu.    IN    NS    rha-server.linuxclass.edu.


;A records are simple name to IP resolution records
bigdog          IN    A    10.100.100.30
ftp               IN    A    10.100.100.40
web             IN    A    10.100.100.50
printer         IN    A    10.100.100.11
linuxserv       IN    A    10.100.100.30
maddog          IN    A    10.100.100.27

;A cname record is a nickname pointing to another server
;so in this case you could refert to 10.100.100.27 as www or maddog
;and still reach it
www          IN    CNAME    maddog

;MX records indicate the email server or servers that
;email for that domain should be delivered to
linuxclass.edu.        IN    MX    10    linuxserv.linuxclass.edu.
```

These are just a handful of example configuration lines for a named.conf file and a zone data file. Langfeldt, Lui and Albitz, Boran and the BIND 9 manual accessible at http://www.bind9.net/manuals are all good sources for BIND file configuration steps and examples. Often times your BIND installation may include configuration file examples and help files as well.

Defining the Threat

Now that a foundation has been laid let us move to the issue of DNS security. Initially DNS security may seem less important because it does not necessarily seem to be a database with confidential or financial information in it. However, your DNS server is the service that directs both internal and external machines towards those resources on your computers. If DNS is compromised, clients can be directed from your site to an illegitimate site. Consider this example.

Imagine that you are a major Internet auction service provider. You provide individuals and businesses a website where they can post items for sale and allow others to bid on them. You also provide financial transaction services to assist them. Suppose someone was able to compromise your DNS information and even though people were typing in your URL, www.gimmeyourmoney.com, they were actually being redirected to my website, www.letmestealyourpassword.com. My site looks exactly like your site and it has the same login screen for usernames and passwords. Unfortunately, the user enters in his/her username and password and now I can collect your customer's information and use it to buy stuff from myself very quickly and make lots of money to pay my attorneys who are going to try and keep me out of jail.

This is just one example of compromising your DNS data and using it for malicious purposes. This particular type of practice is referred to as Pharming. As a side note this may sound very similar to a practice used by spammers to acquire user information referred to as Phishing. The goal is the same, but the method employed is different.

Using the work of Householder and King, Liu and Albitz, Ollmann, Plante, US-Cert and Evers a brief list of common DNS threats has been compiled below.

1. DNS cache poisoning - At any point in the DNS architecture, the DNS cache of a nameserver (or even a simple workstation client) may be poisoned with the wrong information thereby redirecting unsuspecting users to incorrect websites as in the example above.

2. Inherit weaknesses in a particular version of BIND - As with any software, updates are in part designed to overcome flaws in previous versions; older versions of BIND are more susceptible in general to certain types of attacks. Plante (p. 9) mentions specifically the problem of transaction ID prediction. Older versions of BIND made it easier to predict these ID numbers thus easier for a person wanting to "poison" a server's DNS cache by responding as if it is an authoritative name server.

3. Denial of Service attacks - Although not unique to DNS servers DoS attacks can potentially cripple the entire Internet if root name servers are not accessible

4. Illegitimate Zone Transfers - If master and slave nameservers are not secured it is entirely possible for would-be intruders to acquire an entire copy of your DNS information, including internal DNS information and thereby have a partial or perhaps entire picture of your internal IP structure.

5. Search Engine "Page Ranking" - Although not directly related to BIND per se this is an interesting naming threat issue as well as that of "autocompleters" in web browsers. Ollman specifically notes these. Page ranking can be manipulated so that key words bring up Pharming sites that unsuspecting users may click on assuming they are the appropriate site.

6. Compromising the BIND daemon - If the BIND daemon is compromised, an attacker may be able to access other resources on the system as the root user. (Those not familiar with Linux systems should note the daemon for the BIND service is called 'named' and that root is the name given to what would generally be referred to as the administrator account.)

At their core, DNS threats as they relate to BIND are at some level concerned with acquiring or spoofing DNS data. The data itself and the service that BIND provides are the targets. If some techniques can be employed to make the server unavailable I can deny web servers from a practical standpoint. If a nameserver ends up with corrupted information either in its cache or in its data files, it then hands out incorrect information to resolvers, those machines performing queries. Generally speaking, most of the defense mechanisms we will discuss will focus on mitigating these threats and configuration examples will be based upon BIND version 9.x and Red Hat Enterprise Linux. However, the OS on which BIND runs is less the issue than the version of BIND you are running.

DNS Defense Tactics

Upgrade BIND

As indicated above, older versions of BIND are more threat susceptible as is the older version of most any software. Evers indicates that a study performed by Kaminsky found that of 2.5 million

nameservers evaluated, a quarter of a million were found to be "potentially vulnerable" with one of the issues being servers that use older versions of BIND. If you purchase your Linux software then the supporting company will provide you with some sort of updating service that allows you to update your software, including BIND, to newer versions as it makes them available themselves. So for example, if you use Red Hat Enterprise Linux you can use the up2date program, similar to yum, to update BIND and its dependencies. However, even supported Linux purchases do not necessarily make available the latest versions of software when they are released. If you desire to upgrade BIND on your system, an upgraded version can be obtained from http://www.isc.org/index.pl?/sw/bind/ . According to Boran (2001), version 8 was the most commonly used version at the time of his writing. This was several years ago but that data coupled with the more recent Evers article leads to the conclusion that BIND updates are still a pervasive and easily rectified problem. Boran also offers instructions on compiling BIND on Unix systems but for those using different Linux versions use of .rpm files already constructed offer a simpler installation solution. Installation techniques using yum or Ubuntu's synaptec package management system also simplify the process of updating. It is important to make sure that you understand the dependency issues if you are upgrading BIND. If you are using an rpm based system the command

rpm -qR bind

will display a list of bind related dependencies for your current version of BIND. If dependencies are not dealt with properly your BIND server will not restart after an upgrade.

Hide Your BIND Version

Since differing versions have different weaknesses, hiding your BIND version from potential attackers may provide some benefit. Liu and Albitz (2006, p. 288) give the following example command that can be used to determine the BIND version running on your computer's configured DNS server.

dig txt chaos version.bind.

Similarly, Microsoft's support site provides instructions on how to perform this same operation using the nslookup command from a Windows machine. To prevent your BIND nameserver from responding to this type of query the following configuration line can be added to the global options section in your /etc/named.conf file if you are running BIND version 9.3.0 which is the latest BIND version.

