# Practical Domain Name System Security: A Survey of Common Hazards and Preventative Measures

Nicholas A. Plante | nap@ccs.neu.edu
College of Computer and Information Science
Northeastern University, Boston MA

## I. Introduction

The Domain Name System (DNS) is a hierarchical database distributed around the world whose primary function is to translate human-readable domain names to numerical IP addresses for network lookup and communication. The current system was designed in 1984 by Paul Mockapetris to eliminate scalability problems that had become apparent with the previous name-to-IP mapping scheme, which involved maintenance of a single hosts file distributed to end hosts periodically. A vast improvement to its predecessor, DNS is well suited to its task of maintaining a relatively efficient, distributed set of name-to-IP mappings, but unfortunately leaves something to be desired in terms of security.

DNS is a critical part of everyday Internet usage required by anyone who has ever checked email through their provider, surfed to a website, or used a chat application to discuss the latest happenings with friends or family. Despite its relative invisibility to the common user, DNS is a service that most every networked application depends on dearly. This being the case, it is an obvious target for manipulation by malicious users. It is also particularly susceptible to compromise due to the following reasons:

- DNS uses the connectionless User Datagram Protocol (UDP) to convey information from authoritative servers to clients instead of its connection-oriented counterpart, TCP. This decision was made for a number of legitimate reasons, but in terms of security, it makes requests particularly susceptible to hijacking, and responses easy to spoof, as it is not subjected to the three-way handshake that is required to set up a TCP connection.
- To make lookups quick and save on unnecessary bandwidth, name servers are designed to cache results for future client lookups for as long as the Time To Live (TTL) field of the cached resource record (RR) specifies. This means that the system is a potential target for cache poisoning, which can lead to the propagation of falsified information as we shall see in the following discussion.

A Fully Qualified Domain Name (FQDN) for a networked computer system provides a way to reach that system without the human user having to be aware of its IP address. This, in some ways, is similar to a title that you might have at a job. If someone sends a package to the CEO of XYZ Corporation, it is a way for them to send a package to the attention of the right person without having to know their particular name (which might legitimately change from time to time, by the way). DNS then, provides a sort of lookup service analogous to a duty that your receptionist at work might perform. If someone calls the office and doesn't know your name or phone number directly, they can talk to the receptionist who can put them in touch with you, simply by asking for the person responsible for a particular duty or department.

We can also extend this analogy to reflect some of the problems inherent with DNS. Since the receptionist in this case doesn't require any sort of validation that you are who you say you are, it's possible that someone in your office might introduce themselves to your receptionist one day as the new CEO and tell them that from now on, all calls and packages going to the CEO should be delivered to their office. The receptionist in our example, being naïve and having no reason to doubt the authenticity of the imposter's claim, would then go ahead and start directing incoming calls and packages to them. This is obviously dangerous for a number of reasons and is useful in illustrating that the large majority of name servers on the Internet are altogether too trusting and don't support a full suite of authentication protocols to verify the identity of the receiving party (or the integrity of the data they are communicating).

In this paper we shall examine some of the most basic threats to the domain name system as it exists today, and make suggestions where possible to reflect best practices that should be observed to eliminate, or at worst, lessen the impact of potential threats. We limit our discussion to solutions that reflect relatively simple changes that administrators can make without drastically overhauling their existing infrastructure, although we touch briefly on DNSSEC and next-generation solutions merely to note that they exist.

We will not focus on implementation problems with particular instances of DNS servers/daemons but instead spend our time discussing practical security threats inherent to the architecture itself. Accordingly, we will leave exploitation of buffer overflow vulnerabilities in certain versions of a very popular name server daemon out of scope for this discussion. Such security hazards are extremely well documented elsewhere.

Additionally, we endeavor to be as implementation-agnostic as possible and keep the discussion relatively high-level, although BIND (The Berkeley Internet Name Domain, the most common UNIX-based DNS server package) configuration examples will be cited to illustrate concepts where necessary. Before we discuss in-depth the security hazards that plague the DNS, we will first give a brief overview of how name lookups are performed which will be critical to understanding how and why DNS is vulnerable to certain attacks.

## II. A Brief Architectural Overview of DNS

Namespace in DNS is organized in a hierarchical tree structure, with the root DNS servers at the top of the tree. These root servers contain information about the DNS servers at the next level down the hierarchy, which include the Top Level Domain (TLD) DNS servers for .com, .net, .org, and many other top-level domains. Going down the hierarchy another level we find DNS servers for individual corporations, networks, and organizations. It is these systems that we concern ourselves with for the majority of this discussion, although the points we mention are general observations about the architecture of the domain name system and therefore apply at every level. At these servers we find information about individual hosts as well as potentially lower-level DNS

servers that may perform the same delegated function for sub-domains of these organizations (e.g., a host www may exist in the sub-domain accounting.xyzcorp.com).

