DNS Cache Poisoning – The Next Generation

By Joe Stewart, GCIH <jstewart@lurhq.com>

Introduction

The old problem of DNS cache poisoning has again reared its ugly head. While some would argue that the domain name system protocol is inherently vulnerable to this style of attack due to the weakness of 16-bit transaction IDs, we cannot ignore the immediate threat while waiting for something better to come along. There are new attacks, which make DNS cache poisoning trivial to execute against a large number of nameservers running today. The purpose of this article is to shed light on these new attacks and recommend ways to defend against them.

A Brief History of Cache Poisoning

In 1993, Christoph Schuba released a paper entitled "Addressing Weaknesses in the Domain Name System Protocol". In it, he outlined several vulnerabilities, including the technique of DNS cache poisoning. In the earliest incarnation, it was possible to provide extra information in a DNS reply packet that would be cached by the daemon. This allowed an attacker to inject false information into the DNS cache for a network, allowing them to perform man-in-the-middle attacks or other mayhem.

In 1997, CERT released advisory CA-1997-22, describing a vulnerability in BIND, the Berkeley Internet Name Domain software which is used by nearly all of the nameservers on the Internet. This time a very basic principle was finally realized: BIND did not randomize its transaction IDs – they were purely sequential. Aside from layer 3 and 4 protocol checking (source and destination IP addresses and ports must match), the transaction ID is the sole form of authentication for a DNS reply. Because an attacker could easily predict the next transaction ID after making their own request, a cache poisoning attack could be carried out using a spoofed query followed by a spoofed answer. To solve this, all new versions of BIND were updated to use randomized transaction IDs.

In 2002, Vagner Sacramento released an advisory showing another problem with BIND's implementation of the DNS protocol. He found that BIND would send multiple simultaneous recursive queries for the same IP address. Because of this a mathematical phenomenon comes into play known as the "Birthday Paradox". This causes the probability of a successful attack to rise to near 100% with only a few hundred packets instead of the tens of thousands previously believed to be needed.

While researching Sacramento's findings, the CERT team also realized there might be another attack possible, based on the work of Michal Zalewski in the area of TCP sequence numbers and phase space analysis of the psuedo-random number generators used by different operating systems to generate them. Zalewski found that using a certain type of analysis it was often trivial to guess the next sequence number in certain implementations. The CERT team felt that this might also apply to the random number

generators in BIND. This article will attempt to show that their assumption was correct.

A DNS Protocol Refresher

A simplified flowchart of the DNS protocol is shown below in illustration 1. Basically a user who wants to find the IP address of a webserver would first query her local DNS server. This server is designed to make queries on her behalf to as many nameservers as needed in order to find the answer. This is known as recursion.

The domain name system is laid out as a tree for efficiency. At the top are the root level nameservers. They contain information about what nameservers hold the specific information about the hosts in each top-level domain. This information is known as the *authority* record for a domain. In it are pointers to the servers that are considered authoritative for a domain.

The flow of the entire transaction is as follows: The client resolver queries the local recursive (caching) nameserver and asks for the IP address of www.google.com. The recursive nameserver asks the root server for information about the name. It might get an answer back such as the following:

```
Name:    google.com
Served by:
- ns2.google.com
          216.239.34.10
          google.com
- ns1.google.com
          216.239.32.10
          google.com
```

This is a list of the authoritative nameservers for google.com that the root nameserver knows about. At this point, the recursive nameserver will then ask the first authoritative nameserver for the information requested by the user. If it gets an answer, it will pass that along to the client. If no answer is received, it will try each nameserver in the list until it receives an answer.