```
options {
   directory "/var/named";  //this line is the working directory for the BIND service
   version none

};
```

If you are running a version older than 9.3.0 but higher than 8.2 then this option will work.

```
options {
   directory "/var/named";  //this line is the working directory for the BIND service
  version "not disclosed";
```

};

One problem inherent in this example is that for 8.2 and higher if you advertised this type of response you probably indicate that you are not running 9.3.0 but are running greater than 8.2.

Limit Transfers

We have briefly touched on the issue of master and slave nameservers but for purposes of this discussion figure 6 is included to help understand this concept.

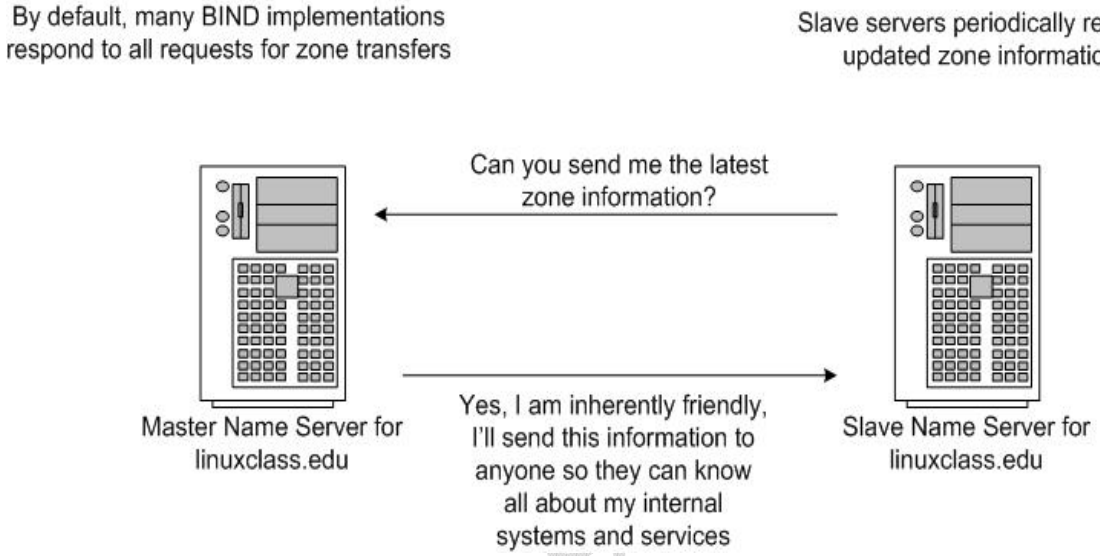

Figure 6

Again, there are several reasons to have more than one nameserver for a particular domain. Common reasons are for load-balancing, fault-tolerance and security. Security in part because it allows you to position your name servers strategically to limit exposure of the master nameserver to the outside world. (Having multiple name servers and positioning them in such a way that protects your internal DNS from the outside world doing lookups is a defense strategy in and of itself.)

As the diagram illustrates, many nameservers are configured to respond to any and all requests for zone transfers. By default, the majority of name server applications allow zone transfers to any other system (Plante p.5). This in essence means that everything in your zone data file as described above is going to be sent to who ever requests it. This gives them a tremendous view of your network and potentially what services you are running and what systems those services are running on. So if a zone transfer occurs and an attacker sees a resource record such as

ftp IN A 10.100.60.22

he or she knows that an ftp server and thus a typically less secure service is running there.

To help control the transfer of zone data there are a number of steps that can be taken. The

simplest is to limit zone transfers to specific IP addresses as seen in the following example.