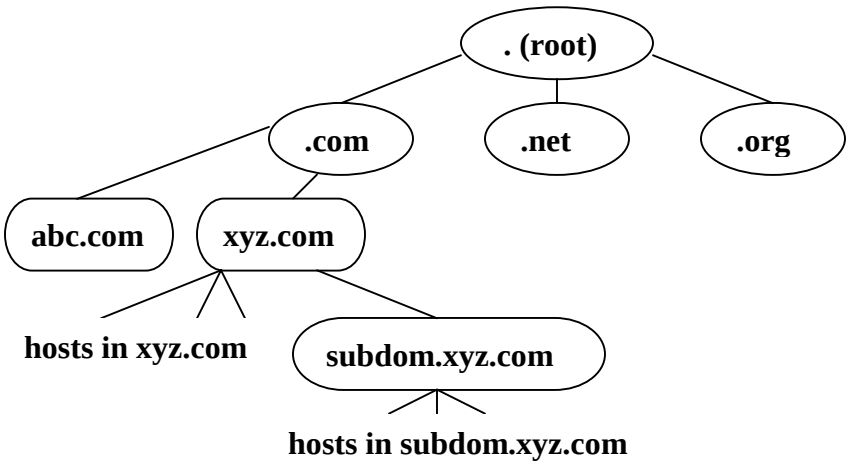
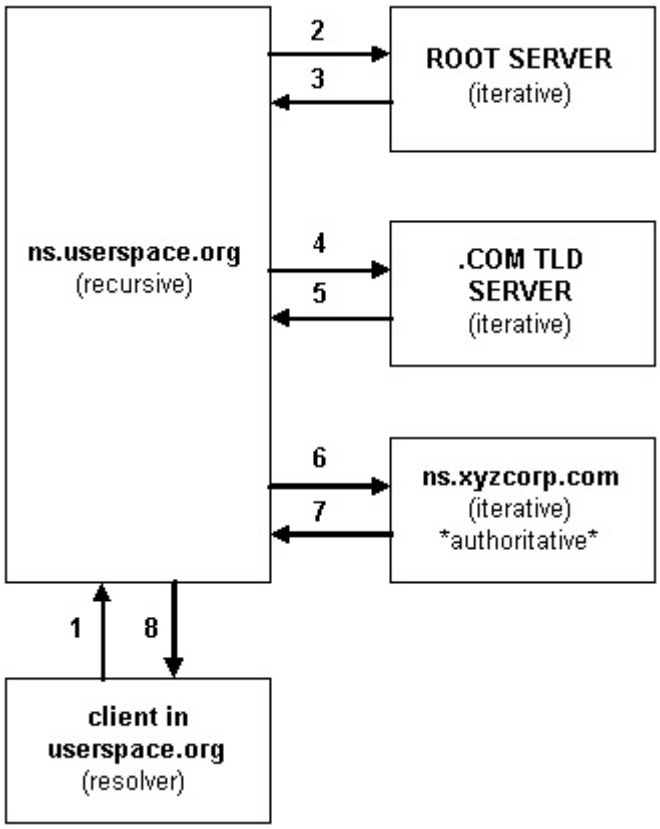



**Figure 1: DNS Namespace Hierarchy**

The process by which client programs search the DNS hierarchy for information about a given host is referred to as resolving. In our example, DNS will be used by a client application (a web browser) in the userspace.org domain to resolve the FQDN www.xyzcorp.com to an IP address (e.g. 192.168.5.250), which it can then use to contact the server and retrieve the requested web pages. We should note that the client system first checks a local cache to see if it already knows the IP address associated with this name. In our case, it does not, so the system formulates a DNS request to its configured (usually local) DNS server, asking for the IP address associated with www.xyzcorp.com.

Since our local name server is not authoritative for the domain in question and does not have this information already cached from a previous lookup, it will resolve the name recursively. This is done by systematically querying various servers in the DNS hierarchy to find the information requested by the client. In this case, our local server would end up talking to the root server (located using root hints), which would point us to the TLD server for the ".com" domain, which would in turn point us to a name server authoritative for the xyzcorp.com domain.

Zone information for a particular domain is always stored on a primary name server for that domain. A zone may also have additional name servers, called secondary name servers, which maintain a mirror of this information for redundancy (and efficiency), which are also authoritative. In this case it doesn't really matter which of these name servers is selected, as long as the server is authoritative for the zone data, and we shall say that the selected name server here is ns.xyzcorp.com. Finally, ns.xyzcorp.com sends ns.userspace.org the information we have requested about www and ns.userspace.org can then pass this on by sending a reply to our client application. At this stage, our client application can access the website by its numerical IP address, transparently to the user.




**1**: Client (resolver) in userspace.org asks its local name server (ns.userspace.org) for the address of www.xyzcorp.com
**2**: Name server does not have any information about this address in local cache, contacts root server with an iterative request
**3**: Root server returns "best effort" address of .com top-level domain server to help continue the resolution process
**4**: Name server contacts TLD name server with an iterative request
**5**: TLD name server returns the address of ns.xyzcorp.com, which its authoritative for the requested zone
**6**: Name server contacts ns.xyzcorp.com with an iterative request for www
**7**: ns.xyzcorp.com returns an authoritative answer mapping www.xyzcorp.com to an IP address
**8**: Name server returns this answer to the requesting resolver

**Figure 2: DNS Lookup Process for Example Case**

It is important to note that, for performance optimizations, ns.userspace.org will cache the entries it receives at each step of the process for future lookups. The next time a user in our domain attempts to access www.xyzcorp.com, ns.userspace.org will be able to supply the IP address for it immediately without repeating the same access pattern, as long as the TTL of www.xyzcorp.com's RR has yet to expire.

In the above example, we are assuming that the local DNS server is configured as a recursive resolver. Queries can be either recursive or iterative by nature. In recursive resolution, the name server has to answer either with a response to the query or an error. Since the server in our example is not authoritative for the data being requested, it has to itself resolve the query again (as shown) before delivering an answer. Once the name server eventually locates the authoritative source for the data it is seeking and obtains either an answer or an error, it passes this response along to the requesting client. Iterative resolution, on the other hand, is when the server simply returns the best answer it has at the moment. Therefore, if it is authoritative it will return a response for the query, and if it is not, it will return a referral to another name server that would help the resolver continue the resolution process, placing the burden on the client. The DNS root servers, for instance, are configured to only respond to iterative requests because of the large volume of queries they receive.

Note that we have intentionally glossed over some important aspects of DNS in this example (such as the reasoning for providing multiple secondary servers, glue fetching, etc) because it is not necessarily critical to the discussion that follows. For a full description of the intricacies of the DNS protocol, the reader should refer to RFC 1034 and 1035 (and their numerous updates).

## III. Zone Transfers and Name-Based Reconnaissance

A zone transfer is an answer to a DNS query to list all DNS information. This includes name servers (NS), host names (A), aliases (CNAME), mail exchangers (MX), start of authority records (SOA), pointer records (PTR), and so on. This also includes the zone serial number, TTL information, and other data associated with name resolution for a domain. Zone transfers are the defined mechanism for mirroring this data between primary (master) and secondary (slave) servers authoritative for a particular domain. Slave servers routinely poll the master to update their zone data, in order to stay current.