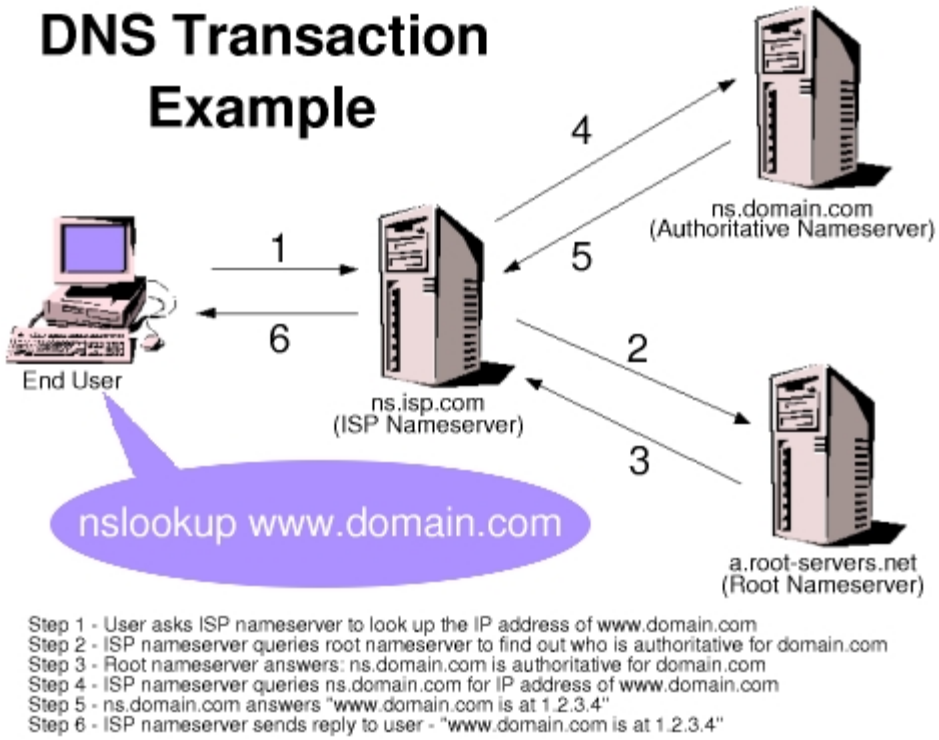

Illustration 1: The DNS protocol in action

In order to verify authenticity of the reply, the DNS system uses transaction IDs (sometimes called query IDs). A 16-bit number is generated by the nameserver or resolver client who is issuing a query. Any reply from the nameserver must contain this transaction ID. As long as the TCP or UDP port number, IP address and transaction ID from the remote host are correct, the reply is considered to be legitimate.

Attack #1 – The Birthday Attack

To perform this attack, one needs to send a sufficient number of queries to a vulnerable nameserver, while sending an equal number of phony replies at the same time. Because the flaw in the BIND software generates multiple queries for the same domain name at the same time, one encounters statistically improved odds of hitting the exact transaction ID. This is the classic “Birthday Attack”, which is derived from the “Birthday Paradox”, described below:

> *A birthday attack is a name used to refer to a class of brute-force attacks. It gets its name from the surprising result that the probability that two or more people in a group of 23 share the same birthday is greater than 1/2; such a result is called a birthday paradox.*
>
> *If some function, when supplied with a random input, returns one of k equally-likely values, then by repeatedly evaluating the function for different inputs, we expect to obtain the same output after about 1.2k1/2. For the*

above birthday paradox, replace k with 365. (unknown author, http://www.x5.net/faqs/crypto/q95.html)

We can apply the same methodology to psuedo-random number sequences, such as the one that generates transaction IDs in BIND.

With conventional spoofing, we would send *n* spoofed replies for one query. Our probability of success is *n* / 65535. With the BIND birthday attack, we send *n* number of spoofed replies for *n* queries. For this, we can predict the probability of success using the formula below where *t* is the total number of possible values in the master set, and *n* is the number of values in the spoofing subset.

$$\textit{Probability of Collision} = 1 - \left(1 - \frac{1}{t}\right)^{\frac{n \times (n-1)}{2}}$$

The power of this method of spoofing versus conventional DNS spoofing is shown in illustration 2. The birthday attack nears 100% success around 700 packets. At this point the conventional spoofing attack would only have a success probability of 700 divided by 65535 (1.07%) The steepness of the curve is such that one needs only 300 packets to achieve a 50% success ratio. This is well within the realm of a trivial attack to anyone with a broadband Internet connection.

This shows that even a perfect random-number generator is vulnerable to attack when it generates multiple numbers for the same transaction. This is something that should be taken into account by any software designer who is working with random numbers. It has long been used in brute-force attacks on one-way hash systems, as described by Bruce Schneier in his book *Applied Cryptography*.