```
zone "linuxclass.edu" IN {
    type master;
    file "linuxclass.zone";
    allow-transfer { 10.90.90.5;10.80.90.5;
    };
};
```

In the example above, the allow-transfer statement identifies what IP addresses will be allowed to receive zone transfers from this zone. Keep in mind this is for this particular zone. If the name server is responsible for different zones then each zone would need to be configured with allow-transfer statements for the appropriate slave servers. This IP based prohibition is not the only strategy that may be employed to prevent unwanted zone transfers. There are obvious problems with a solution that is circumvented simply by someone spoofing an IP address. It would be possible for someone to spoof the IP address of the master. Then, the slave name server(s) might contact the impostor master and receive completely false zone data that redirects clients to alternative locations. So, a solution that goes beyond simple IP based restrictions may be necessary. One of these options is referred to as TSIG.

Transaction Signatures (TSIG)

As Householder and King and others point out, TSIG allows nameservers to "cryptographically authenticate and verify zone data." Simply stated, TSIG is a mechanism that nameservers can use to make sure that the data they are receiving is the data that was originally sent and that it was sent by the host they requested it from. We have already stated that part of the problem with simple IP based restrictions is that it is too easy for impostor machines to pose as legitimate sources or recipients. However, TSIGs rely on the use of shared secret keys to "authenticate and verify" the data that is being transmitted.

The shared secret key principle is fairly straightforward. Each host has a key that allows it to encrypt and decrypt data. If a slave server requests a zone transfer from a master server and it is using TSIG then the master server responds with a zone transfer that has been "signed" using the secret key. When the slave server receives the message it decrypts the signature with its key. If this procedure works then it guarantees that both parties have the same shared key and ideally, unless the key has been compromised, that each name server is a legitimate participant in the information exchange.

Using work from Boran, Liu and Albitz, Householder and King and the BIND 9 Administrators Reference Manual available at http://www.bind9.net/manuals we can construct some TSIG examples.

First to generate the key the dnssec-keygen command can be utilized.

```
[root@ns-linuxclass named]$ dnssec-keygen -a hmac-md5 -b 128 -n HOST linuxclass.edu.

Klinuxclass.edu.+157+26140
```

In this example the -a indicates the algorithm, -b is the key length, -n is the type of key being generated and linuxclass.edu. is the name of the key. The output Klinuxclass.edu.+157+26140 is the name of the file where the key is located. The file is created in the present working directory. It

is actually two files by the same name but with a different extension. One ends in .key and the other ends in .private. The key is located in both files but the format of each file is different. The .key file is in a resource record format.

Now that a shared key value has been generated, both the master and slave name servers would need to have this information included in their named.conf files. Keep in mind this is a shared secret as opposed to public-private key pairs. The administrator of the servers will need to put this shared key value on both devices utilizing some method that does not potentially compromise the key value. The named.conf entry would look like this:

```
key linuxclass.edu. {
   algorithm hmac-md5;
   secret "Xwi7EPLIhO1ZEitBQToJIg==";
};
```

Then, the name servers must be told what servers utilize this key pair so they each will expect communications from the specified server to use that particular key. Keep in mind that name servers may service multiple zones and may be master and slave for different domains. The named.conf entry would be the following:

```
server ip_of_other_server_using_same_key {
   keys { linuxclass.edu. ;};
};
```

It should be noted here that the secret key value is just that - secret, and should not be included in a file that can be read by the "other" access class in Linux. So if your named.conf file is publicly readable this secret key information should be placed in a separate file and referred to using the "include" statement often seen at the end of default /etc/named.conf files. The include statement allows the administrator to include data in the named.conf file that is located in another file. The include statement seen identifies the file. The appropriate statement would be the following:

```
inlcude "/etc/Klinuxclass.edu.+157+26140.key"  //this asssumes that the .key file is located in the /etc directory
```

The statement identifying what server is using the shared key does not contain secret shared information so it could be in a file with more liberal read permissions. Be sure to check the permissions on your /etc/named.conf file. The work is not done yet for TSIG configuration. Thus far, we have simply indicated what the key is and which servers will be using it. Nothing has been configured to tell the name server what to use the key value for. Liu and Albitz state explicitly that "[i]n BIND 8.2 and later and all BIND 9 nameservers, we can secure queries, responses, zone transfers and dynamic updates with TSIG." (2006, p286) We have seen in previous examples how the allow-transfer directive can be used to identify hosts by IP address. Similarly the same directive can be used with the name of a key. This can also be used with allow-query and allow-update directives as well. An example would look like this:

```
zone "linuxclass.edu" IN {
   type master;
```

```
    file "linuxclass.zone";
    allow-transfer { linuxclass.edu. ;
    };
};
```

This statement restricts zone transfers to systems with the same shared key and thus increases the level of integrity of our zone transfer transactions beyond that offered by IP based statements. Understand that in the allow-transfer statement both a key and an acl can be listed but it is an "or" type statement. You cannot require both the key and IP address (Boran 2001 p.18) Although beyond the scope of this document, TSIGs can be used in the dynamic update process as well. Dynamic updates permit other systems to update DNS resource record information. Many readers may be familiar with this type of event if you operate in a Microsoft Active Directory environment where domain workstations receiving their IP addresses via DHCP are allowed to update their corresponding resource record in the MS DNS server. Dynamic updates are somewhat controversial in and of themselves and beyond the scope of this document but know that TSIGs can be used to add integrity to this arguably necessary functionality inside of a corporate network if DHCP is relied upon. It should also be stated that although TSIG is appropriate for dynamic updates it is more so designed for updates between nameservers and not between an enterprise of hosts due in part to the impracticality of managing a shared secret amongst potentially hundreds if not thousands of nodes (Ateniese and Mangard , 2001, p.3). An additional option in verifying DNS response validity has risen that does not depend upon shared secret information. It is referred to as DNSSEC.

DNSSEC

"DNSSEC is intended to protect DNS clients from forged DNS data" (Huston 2006, p.3). As mentioned earlier, one of the vulnerabilities in the DNS process is the possibility of delivering illegitimate DNS information to other hosts, thereby sending them to the wrong, potentially harmful destination. Perhaps even worse is the delivery and acceptance of illegitimate information by other nameservers that then disperse this information to their clients and poison DNS cache everywhere. TSIG examples described above help to prevent this in the master-slave nameserver relationship but since we cannot deliver our shared secret key to every conceivable host that may need resolution information it is a solution that does not necessarily scale well due to the private-private nature of the key.

The obvious albeit not so simply implemented solution is some sort of public key infrastructure. For those not aware of public key functionality, we give a brief and simplistic overview follows. (Perhaps painfully simplistic to the cryptographers in the crowd.) The idea behind public key, sometimes referred to as public-private key, encryption methods is that two keys exist. One is used to encrypt data. The other is used to decrypt data. And unlike a simple algebraic expression, one should not be able to determine the value of the encrypting key known as the public key simply by possessing the decrypting key known as the private key. The private key is kept, no big surprise, private, whereas the public key is freely distributed because possession of it does not give away any secret information about my private key. It is then possible that I as the public key holder can encrypt some information and send it to you knowing that only you can decode the information. Thus I can send data like credit card information across a public network like the Internet and know that even if the data were intercepted it cannot be decrypted. In the case of talking to a secure webserver my computer requests that server's public key which my computer

uses to encrypt my credit card information. The question then arises, what if someone intercepted my communication and sent me his/her private key instead and I ended up sending this person my encrypted credit card data. This information went across the internet encrypted but now when the cyber thief captures it he can decrypt it because I encrypted it with his key thinking it was the legitimate website's key. How can I be sure that the key belongs to the site I think I am requesting it from? This requires a third party to vouch for the authenticity of the public key I am receiving which in web server world is referred to as a certificate authority such as Verisign. The third party is able to vouch for the legitimacy of the public key through a different series of encryption events.
This whole process can be reviewed in more detail at http://computer.howstuffworks.com/encryption1.htm, in Liu and Albitz (2006 p. 322) and Huston (Aug 2006 p.2). It is this type of public-private architecture that is implemented in the new DNSSEC standard.

The public key architecture can work in a reverse manner as well by allowing the private key to the encrypting agent and the public key the decrypting agent. DNSSEC uses a variation of public-private keys to ensure the validity of the query response that a resolver is receiving. The idea being to prevent someone from being able to poison DNS cache or other stored DNS data, potentially redirecting uses to dangerous sites or simply diverting traffic away from the legitimate site. This technique requires the use of new resource records introduced for DNSSEC. Drawing on the work of Liu and Albitz (2006 p.324-334), Huston (August 2006 p.4-6), the BIND 9 Administrators Reference Manual available at http://www.bind9.net/manuals, a RIPE tutorial at http://www.ripe.net/disi/dnssec_howto/ and Arends et al. in RFC 4034 we can describe some of these new resource records and give the reader an idea of how they relate to each other.

The DNSKEY record is the DNS resource record holding the zone's public key. Before we create the DNSKEY record we must first have the public-private key pairs generated. The command is almost identical to the one used previously for TSIG. It would look like this to generate a key pair for the linuxclass.edu zone.

dnssec-keygen -a RSASHA1 -b 1024 -n ZONE linuxclass.edu. (this command is based on BIND 9.3.2, older versions do not support RSASHA1 as the encryption type)

Two files are again created but one contains the public, the other the private key value. The corresponding DNSKEY resource record in the associated linuxclass.edu zone file would look something like this:

linuxclass.edu. 86400 IN KEY 256 3 5
an_extremelylong_key_value_generated_by_above_command_located_in_.key_file

The 86400 seconds indicates a TTL for this record of one day, the 256 is the ZONE key indicator, 3 is always 3 in DNSSEC because it is a holdover from earlier DNSSEC versions and 5 indicates the RSASHA1 algorithm which is mandatory at least in BIND 9.3.2. Additional algorithms can be used as well in case some of the hosts interacting with your DNS server require a different algorithm. This key could be placed into the zone file using an "include" statement somewhat like this:

$INCLUDE *name_of_my_zone_signing_public_key_file.key*

Keep in mind the function of this key pair. The private key is used to create signatures for the DNS

information (the resource records) stored on our server. These signatures act as verification that the data came from the legitimate DNS server just like our signature uniquely identifies us.

The RRSIG is a Resource Record Signature. Each Resource Record set has an RRSIG configured for it. Resource Record set (RRset) is defined as "RRs in a DNS ZONE that share a common name, class and type." (Huston August 2006, p.4) The RRSIG is used to store the zone's private key signature as noted the RRSIG is for groups of records with common type, name and class. A sample RRSIG taken form RFC 4034 follows.

```
host.example.com. 86400 IN RRSIG A 5 3 86400 20030322173103 (
                    20030220173103 2642 example.com.
                    oJB1W6WNGv+ldvQ3WDG0MQkg5IEhjRip8WTr
                    PYGv07h108dUKGMeDPKijVCHX3DDKdfb+v6o
                    B9wfuh3DTJXUAfI/M0zmO/zz8bW0Rznl8O3t
                    GNazPwQKkRN20XPXV6nwwfoXmJQbsLNrLfkG
                    J5D6fwFm8nN+6pBzeDQfsS3Ap3o= )
```

The purpose of this record is to act as a signature for some Resource Record Sets. This could be all of the A (address) records that point to

maddog.linuxclass.edu. So this is a signature that verifies that the resource record information for maddog.linuxclass.edu is coming to your from the correct DNS server and that it is getting to you in the correct form.

The NSEC record is used to validate negative responses. Strangely enough the NSEC record is a validation of a "no, that domain name doesn't exist here". So if someone requests snortmore.linuxclass.edu they are requesting a record that does not exist. That would require the DNS server to send back a "That name does not compute" type message. But how does the recipient of the error know the error is legitimate? This record is a signature that can be used to validate the negative response. Creating a sample NSEC record from the zone file used an example at the beginning of this document would look like this:

```
linuxclass.edu.   NSEC   ftp.linuxclass.edu. NS SOA MX RRSIG NSEC DNSKEY
```

If creating all of these records seems like an overwhelming task some of your anxiety may be reduced by the fact that there are BIND commands that do it for you. If you are thinking that all of these new resource records are going to radically increase your zone file size you would be correct. The zone file size and the responses get larger which means more overhead on the server. This means no more buying the cheap stuff for your DNS server hardware.

To add to the mayhem of a DNSKEY record that defines your new public key, a RRSIG record created for all RRsets and NSEC records to validate your "it ain't here" responses, we add one last key into the arena at this point. First, let's review once again the idea behind DNSSEC. The purpose of DNSSEC is for those requesting DNS data from a nameserver to be able to verify that the data they are receiving is coming from whom they requested it and that the data arrived as it was sent. This is accomplished by signing all of the data that comes from the nameserver with a digital signature that was created using a private key. (The super secret kung fu command that does this will be shown shortly.) When DNS data is requested, the recipient also gets a copy of the public key which he/she can use to decrypt, verifying the signature thus certifying the data. But, how do we know if this public key just received is the real public key of the DNS server I think I

requested it from? For websites this is taken care of by the certificate authority that vouches for the key you are receiving. The DNS hierarchy does not have a CA apparatus, yet in reality, the eventual idea behind DNSSEC is that each parent domain will act as the certifier for all of it subdomains. This is not a reality yet, so no long discussion will ensue here. But, there is a way to make DNSSEC work now using a trusted-keys statement in your named.conf file. It should be noted that this description assumes a single domain with no child domains.

Previously the command used to generate the zone's public-private key pair was seen. This specifically is referred to as the zone signing key because it will be used to sign all data in the zone. There is also a key signing key that is used to sign the actual zone key. So the nameserver's public key will have a digital signature certifying it as well. Insane isn't it? This key is generated with the following command.

```
dnssec-keygen  -f KSK -a RSASHA1 -b 1024 -n ZONE linuxclass.edu.
```

This command is identical to the previoius dnssec-keygen command generating the zone signing key with the exception of the "-f KSK" option which stands for key signing key. A reference to it is also placed in the zone database file as seen with the zone signing key. The public key that was created is also referenced in the /etc/named.conf file with a trusted keys statement such as this one found in a DNSSEC tutorial at http://www.ripe.net/disi/dnssec_howto/#chapter.introduction. I strongly recommend reviewing this tutorial along with other references for this document before attempting your DNSSSEC implementation.

```
 trusted-keys {
 "example.net." 257 3 5 "AQO8VL6u4R3BopupRb9p0Nsns2Sy3+YDlu//TG+zWRJ+ppbhTGbhxtI8
                1vbDNeFnnGHdP6TWimibhwnqaG5D8XVoMRk5A2E/al8/8DmyHu2fqtTt
                y0MZHHZJDKCUep+nJnQLxESdbFhHKmBZEzN9Lb3clKcnHSXDWP2qYP4S
                cqX4uQ==";
 };
```

This may seem a bit odd in this single domain example because the nameserver vouches for itself. For now this is a workaround. It reality, the ideal implementation as stated before is that the root zone would vouch for .edu information and the .edu zone would vouch for linuxclass.edu information, etc., etc. They would do so by using a DS record stored in their zone files which acted as a verifier of child nameservers. As stated this has not been implemented but could be implemented within your domain tree if you had parent and child domains.

Last but not least are the commands that take all of these keys and sign the zone data. That command would look something like this.