Additionally, a zone transfer can also be requested from a regular host to look up information for the entire domain. In fact, the default behavior of most name server packages allows any host to request and receive a full zone transfer. This is usually one of the first actions undertaken by an attacker when determining entry points into a target network, because it allows them to see a complete picture of named systems and clues to network services available to them in a domain. From there, they can scan or probe individual services on hosts and gather information that may be useful in an attack such as the web server name, platform, and version, the type of name server daemon running on your authoritative servers, and so on. Going back to our receptionist analogy, it is similar to a malicious party calling the front desk and asking for a full listing of all the names, titles, offices, and phone numbers of all employees in your company. The caller also hopes that the receptionist might make mention of what someone's favorite sport is, their children's names, or the time they normally leave for lunch.

In a typical configuration, the primary server is configured to only allow the addresses of secondary servers to retrieve the zone file from it. Often, however, even these basic address-based authentication mechanisms are not provided and any user can grab the complete zone file for a particular domain using a tool like dig, which can trigger a zone transfer and retrieve this information using its AXFR function as in the following example:

```
; <<>> DiG 8.3 <<>> @192.168.10.10 localtestnet.com axfr
; (1 server found)
$ORIGIN localtestnet.com.
@                       2H IN SOA       ns1 root (
                                        2004012383      ; serial
                                        2H              ; refresh
                                        1H              ; retry
                                        3D              ; expiry
                                        2H )            ; minimum

                        2H IN NS        ns1
                        2H IN A         192.168.10.10
                        2H IN MX        10 mail
angel                   2H IN A         192.168.10.54
                        2H IN HINFO     "Solaris" "2.7"
dastun                  2H IN A         192.168.10.107
dominus                 2H IN A         192.168.10.200
mail                    2H IN CNAME     angel
orion                   2H IN A         192.168.10.60
rd                      2H IN A         192.168.10.250
                        2H IN TXT       "admin" "desktop"
rosewater               2H IN A         192.168.10.24
schwartzwald            2H IN A         192.168.10.80
                        2H IN HINFO     "Windows" "WWW/FTP Server"
tuxweb                  2H IN A         192.168.10.199
                        2H IN HINFO     "RedHat" "testing pre-deployment"
wopr                    2H IN A         192.168.10.215
```

**Figure 3: Dig Command Output for Sample Domain**

Of course, advertising a complete directory service for every single host in a network is dangerous, but more dangerous still is when a domain is configured with more information than is even required to perform basic resolution. Sometimes hosts are named after people or ongoing projects, which are useful to attackers in determining the purpose of a system. Particularly interesting may be names of systems including the words “test”, “unstable”, or “development”. Sometimes, as seen above, administrators add TXT and HINFO record types that aren’t intended to be seen by outsiders. These records could give away information about a host that would be useful in the hands of a malicious party, such as the operating system type, hardware platform of a particular host or other sensitive information (e.g., “Administrator Desktop”).

As we have seen, failing to limit the retrieval of zone transfers allows a malicious party to perform name-based reconnaissance with great ease. This serves as a first step to gaining information about a target network, and often leads to more in-depth scanning of what the attacker may perceive to be interesting targets. If it’s possible to limit the dissemination of this data to only those who need to be “in the know” (our slave servers) then it is advisable to do so. We shall see how to limit access to this information later on, and how to better structure domains to expose less information to the public.

Before moving on, we also note also that an attacker spoofing the IP address of a master server authoritative for a domain could potentially spoof a zone transfer to one of its slaves if access to updates is not restricted by something stronger than IP-based authentication. Through this means an attacker could load a slave name server with false information about a domain it is supposed to be serving authoritative information for.

Before we're finished we will discuss mechanisms for ensuring the identity of foreign servers during the zone transfer process and verifying the integrity of the data that is sent.

## IV. Simple DNS Denial of Service Techniques

Like all Internet resources, DNS servers are vulnerable to denial of service (DoS) attacks. If a large number of DNS queries are sent to one or more DNS servers on a network from spoofed sources, a denial of service condition may result where the server's network uplink becomes congested or the DNS server response time becomes severely degraded. Although this particular DoS is little different in its effect than a simple ping flood, it is notable because DNS query traffic cannot be traffic-shaped by routers in the same fashion as ICMP traffic without severely inconveniencing legitimate users of the service.

More dangerously, DNS servers can be used for amplification of attack traffic (similar to the way in which a Smurf attack works) if we can create small request packets that generate large responses from the queried server. An obvious example of a small request / large reply packet pair is a request for a zone transfer. Assuming that the name server allows zone transfers from just about anyone, an attacker could spoof a large number of small zone transfer requests using source addresses on a specific victim network. The DNS server will then amplify the traffic sent to it as it returns the significantly larger zone-transfer reply packets to the alleged requestor.

The nature of recursive lookups may also lead to another potential DoS condition. Consider sending a large number of requests for domains guaranteed not to be cached at a particular name server (again, from spoofed sources). For each small query packet sent, the resolving name server will have to perform at least one recursive lookup per packet. This could lead to severe service degradation if a coordinated attack could be launched from numerous sources in a Distributed Denial of Service (DDoS) scenario.

Note that these attacks are general Denial of Service techniques and are not related to the implementation-specific DoS attacks (BIND, etc) that have been discovered over the past few years.

## V. Local Query Interception and Response Spoofing (DNS Hijacking)

If a malicious party is in a situation where they can see DNS queries on the network being sent by clients to a DNS server, then there exists the possibility that the attacker can intercept the queries and beat the name server's response by sending a spoofed response with their own information. This attack results in a race condition but the odds are usually in the attacker's favor if he resides on the same LAN as the victim (which is the case with most attacks of this sort), because the legitimate server may not be on the same LAN or may need to perform a number of recursive queries to return a result, which will slow it down considerably.

Numerous tools have been created that allow for DNS Hijacking of this sort, including DNSSpoof from Dug Song's popular DSniff suite. A DNS Spoofer is a small application that runs on a host and looks for DNS queries on a given interface. It reads data from a fabricated hosts table (formatted like the /etc/hosts file familiar to UNIX administrators), which contains entries for each destination that the spoofer hopes to redirect. Whenever a query is discovered that asks for an address associated with one of the hosts listed in this table, the spoofer is capable of quickly crafting a reply and returning it to the victim.