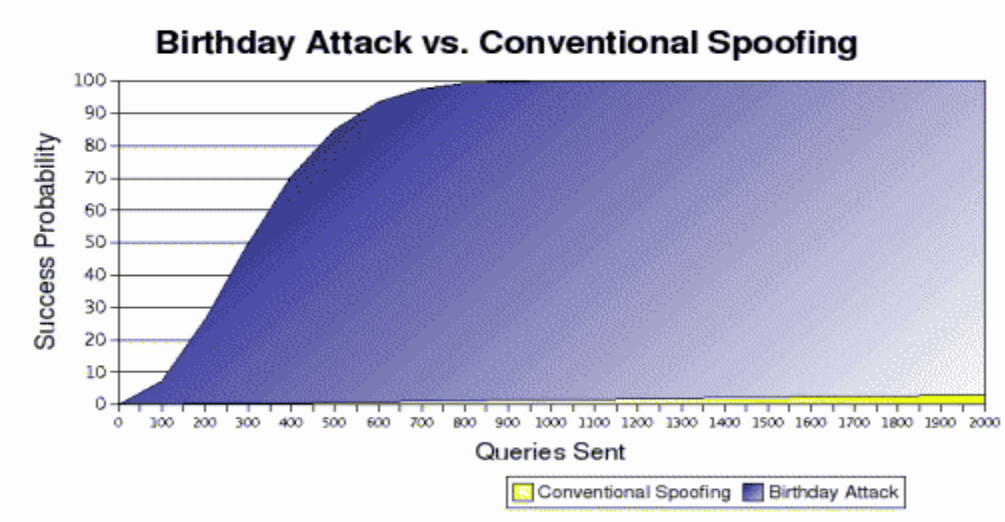

Illustration 2: Birthday Attack vs. Conventional Spoofing

The BIND birthday attack would follow the sequence shown in illustration 3. The attacker merely needs to send a few hundred queries to the ISP nameserver asking for the IP address of the domain name to be hijacked. At the same time, he will send the same number of replies formulated to look as if they were sent from the authoritative nameserver. In each packet he will assign a random transaction ID. In order to be successful, one of his spoofed packets transaction IDs, source and destination IP addresses and ports must match a legitimate recursive query packet from the victim nameserver.

Finding the correct IP addresses is easy; we know our target, and we know the addresses of the legitimate nameservers for the domain to be hijacked. Finding the port is slightly harder. We know that the destination port of the recursive query is UDP port 53, but the source port is a moving target. Fortunately for our attacker, BIND will more often than not reuse the same source port for queries on behalf of the same client. So, if the attacker is working from an authoritative nameserver, he can first issue a request for a DNS lookup of a hostname on his server. When the recursive query packet arrives, he can look at the source port. Chances are this will be the same source port used when the victim sends the queries for the domain to be hijacked. Look at the tcpdump output of four subsequent queries for different domain names:

10:54:12.423228 192.168.1.2.33748 > 66.218.71.63.53: 21345 [1au] A? www.yahoo.com. (42) (DF)

10:54:21.313293 192.168.1.2.33748 > 216.239.38.10.53: 53735 [1au] A? www.google.com. (43) (DF)

10:54:27.182852 192.168.1.2.33748 > 149.174.213.7.53: 19315 [1au] A? www.netscape.com. (45) (DF)

10:54:43.252461 192.168.1.2.33748 > 66.35.250.11.53: 43129 [1au] A? www.linux.com. (42) (DF)

All four queries used source port 33748 while querying four different nameservers. If BIND used randomized source ports, it could improve its chances of warding off spoofing attacks.

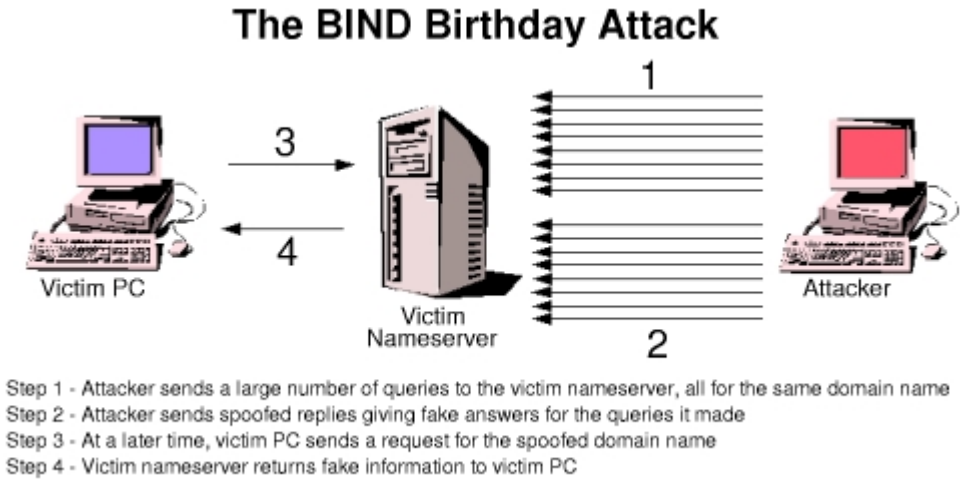

Illustration 3: The BIND Birthday Attack in action

At this point, all the attacker needs to do is win the race between the first successful collision of his spoofed transactions and the legitimate answer from the true authoritative nameserver. This race is already slanted in favor of the attacker; however, he could utilize other methods to gain an even bigger edge, such as flooding the authoritative nameserver with bogus packets in order to slow down its response time.