```
dnssec-signzone -o linuxclass.edu. linuxclass.zone
```

Here, the '-o' indicates that 'linuxclass.edu' is the domain being signed and that the name of the zone file is 'linuxclass.zone'. The private key file for the zone is presumably, at least while this is command is being executed, in the same directory as the zone file which with this command example would be the current working directory. The output would be a file called 'linuxclass.zone.signed.' The /etc/named.conf file would then need to be edited to change the zone file name for linuxclass.edu zone.

```
zone "linuxclass.edu" IN {
   type master;
   file "linuxclass.zone.signed";
   allow-transfer { linuxclass.edu. ;
   };
};
```

Regardless of how lengthy and cumbersome this DNSSEC overview may have been it is a spattering of the detail, theory and implementation of it. DNSSEC is still relatively new and arguably not entirely mature yet. It is obviously not widely implemented in the DNS hierarchy as evidenced by the fact that the root and the vast majority of top level domains are not digitally signed as of yet.

After a discussion of TSIG and DNSSEC all other topics seem somewhat benign but before we end there are a few more simple but still necessary defense strategies that can be implemented to help protect your nameserver.

Run BIND (named) As Unprivileged User

As Unix and Linux admins know, services run as a particular user account on that system and thus have the access levels of this user. If your namserver daemon, named, is running as user root and is compromised then effectively the attacker has taken control of a service with root privileges which gives him/her greater permissions on the system. In most Linux implementations this is no longer a problem simply because the default installation procedure creates a user account called named and the named daemon runs as that user account.

Run BIND in a chrooted jail

Another threat to a compromise named daemon is file access. Traditionally the home directory for the named daemon has been /var/named. However, it is possible to create an alternative file system that mimics the before default file system and locks the daemon inside of this file mimicked file system structure making it more difficult for a compromised daemon to get outside of the default directory system and violate other system files. Again, like running as an unprivileged users, with many BIND installations there is an installation option that creates this chrooted jail environment for you but may seem a bit odd because at least in Red Hat Enterprise Linux you end up with a file structure like this.

```
/var/named/chroot
                    /dev
                    /etc
                    /var/named
```

The root of the file system from the perspective of the jailed daemon is actually four layers deep into the file system and contains only a small, re-created subset of the real file system. For an example on how to do this manuall see Boran's, Running the BIND9 DNS Server Securely. (2001)

Restrict Who Can Query Your Nameserver

By using acls as seen in previous examples BIND admins can restrict those users that are allowed to send queries globally or to a particular zone that the nameserver services. Globally queries can be controlled by a directive in the global options section of the named.conf file.

```
options {
```

```
    directory "/var/named";
    allow-query { labs; };

};
```

As seen in this example in the options section queries have been limited to those IP subnets defined in the labs acl. Alternatively, queries can be restricted on a per domain basis with an entry like the following.

```
zone "linuxclass.edu" IN {
    type master;
    file "linuxclass.zone";
    allow-transfer { 10.90.90.5;10.80.90.5;
    };
    allow-query { labs; };
};
```

Single Purpose DNS Servers

It is not always possible, but restricting your DNS servers to performing only DNS task as well as having multiple DNS servers responsible for different name resolution tasks is another defensive as well as load-balancing and fault tolerance strategy. First, restricting DNS servers to being just DNS servers means that the admin can have a system with minimal services running and thus reduce its workload and more importantly for security reasons, its attack surface. Simply stated, the less things your DNS server is doing the harder it can work at being a DNS server and the fewer services that are open to attack.

If more than one machine is at your disposal, then having DNS servers that respond to different types of requests and that are potentially placed in different parts of your network structure offers additional security as well. For instance you may choose to maintain one or more DNS servers for your internal clients. These servers restrict queries to just internal clients and you can allow recursive queries. A recursive query is typically what your workstation is issuing. In our earlier example Bubba's workstation sent a recursive query to his ISP's DNS server meaning that if the ISP's DNS server did not know the answer it would go and find it for him. For your internal clients this is necessary for them to reach external sites. However, for those on the outside of your network who only need to obtain specific IP addresses for your domain recursive querying is not necessary, simple iterative queries will suffice. An iterative query is the type of query sent from Bubba's ISP's DNS server to the other DNS servers on the Internet. This was a simple "can you tell me" query and the Internet DNS servers respond back with what they know but do not try to look elsewhere for the answers. Thus, a DNS server setup to respond to these external queries would not been threatened with cache poisoning because it is never going to look for information on someone else's behalf; it is simply going to answer queries with information it holds.

BIND and Firewalls

A final note on BIND and we shall end the insanity. You may have noticed in our initial discussion of BIND the following lines in the named.conf file.