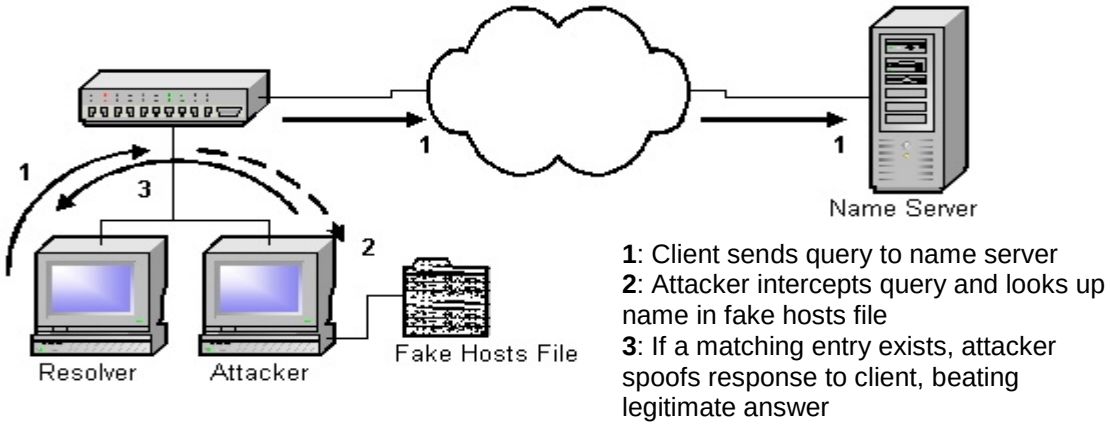



**Figure 4: Local DNS Hijacking Scenario**

Since DNS uses UDP as its primary communication mechanism rather than a connection-oriented protocol like TCP, it is designed to use a field in the message header called the Transaction ID in order to establish orderly communication between the client resolver and name sever. When the DNS server receives a query for a particular bit of information with a Transaction ID of 1674, it will do what it needs to in order to retrieve the requested data and then send a DNS reply to the originator of the query with the original Transaction ID of 1674. Each connection pair it establishes in order to discover this data is assigned its own Transaction ID, and it may have to respond to other requests while awaiting replies.

In the case of DNS Hijacking we assume total knowledge of the DNS query, including the Transaction ID, since we can physically see the packet on the wire. Therefore, it is a simple matter to craft a reply with exactly the same characteristics that a legitimate name server would, but include an IP address mapping from our own malicious hosts table. Note that if the LAN that the hosts reside on is switched, ARP spoofing techniques (described elsewhere) could be used to intercept the traffic being generated by the victim.

As long as the spoofed reply arrives before the legitimate reply, it will be trusted by the resolver, used for the current connection, and cached locally. When the legitimate reply arrives, it will be discarded.

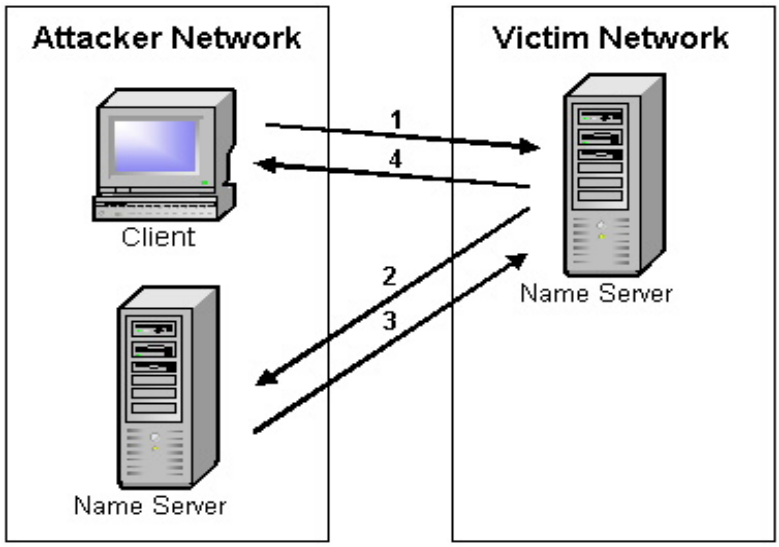
The existence of a local DNS spoofer as described above can be used to set up various scenarios including malicious site redirection, denial of service, and man-in-the-middle, all of which will be discussed shortly.

## VI. DNS Cache Poisoning

A slightly more complicated attack, but one which is much more dangerous because of the fact the attacker doesn't have to be positioned near the vulnerable name server to observe the replies, is the case of DNS Cache Poisoning. In this scenario the attacker would like to convince a legitimate DNS server for a given domain to cache falsified information about a foreign domain that they supply.

As mentioned, DNS uses the Transaction ID in the DNS message header to establish a sense of order amongst queries. So, in order to convince a name server to accept falsified data as legitimate, we must be able to predict the ID that will be used to fetch information about a particular host. This is the key to spoofing an authoritative name server's reply.

The attacker host begins by asking the authoritative name server for xyzcorp.com to resolve a host on his own (attackerspace.org) domain. We assume that the attacker's name server is ns.attackerspace.org and that the attacker can monitor the packets sent-to/received-by that system. The victim name server, which we assume is configured to be a recursive resolver, will look up the authoritative name server for attacker.net in the DNS hierarchy as shown below and then send a query to ns.attackerspace.org asking to resolve the host named www in that network. The attacker's name server will respond in the usual manner.