After a collision between a legitimate recursive query and a falsified reply packet, the targeted nameserver at the ISP will cache the spoofed record for the time indicated in the TTL section of the reply. At this point the attacker is finished, but the effect lives on for the time the ISP holds the phony record in its nameserver cache. The victim user at the ISP is exposed to the attack any time it makes a query for the domain name in question.

Attack #2: Phase Space Analysis Spoofing

The CERT vulnerability note at http://www.kb.cert.org/vuls/id/457875 made the following observation:

> *Additionally, Michal Zalewski's paper "Strange Attractors and TCP/IP Sequence Number Analysis" [ZALEWSKI] describes a method for analyzing the predictability of transaction IDs which we believe could be extended to analyze Transaction ID / UDP port pairs as well.*

From my observations, it appears they are correct. I downloaded Michal Zalewski's tools for phase space analysis and ran them on a set of 100,000 16-bit numbers generated by BIND 8, BIND 9, and djbdns.

Zalewski's description of phase space analysis of PRNG functions follows:

> *Phase space is an n-dimensional space that fully describes the state of an n-variable system. An attractor is a shape that is specific to the given PRNG function, and reveals the complex nature of dependencies between subsequent results generated by the implementation. (Zalewski, http://razor.bindview.com/publish/papers/tcpseq.htm)*

Essentially this means we can tell a lot about how random a PRNG really is by looking at its results in 3-D space. We can also make predictions about the next number in the sequence based on the math involved here.

The tools provided by Zalewski that are used in this article are *vseq*, a linux svgalib program that visualizes the dataset in 3-D space, and *calprob*, which tries to predict the probability of guessing the correct next sequence number of a PRNG when given 3 preceding sequence numbers. These tools are contained in this archive: http://razor.bindview.com/publish/papers/tcpseq/vseq.tgz.

BIND 8 PRNG Analysis

Below is the output of the vseq program, showing the dispersion of pseudo-random transaction IDs in 3-D space:

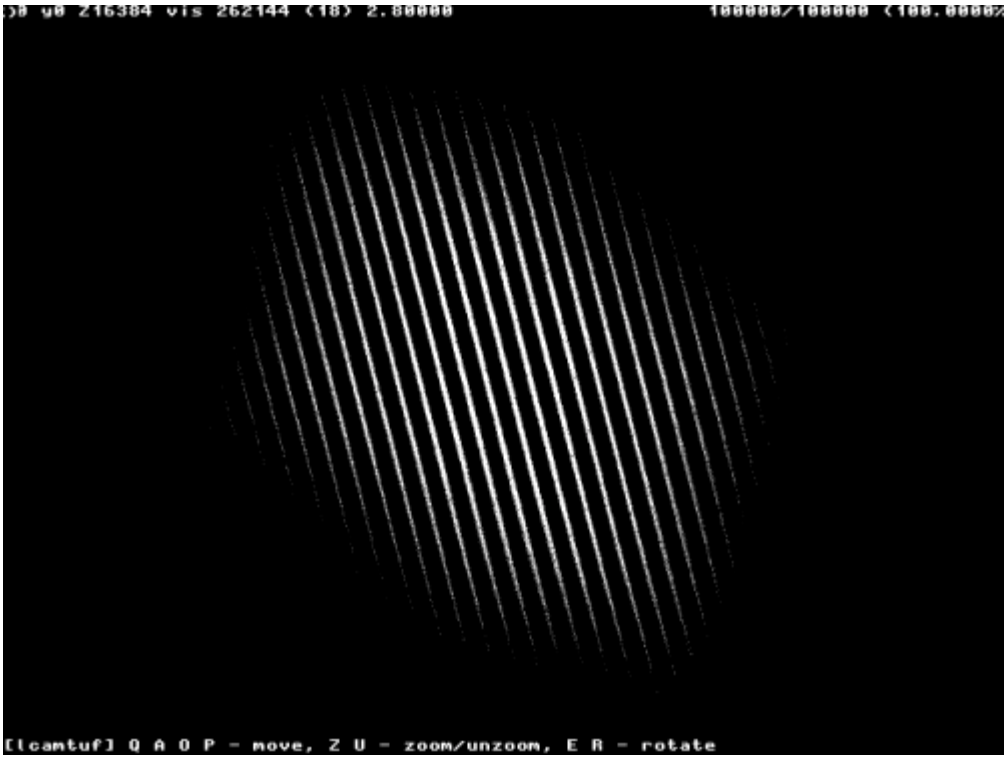

Illustration 4: Phase Space Analysis of BIND 8.3.4 Transaction IDs

An ideal random number generator would appear as an evenly dispersed cloud. When there are geometric patterns, or “attractors” in the output, this indicates imperfections in the randomness of the data. BIND 8.4.3 clearly shows the randomness following distinct lines with space in between; indicating that there are some numbers that are more likely to be chosen in a sequence than others. If we limit our spoofing set to those numbers, we greatly increase our odds of a collision between a query and a spoofed reply.

Zalewski includes a tool to analyze the probability of successfully guessing the next number from a sequence using attractor analysis. That program accepts 4 arguments; a datafile with 100000 randomly generated numbers, a radius R1, a spoofing set size, and number of tries. Here is the output of that program run against the BIND transaction ID sequence:

```
./calprob ../bind.nsid.gz 10 1 1
1: Trying 13085 32386 21499 (-> 36312) - r=10
  + guess3d gave 51 answers, rsort suggests R2 of 0...
  -> SUCCESSFUL (difference 0).

Data file:         ../bind.nsid.gz
Failed attempts:   0/1 (0%)
Average R2:        0
Average N:         51
Average error:     0
Average correct N: 3
Probability:       100.0000%
```

With this algorithm Zalewski's tool predicts a probability of success of 100% with a spoofing set size of 1. In other words, you can mathematically predict the next transaction number with only 3 previous transaction IDs 100% of the time. This is completely

independent of the flaw in BIND that allows the Birthday Attack. This pretty much ensures that even if the Birthday Attack is patched against, BIND 8.x will still be vulnerable to cache poisoning as long as it continues to use the same pseudo-random number generator.

BIND 9 PRNG Analysis

BIND 9.x uses a completely new random number generation sequence which uses the /dev/random device available in most modern Unix-type operating systems. When properly seeded with adequate entropy, this device gives us a stream of very random data. Of course, operating systems may implement the random device differently, so that should be taken into account. In this case, the output relied on the random device implementation of the Linux 2.4.19 kernel.

Looking at the vseq program output for BIND 9, we see an interesting snowflake-like pattern. The calprob output gives better results than BIND 8, however:

```
./calprob ../bind9.nsid.gz 200 5000 10
...
Data file:          ../bind9.nsid.gz
Failed attempts:    0/10 (0%)
Average R2:         75
Average N:          522
Average error:      12
Average correct N:  24
Probability:        20.0000%
```