```
/*
    * If there is a firewall between you and nameservers you want
    * to talk to, you might need to uncomment the query-source
    * directive below.  Previous versions of BIND always asked
    * questions using port 53, but BIND 8.1 uses an unprivileged
    * port by default.
```

```
*/
// query-source address * port 53;
```

As the comments explained, earlier versions of bind always used port 53 to DNS queries and responses. However, newer versions of BIND use dynamic ports for this purpose so firewall administration may not be as simple as opening port 53 to the nameservers IP address. By uncommenting this line it ensures that query and response traffic from this BIND server if from port 53.

Conclusion

Other than the fact you are probably tired of reading about BIND at this point, there are a few generalizations we can make. First, DNSSEC, is a complex measure to implement. It is advised that any attempt to implement it be done so in a test environment first after thorough review of documentation starting with the references for this paper. Second, if DNSSEC is not a reasonable step to take for your situation there are still some simpler strategies that can be implemented to help reduce the threat to your DNS information. TSIG offers a reasonably straightforward means of protecting data exchanges between your name servers. Next, although some have suggested its replacement (Walfish, Balakrishnan and Shenker)' it is unlikely that DNS is on the way out any time soon. Understanding DNS from at least a theoretical level and from an implementation level for network admins and engineers is essential. And finally, regardless of how much you have just digested about DNS and DNS security the amount left to consume is phenomenal. This document is merely a snowflake in the blizzard.

References

Arends, R., *RFC 4034: Resource Records for the DNS Security Extensions.* http://www.ietf.org/rfc/rfc4034.txt

*Ateniese, G.,, & Mangard, S. (2001) *A New Approach to DNS Security (DNSSEC).* http://www.cs.jhu.edu/~ateniese/papers/dnssec.pdf

BIND 9 Administrator's Reference Manual, http://www.isc.org/sw/bind/arm93/Bv9ARM.pdf

*Boran, S. (2001) *Running the BIND9 DNS Server Securely.* http://www.boran.com/security/sp/bind9_20010430.html.

Brain, M., How Domain Name Servers Work, http://computer.howstuffworks.com/dns.htm

Chappell, L. & Tittel, E. (2004). *Guide to TCP/IP*. Canada: Course Technology.

Dean, T. (2004). *Network + Guide to Networks*. Canada: Course Technology.

Evers, J. (2005). *DNS Servers: An Internet Achilles Heel.* CNET News.com. http://news.com.com/2100-7349_3-5816061.html

Liu, C., & Albitz, P. (2006). *DNS and BIND*. Sebastopol, CA:O'Reilly.

*Householder, A., & King, B., (2002). *Securing an Internet Name Server.* http://www.cert.org/archive/pdf/dns.pdf

*Huston, G. (August 2006). *DNSSEC-The Theory.* http://ispcolumn.isoc.org/2006-08/dnssec.htm

ISC BIND, Internet Systems Consortium. (2004). http://www.isc.org/index.pl?/sw/bind/

*Kato, A. & Murai, J. (2001). *Operation of a Root DNS Server.* http://www.wide.ad.jp/news/event/stanford2002/documents/Root_DNS_System/dns-e84-b_8_2033.pdf

*Langfeldt, N. (2001). *DNS HOWTO.* http://www.ibiblio.org/pub/Linux/docs/HOWTO/other-formats/pdf/DNS-HOWTO.pdf.

*Ollmann, G., (2005). *The Pharming Guide: Understanding & Preventing DNS-related Attacks by Phishers.* http://www.ngssoftware.com/research/papers/ThePharmingGuide.pdf.

*Olzak, T., (2006). *DNS Cache Poisoning: Definition and Prevention.* http://www.infosecwriters.com/text_resources/pdf/DNS_TOlzak.pdf.

*Plante, N. (2004). *Practical Domain Name System Security: A Survey of Common Hazards and Preventatitve Measures.* http://www.infosecwriters.com/text_resources/pdf/dns-security-survey.pdf.

*Walfish, M., Balakrishnan, H., & Shenker, S. (2004). http://nms.csail.mit.edu/papers/sfr-nsdi04.pdf .