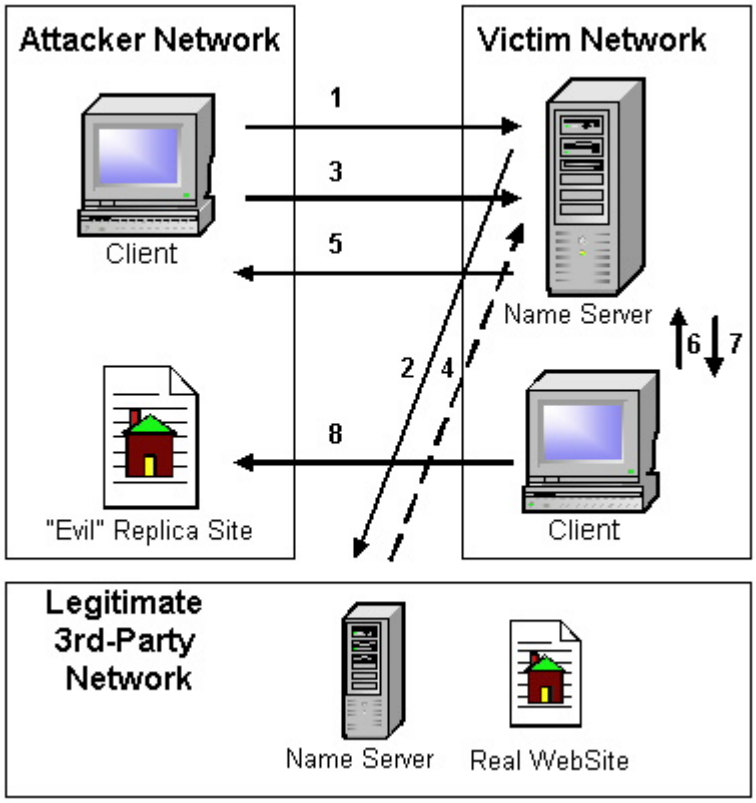

**1**: Attacker sends legitimate query for www.attackerspace.org to victim name server (NS)
**2**: Victim NS doesn't have an answer so it locates the authoritative server at ns.attackerspace.org (this process omitted from diagram) and queries it (Transaction ID = 1674)
**3**: Attacker name server sends the reply. Attacker notes Transaction ID (1674) and will next send other requests to observe how the ID increments (sequentially, pseudo-randomly, etc)
**4:** Victim NS returns this information to the requesting attacker client

**Figure 5: Retrieval of Recursive Query Information including Transaction ID**

The attacker can now examine the Transaction ID that was exchanged with the data for www and may be able to use this for predicting a future Transaction ID. Older versions of BIND (unpatched versions of 4.9) and the Windows NT DNS Server used sequential Transaction IDs. This oversight would allow an attacking spoofer program running on ns.attackerspace.org to observe the previous Transaction ID as discussed and then set the

attacker up to immediately send subsequent requests for domains they wish to spoof while simultaneously sending fake replies with incremental Transaction IDs. The result is once again a race condition between the legitimate host (whom the victim name server will of course be sending legitimate queries to) and the spoofed one. In most cases, it should be trivial to "beat" the legitimate query.

Once the victim name server's cache is poisoned with the falsified entry, any number of the techniques described in the following section may be employed to redirect traffic or cause denial of service conditions for clients using that server as a resolver and attempting to access the poisoned site. In the example scenario shown in Figure 6, the attacker is using the poisoned cache to redirect clients on the victim network to an imposter website under their control.




**1**: Attacker sends query for www.mybank.com to victim NS
**2**: Victim doesn't have an answer so it locates the authoritative server at ns.mybank.com and asks it
**3**: Attacker immediately sends a spoofed reply from ns.mybank.com using an incremented Transaction ID from a previous session (sequential IDs assumed). Response includes a mapping of the requested host to an attacker-controlled IP address
**4:** Victim receives response from ns.mybank.com but discards it
**5**: Victim replies to original query with the attacker-supplied address
**6**: At some later time, a client that uses Victim NS for resolution will query for www.mybank.com
**7**: Response will be the poisoned cache entry (attacker IP), which might host a replica site the attacker uses to steal login information
**8**: Client logs in to replica site, under the belief that it is the legitimate web host…

**Figure 6: DNS Cache Poisoning Attack with Sequential Transaction IDs**

The sequential Transaction ID problem was fixed and current versions of BIND use randomized values. However, the challenge for cache poisoning remains the same: if the Transaction ID is predictable and the server is externally available and recursive, the cache on the server can be poisoned.

Versions of BIND 4 and 8 were also later found to be vulnerable to DNS cache poisoning that worked in a slightly different manner. The implementation flaw was that BIND would send multiple recursive queries for the same domain at the same time if queried for them. This quirk made the daemon vulnerable to cache poisoning by exploiting the

birthday paradox (not really a paradox, but a noteworthy statistical fact that says that in a group of 23 people, the chances of at least two people having the same birthday is over 50%), a technique discussed in Joseph Stewart's paper, "DNS Cache Poisoning – The Next Generation". BIND 9 is not affected by this problem.

Intuitively, any technique that allows Transaction ID numbers to be predicted in a flawed implementation of a name server will result in this kind of vulnerability. At this time, BIND 9 and D.J. Bernstein's TinyDNS (another popular UNIX name daemon) are not known to be vulnerable to any specific techniques of this sort. Of course, this does not keep cache poisoning from being possible. With only 16 bits of space reserved in the DNS message format for the Transaction ID, other techniques may be possible, including simple random guesses (given that there are only 65536 possible values). This obstacle is similar to the difficulty of predicting TCP Sequence Numbers except much more feasible since TCP reserves more space (32 bits) for the Sequence Number field than DNS does.

## VII. Follow-On (Enabling) Attacks

As discussed above in the sections on DNS Hijacking and Cache Poisoning, there are numerous techniques to falsify DNS information to clients. Up until this point we have left the repercussions of these actions to the imagination of the reader, but it is perhaps more frightening and therefore illustrative to examine several situations that could arise if DNS information has been compromised:

- The attacker could take the opportunity to "blackhole" the selected destination, pointing the client to an offline or nonexistent address and therefore performing an effective denial of service against both the requesting party as well as the service provider at the other end.
- The attacker could redirect traffic to a hostile site configured to look like the legitimate server the user is trying to contact. For example, consider a scenario where a user's online bank was redirected to a replica site constructed by an attacker. The hostile site could capture user login credentials and return an error that looks legitimate ("Temporarily out of service. Please try back later."). This appears harmless enough but in reality the user's login and password information are stolen. Later, when the cache entry expires, things return to normal and the user has had no indication that there login credentials were compromised. The replica website can even be set up to handle secure web transactions via SSL and provide users some sense of false security when conducting transactions (although the certificate may not look so legitimate if closely inspected).
- The above scenario is also possible for services other than WWW, such as FTP. Not only could client uploads be lost or compromised, but the attacker could also potentially pose as a repository for system updates or vendor patches. While thinking that they're downloading a new service patch, the victim could in actuality be downloading a trojaned program that will install as a service patch but actually grant the attacker access to their system once installed.