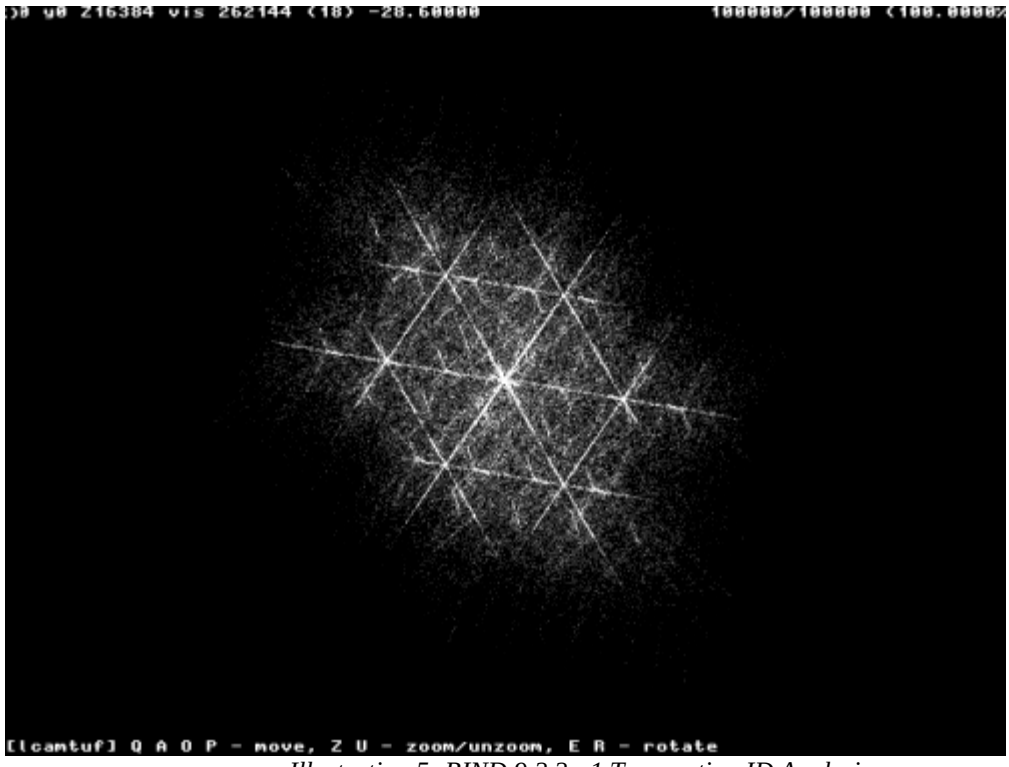

Illustration 5: BIND 9.2.2rc1 Transaction ID Analysis

According to this, BIND 9's random number sequence is predictable 20% of the time

with a spoofing set size of 5000. This is far better than BIND 8, but still leaves opportunity for a dedicated attacker. It would take a fair amount of bandwidth in order to be able to send 5,000 spoofed packets to a target nameserver and beat the authoritative nameserver. However, if the authoritative nameserver was vulnerable to any of the BIND denial-of-service attacks, one could merely kill the authoritative server with a malicious packet and have a one-man race to win. If the denial-of-service attack was combined with the phase-space analysis attack on BIND 8, it would be impossible to detect a spoof with any intrusion detection system, as it would there would be only one query and one answer received.

djbdns PRNG analysis

Looking at djbdns, we see fewer attractors, just a geometric cloud of points (see illustration 6). The calprob output shows the randomness is still not perfect (in fact, it is slightly worse than BIND 9; 30% probability of a successful guess with a spoofing set size of 5000):

```
./calprob ../djbdns.nsid.gz 200 5000 10
...
Data file:          ../djbdns.nsid.gz
Failed attempts:    0/10 (0%)
Average R2:         33
Average N:          623
Average error:      22
Average correct N: 31
Probability:        30.0000%
```