- A modification on the traffic redirection scheme would be where an attacker conducts a man-in-the-middle attack transparently proxying connections through to the real website. An example of this can be conducted with the DSniff suite's DSNSpoof and WebMITM tools. Even SSL sites can be vulnerable to this, as the user will be handed the certificate of the intermediary rather than the certificate of the legitimate host. It might, of course, be noticeable that the certificate is self-signed or signed by an untrusted authority, or that the connection is slightly slower than usual. However, depending on the user awareness level, this may not be enough to dissuade a casual user from proceeding as normal. Revisiting the online bank scenario, an attacker could proxy connections and serve as a man-in-the-middle, capturing all the data exchanged between the client and the bank, including login information, etc, and would not even be forced to construct a replica site or return potentially suspicious error messages.
- MX records are also susceptible to poisoning. Therefore, they can be changed to point a domain's mail to some other (malicious) mail server, which can be configured to receive mail for that domain. All email for the target domain from the poisoned source network could then be redirected to the hostile mail server.
- Many legacy applications rely on trust relationships built upon IP or name-based authentication. At one point in the history of digital communications, this seemed like a good idea to someone, but as of late it has fallen out of favor for obvious reasons. Any application which builds trust relationships based on weak IP-based authentication can be exploited through IP spoofing techniques (as described elsewhere, most notably in the number of articles describing the now-infamous Mitnick attack). Intuitively, applications that perform authentication based on DNS names are vulnerable to the same basic sort of problem. Examples of this include the UNIX rhosts suite of applications (RSH, RCP, etc).

Those scenarios described above are but a sampling of the problems that could be caused by spoofed DNS responses or poisoned name caches. Other examples are left to the reader's imagination.

Next we shall examine a number of proactive preventative measures that DNS administrators can take to prevent these situations.

## VIII. Securely Structuring DNS Deployments (Split-Split Strategies)

Many of the above problems can be completely eliminated by simply limiting certain resources to the hosts that require it. When installing DNS servers for an organization, a number of questions should be asked in order to determine how to limit these resources, such as: "Who and where are the hosts that will need to recursively resolve external DNS servers from our domain?", "Do external hosts need to know about all of our named resources or only a select few?", and so on. The answers to these questions lead us to split the DNS architecture up into different zones and different types of servers. Many organizations use the same DNS servers for all of these tasks, which leave them particularly vulnerable.

Even if zone transfers are properly protected (see the next section), DNS can still expose the structure of an organization's internal network to attackers. Hosts external to your network, for instance, most likely don't need to know your corporate file server's DNS name, although to internal SMB or NFS clients this bit of information may be critical. In order to advertise one set of hosts and services (web servers, email forwarders, etc) to the outside world and keep other hosts that still need names but do not require external visibility separate, we employ a technique often referred to as Split DNS. In order to do this, we maintain two different zones for the domain that are updated separately.

Another type of "split" strategy (sometimes called "Split-Split") that we can make use of is to explore the difference between the concept of an Advertising Name Server and a Resolving Name Server and note how we can separate these two functionalities to lessen the risk of cache poisoning-type attacks. For this purpose, we shall define the concept of an Advertising Name Server as an externally accessible name server that is authoritative for our domain and a Resolving Name Server as a server that clients (presumably only from our domain) will use to resolve names and cache information for domains that are not managed locally (ones we are not authoritative for).

Obviously, an advertising name server needs to be externally accessible in order to be any good; other servers need to know how to find out information about our domain through it using their resolvers, just like *our* resolvers will depend on the existence of externally available advertising servers for other domains. However, most organizations don't need a *resolving* name server to be externally accessible because they only care about assisting clients in their own domains that need to look up DNS data. Not only does externalizing the resolving server make it susceptible to attack but it also further burdens the name server with legitimate queries that they would rather not be servicing. It's wise, in most cases, to take every precaution to ensure that internal systems get the most reliable, correct, and best possible service rather than spending valuable CPU cycles tied up handling requests from other (external) parties whom we owe no allegiance.

With this split architecture in mind, an advertising name server (that only resolves requests for zones that it is authoritative for) can be deployed external to the organization, and a resolving server can be set up internally that only handles requests from local clients and does lookups on external domains. This architecture makes the external server much less susceptible to attack because it never has to look up information about another zone and therefore doesn't have to have any cache for that information or level of trust for other servers. It also protects the internal resolving server by placing it behind a firewall and removing it from the view of intruders who are non-local. However, it is of course still vulnerable to attacks from internal sources. Resolving servers are by nature more susceptible to attack than the external advertising server, because they need to accept (and presumably, cache) data from other DNS servers and therefore require a higher level of trust with the parties from whom it will request information.

To prevent the advertising name server from resolving requests, recursion must be disabled, which puts the name server into passive mode, instructing it to never forward or

send queries on behalf of other resolvers. A non-recursive server is obviously much more difficult to manipulate since it doesn't send queries and therefore doesn't cache any. In BIND 9, recursion can be disabled by adding a "recursion no" substatement inside an options block. This can be done for a specific zone or for global options, depending on the purpose of the name server. We will demonstrate example configurations for both types of servers below:

```
acl slaves { 192.168.1.10, 192.168.2.5; };
options {
        directory "/var/named";
        recursion no;
        allow-query { any; };
};
zone "xyzcorp.com" {
        type master;
        file "db.xyzcorp.com";
        allow-transfer { slaves; };
};
```

**Figure 7: Advertising Server (External)**

```
acl internal { 192.168.2.0/24; };
options {
        directory "/var/named";
        allow-query { internal; };
};
zone "." {
        type hint;
        file "db.cache";
};
zone "xyzcorp.com" {
        type slave;
        masters { 192.168.1.5; };
        file "db.xyzcorp.com";
        allow-transfer { internal; };
};
```

**Figure 8: Resolving Server (Internal)**

If recursion cannot be disabled entirely in a particular situation, it is useful to at least limit the range of addresses able to utilize this functionality or the names of zones they can ask about. This can be specified in BIND 9 by using the allow-recursion substatement in an options block as such:

*Allow-recursion { 192.168.1.0/24; localhost; };*

Limiting recursion to internal/local sources significantly reduces the risk of cache poisoning attacks if it is not feasible to disable recursion entirely.

## IX. Access Lists and Transaction Signatures for Zone Transfers

Another one of the recurring problems in our discussion of DNS security is that there is no authentication method to ensure that the reply a resolver (be it a client or a resolving name server) receives from another name server is guaranteed to actually be from that source and/or have correct data. It seems that just about anyone could craft a reply to a DNS query that would be treated as valid by the querying party as long as they could see the request itself or at least predict the Transaction ID associated with it. The fact that DNS caches information in order to improve response times for oft-queried domains compounds the problem but is necessary for efficiency reasons.

To restrict foreign access to zone transfers, most name servers can be configured to allow transfers only from specified servers. BIND, for instance, supports an access control list feature which can be used for this purpose. The basic format is:

*Allow-transfer { 192.168.1.10; };*

Where 192.168.1.10 is a secondary name server for our domain. This can be specified inside of the global options or specifically for each zone handled by the server if there are more than one.

Note that this weak form of authentication is based solely on IP source address, and therefore it does not address IP spoofing problems. Thus, it could be argued that this mechanism is relatively insecure. For example, consider a slaved secondary name server for a domain that retrieves zone data from a master primary name server by means of this IP-based authentication scheme. If an attacker is able to spoof the IP address of the master (this involves spoofing a TCP connection), they could potentially supply falsified zone data to the slave, which would then serve it to clients. This is therefore a less than ideal mechanism to control access.

Although this is certainly better than nothing, a superior solution in this case is to use Transaction Signatures (TSIG). TSIG, as defined in RFC 2845, is a more secure method of access control that uses a shared secret key and a one-way hash function (only HMAC-MD5 is supported at present, which requires as input both the message as well as the secret key) to establish trust between two name servers and authenticate the DNS messages that are exchanged. TSIG is most often used to authenticate the zone transfers that occur between primary (master) and secondary (slave) name servers authoritative for a particular domain. It can also be used between a DHCP server and a DNS server to secure the dynamic update process if this is configured (although we avoid the topic of dynamic updates in this paper, we acknowledge that they are in common use).

A DNS server configured to use TSIG adds an additional TSIG resource record to the DNS data section of a message. This record is not actually part of the zone itself, but is used as a signature to verify the authenticity of the message when it is received. As such, the TSIG RR is not cached by the recipient. Successful verification of the signature proves to the receiving party that the sender had the shared key it was expecting and therefore no changes were made to the record while in transit, as any changes would have altered the result and failed the verification process. Therefore, we can trust the data we have received.

To demonstrate how this works, we consider the scenario where a slave server requests a zone transfer from its master. The slave will sign the request it sends with the shared key it is configured to use when corresponding with that system. The master will verify the key and if the request is deemed valid it will sign its answer with the same shared key and send it back. If the signature shows the message to be invalid or tampered with, the request will be rejected. When the slave receives its answer, a similar process takes place.

The benefit of this is that it restricts zone transfers to only those hosts configured with valid DNS TSIG keys, meaning that each DNS query and response can be verified as coming from a legitimate source with valid TSIG keys.

To conclude our illustration of the use of TSIG, we include two excerpts from BIND 9 configurations. The left listing reflects the configuration of a master server that needs to support TSIG transactions with a slave, which is listed on the right. The key they share for this purpose is ns1-ns2.tsigkey.

```
key ns1-ns2.tsigkey. {
        algorithm hmac-md5;
        secret “k2Pb7gEcbXg6ZosOqAbV8A==”;
};
server 192.168.10.10 {
        keys { ns1-ns2.tsigkey.; };
};
zone “xyzcorp.com” {
        type master;
        file “db.xyzcorp.com”;
        allow-transfer { 192.168.10.10; };
};
```

**Figure 9: TSIG for Master Name Server**

```
key ns1-ns2.tsigkey. {
        algorithm hmac-md5;
        secret “k2Pb7gEcbXg6ZosOqAbV8A==”;
};
server 192.168.10.5 {
        keys { ns1-ns2.tsigkey.; };
};
zone “xyzcorp.com” {
        type slave;
        file “bk.xyzcorp.com”;
        allow-transfer { none; };
};
```

**Figure 10: TSIG for Slave Name Server**

In BIND, TSIG keys can be generated by the dnskeygen utility and distributed to the remote host out of band or using TKEY (RFC 2930), a mechanism that allows hosts to generate a shared secret automatically using the Diffie-Hellman key exchange. It is recommended to configure unique keys between each pair of name servers that need to communicate in this manner, otherwise the compromise of a single key could have infrastructure-wide effects.

We should also note that time synchronization is absolutely critical if TSIG is to be used. One of the fields in the TSIG record is time, which was added to prevent replay attacks after a certain expiration time. Therefore a Network Time Protocol (NTP) source must be available at all times to both servers in order to avoid rejection of messages due to time expiration.

## X. General Suggestions for Improving DNS Site Security

In addition to the strategies above, there are a number of other very general suggestions that should be taken under consideration in order to improve the security of your organization’s DNS. We list some of them here:

- Configure the DNS server host to only respond on TCP/UDP 53 and no other ports in order to eliminate the possibility that an unrelated (and potentially unnecessary) service is exploited and used as a means to compromise DNS.
- Additionally, TCP port 53 could be limited at the firewall to prohibit zone transfers outside the local network if this isn't deemed necessary. This provides extra implicit protection, but actually violates the RFC because it specifies that DNS messages larger than 512 bytes must be sent via TCP. Although it is unlikely that anything but a zone transfer will generate a packet this large, it is a possibility.
- Don't run the server daemon as root in case there are undiscovered exploits existing in the listening process that could be used for an attacker to gain access to the host. Running the daemon in a chroot jail would further limit the amount of damage that could be caused.
- To limit the effect of localized DoS attacks, make sure you have at least one secondary server configured for serving authoritative data for your domain. Often, it is advisable to have two or more secondary name servers located on physically disparate networks for any volume of lookups.
- Remove unnecessary HINFO and TXT records from your zones. There is simply no convincing reason to include this information in DNS records, as they provide no additional functionality to DNS and there is no need to advertise more information about your machines than necessary.
- Don't allow dynamic updates to a zone. Although not discussed specifically in our paper, some DNS servers allow dynamic updates from DHCP servers as hosts come online. This has rather obvious implications for security, and if possible, should be avoided. If your organization requires the functionality, using an access-list to limit the hosts allowed to update it is highly advisable.