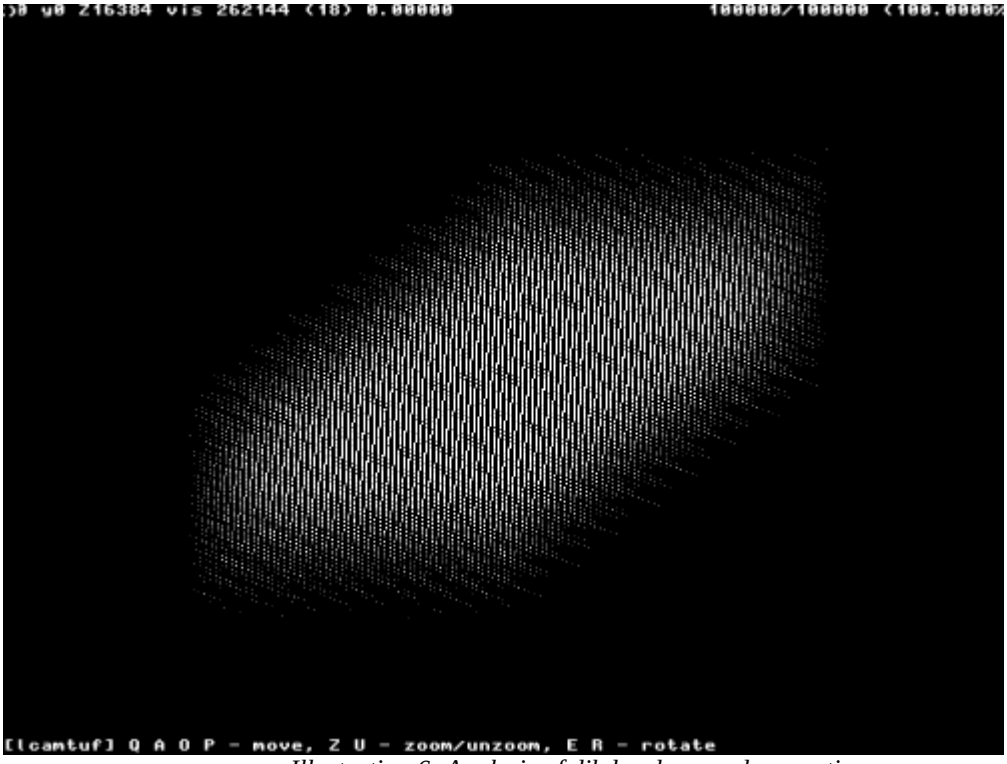

Illustration 6: Analysis of djbdns dns_random routine

This is however offset by the fact that djbdns also generates random numbers for the

source port of each query, as evidenced by the tcpdump traffic below:

22:42:41.790753 192.168.1.2.16075 > 64.58.77.85.53: 36904 A? www.yahoo.com. (31) (DF)

22:42:53.876719 192.168.1.2.53928 > 216.239.38.10.53: 1776 A? www.google.com. (32) (DF)

22:43:07.996666 192.168.1.2.59368 > 207.200.73.80.53: 16261 A? www.netscape.com. (34) (DF)

22:43:18.290976 192.168.1.2.9183 > 66.35.250.10.53: 5110 A? www.linux.com. (31) (DF)

In this instance, a unique random source port was used for each query. This forces the attacker to guess transaction ID and source port simultaneously. It is immensely difficult to succeed at such an attack, but as D. J. Bernstein warns on his site:

> *Note, however, that there are only about a billion possible ID-port pairs, so a prolonged blind attack will succeed eventually. (Bernstein, http://cr.yp.to/djbdns/dns_random.html)*

Sub-Attacks

Once an attacker has managed to poison a DNS cache, there are a number of ways she can subvert protocols that rely on DNS. Some of the potential methods are listed below.

Redirecting Web Traffic

An attack of this nature might range from a simple annoyance to a financial nightmare for a great number of people. The goal here is to set up a website that looks enough like the original so as to not raise any suspicion. Then the domain is hijacked via cache poisoning for as many ISPs/companies as possible, causing their traffic to hit the phony site instead. Some of the sub-attacks here are:

- Redirect a popular search engine to a pop-up ad site
- Redirect a bank website to gain access to account passwords
- Redirect news site to inject false stories and manipulate stocks

Man-in-the-Middle

In this scenario, an attacker tries to intercept secure communication between two parties. For example, Xavier wants to make an online purchase at Yuri's website. The attacker poisons the cache at Xavier's ISP, pointing traffic to Yuri's site to Zamfir's system. Zamfir accepts the incoming SSL connection, decrypts it, reads all the traffic, and makes the same request via SSL to Yuri's site. Replies from Yuri are read by Zamfir then sent back to Xavier over the same encrypted session. Zamfir now has Xavier's credit card number and all other details needed to make illicit use of it.

Recommended Defenses Against DNS Cache Poisoning

Users of BIND

When attempting to protect yourself against a DNS spoofing attack, you have to consider the different aspects of the attack and where you fit in. For instance, do you want to prevent against your users being returned bogus data? Or are you trying to prevent your

domain name from being hijacked? In any DNS spoofing attack there are two victims – the hijacked domain owner, who is losing traffic to his site, and the end user who gets redirected to a phony IP address.

Domain Owner

For a domain owner, there is little you can do to protect against someone spoofing your domain name to a vulnerable nameserver. If you are running a webserver, consider using SSL to authenticate yourself to browsers. Eventually DNSSec will allow all domain servers to have cryptographically signed records, but it is not widely implemented at this time. Even detecting such an attack would be difficult, since the hijack would be largely independent of your servers. Be on the lookout for short denial-of-service attacks; they may indicate someone trying to slow your server down temporarily or crash it in order to complete a spoofing attempt.

ISP Nameserver Admin

You can upgrade BIND to the latest version in the 9.x series, which is not vulnerable to this attack. Alternatively you may try using djbdns, an alternative to BIND written by D. J. Bernstein, author of the MTA program qmail. The djbdns software comes with a security guarantee, basically offering a monetary reward to anyone who publicly discloses legitimate buffer-overflow vulnerabilities in djbdns. Although the guarantee doesn't cover cache-poisoning attacks, I will show later in this article that djbdns offers much greater protection against such attacks when deployed properly.