## XI. The Future? DNS Security Extensions (DNSSEC)

DNSSEC is an extension to the original DNS protocol proposed in RFC 2535 (later updated in RFC 3007, 3008, and 3090) and aimed primarily at securing the client-server lookup portion of the resolution process. It allows the client to both authenticate the identity of the server as well as verify the integrity of the data it is receiving from that server using public key cryptography.

The basic idea is that each zone is given a public/private key pair and every record in a zone file is signed with its private key, creating a SIG resource record. The zone's public key is then stored in a KEY record. The authoritative name server will sign the data with its private key, thus allowing anyone to decrypt it using the public key retrieved from the server's KEY RR.

If fully implemented throughout the Internet, advocates claim that DNSSEC would be a truly elegant solution to most (if not all) of the problems plaguing the current protocol. Unfortunately, there are numerous obstacles standing in the way of its deployment and reasons why it has yet to become widespread.

DNSSEC raises many questions that at this time may not have clear-cut answers: How will TLD administrators sign public keys from a given domain in order to build a chain of trust? What happens if the root key (used to sign domains) or keys at the TLD level are compromised? What about key revocation and rollover? It is important to observe that the new system, like the current one, provides no defense against DoS attacks. Performance issues may also be prohibitive. Signed zones are significantly larger than their unsigned counterparts, given that the size of a signature dwarfs the size of the record it is signing. Not to mention that the signing and verifying operations take valuable CPU time and memory.

Although (mostly) implemented in BIND 9 and successfully tested in workshop and test bed environments, DNSSEC has yet to be deployed in environments large enough to simulate its use on the global Internet. Most parties seem to agree that further modification and refinement is required before DNSSEC will be able to scale appropriately. Ultimately, the real test is global adoption involving not just the deployment of DNSSEC servers but also the disablement of the current "weak" crop of DNS servers, which makes the barriers standing in its way comparable to those of IPv6 or an SMTP replacement for secure (spam-proof) mail exchangers. In other words, don't hold your breath.

The above discussion does not do justice to the rich features and deployment concerns regarding the DNSSEC extensions, but we limit our discussion of the topic here in order to observe that it has yet to be successfully implemented on a wide-scale basis and therefore is not currently a practical solution that can be easily adopted by organizations to secure their existing DNS resources.

## XII. Conclusion

The domain name system is an extremely critical service, one that every casual Internet user relies on for the most basic of network applications – whether they know it or not. This system of name resolution, however, having been established long before computer networks were considered for commerce-centric applications (and therefore long before the current crop of security hazards existed), is also plagued by a number of vulnerabilities that to this day are highly exploitable if certain precautions are not taken.

The need for authentication during zone transfers and between resolving name servers and clients will eventually necessitate the widespread deployment of DNSSEC or a similar system, but until that day comes we must combat the existing problems in the most practical and efficient manner possible. For now there are a number of far simpler steps that administrators can implement in order to make DNS as secure as possible for our users. Many of the weaknesses described in this paper represent relatively "simple" problems that could be easily buttoned up if well-informed, security-conscious administrators were educated about them and trained to structure their network architecture to mitigate the risk of attack. While the previous sections describe current

best practices that should be implemented where possible, further research will no doubt be required for more advanced architectures.

Likewise, even the most casual Internet users generally take the security of DNS for granted and never think twice about it. These users need to learn that the bridge they are walking on may not be as fireproof as they at first thought! For example, not only should users note the difference between an HTTP and HTTPS connection when making an online purchase, but they should also be extra-careful to verify the integrity of a certificate they are handed, and to ensure that it comes from the expected source (instead of immediately clicking "yes"). This is merely one example, of course, and is far from a universal solution, but it is certainly a step in the right direction.

Like it or not, the big picture is that DNS is at the foundation of many "secure" protocols, and it is important to note once again that the security of the entire system is only as strong as the weakest link. In many cases, the weakest link is DNS. Until a truly secure next-generation replacement is finally settled upon and deployed throughout the world (quite a task indeed!), the best weapon we have to combat these problems is education.

## XIII. References


1. E Nemeth. Securing the DNS. ;Login volume 25 number 7, November 2000.

2. C Liu et al. DNS and BIND, 4th Edition. O'Reilly and Associates, April 2001.

3. E Skoudis. Counter Hack. Prentice Hall PTR, July 2001.

4. J Stewart. DNS Cache Poisoning – The Next Generation. January 2003. http://www.securityfocus.com/guest/17905

5. Dug Song. Dsniff. http://naughty.monkey.org/~dugsong/dsniff/

6. S Boran. Running the BIND9 DNS Server Securely. March 2001. http://www.boran.com/security/sp/bind9_20010430.html

7. P Mockapetris. RFC 1034: Domain Names – Concepts and Facilities. November 1987. http://www.ietf.org/rfc/rfc1034.txt

8. P Mockapetris. RFC 1035: Domain Names – Implementation and Specification. November 1987. http://www.ietf.org/rfc/rfc1035.txt

9. P Vixie et al. RFC 2845: Secret Key Transaction Authentication for DNS (TSIG). May 2000. http://www.ietf.org/rfc/rfc2845.txt

10. D Eastlake. RFC 2930: Secret Key Establishment for DNS (TKEY RR). September 2000. http://www.ietf.org/rfc/rfc2930.txt

11. D Eastlake. RFC 2535: Domain Name System Security Extensions. March 1999. http://www.ietf.org/rfc/rfc2535.txt

12. B Wellington. RFC 3007: Domain Name System (DNS) Dynamic Update. November 2000. http://www.ietf.org/rfc/rfc3007.txt

13. B Wellington. RFC 3008: Domain Name System Security (DNSSEC) Signing Authority. November 2000. http://www.ietf.org/rfc/rfc3008.txt

14. E Lewis. RFC 3090: DNS Security Extension Clarification on Zone Status. March 2001. http://www.ietf.org/rfc/rfc3090.txt