Disable recursive queries from the outside world, using split-split DNS if possible (see illustration 7). Split-split DNS means you have 2 nameservers; one to serve your public domain information to the outside world, and one to do recursive queries for your users. The public server should not allow recursive queries, and the recursive (caching) server should be protected from the Internet by a firewall.

This is sometimes confused with split DNS, where the internal server forwards requests to the outside server which makes recursive queries on its behalf. This arrangement offers no protection against cache poisoning, and should be avoided.

If you cannot use split-split DNS, you should at least try to restrict who can do recursive queries from your nameserver. Using the “allow-recursion” option doesn't give you very much protection - remember, the attacker is already spoofing the source IP address, so it is just a matter of them knowing which addresses are allowed to do recursive queries. If possible, use the “listen-on” option to bind the nameserver daemon to an interface that is protected from the outside world.

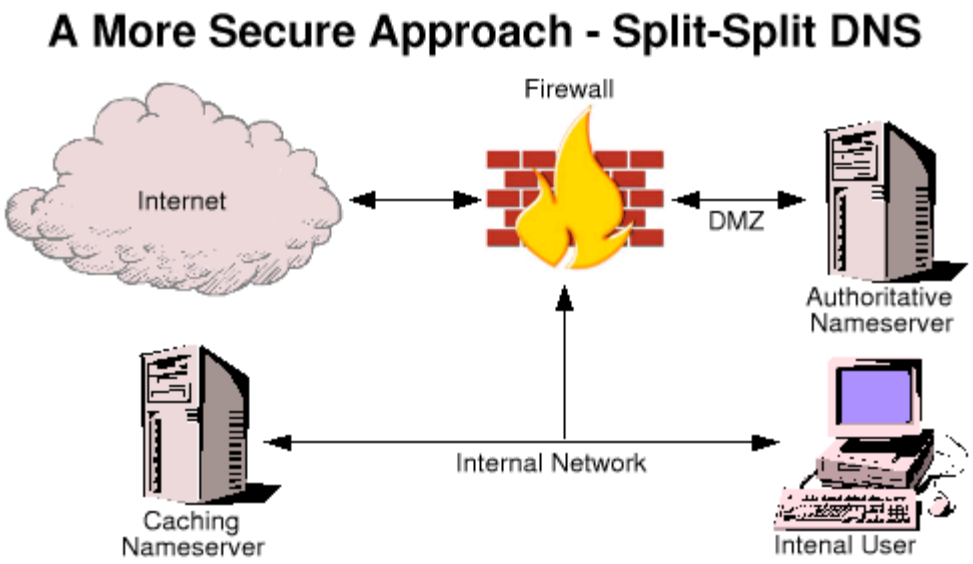

Illustration 7: Using split-split DNS to prevent cache poisoning

End User

Urge your ISP/company to upgrade BIND. If they are not open to this, you can always run your own recursive resolver and bypass the ISP's nameservers. Practice safe computing; keep antivirus software updated and signatures current. Always confirm SSL certificates when making secure online transactions. If you suspect a site is being spoofed, you can make use of the ARIN whois records to determine whether an IP address actually belongs to the organization that owns the domain name.

Vendor

This vulnerability could be remedied by changing the behavior of BIND 8 and 4 to only send one request for any number of queries for the same name. BIND 9 already does this, so it is more likely the vendor will just urge people to upgrade instead. However, BIND 9 also demonstrates the same behavior of reusing source ports. This should be corrected, as it would make any new attacks on the PRNG harder to execute.

Proof-of-Concept Code: spooftest.pl

I have written the following program to demonstrate the feasibility of the BIND Birthday attack. It is a passive demonstration only; it cannot be used to execute an actual attack. It will send *n* number of queries to a recursive server, while generating a "spoofing set" which is just an array of psuedo-random numbers. If any of the transaction IDs of the recursive queries coming back from the targeted nameserver match a number in the spoofing set, an attack would have succeeded.

```
#!/usr/bin/perl

##################################################################
# spooftest.pl                                                   #
```

```
# By Joe Stewart                                                    #
# Tests to see if a BIND server is vulnerable to the Birthday       #
# Attack spoofing technique described by Vagner Sacramento in       #
# http://www.rnp.br/cais/alertas/2002/cais-ALR-19112002a.htm        #
#                                                                   #
# This script MUST be run from an authoritative nameserver's IP     #
# address and must be running in place of the real nameserver       #
# daemon. It sends a number of queries and sees if it can guess     #
# a correct transaction ID from the target IP address. If it        #
# guesses correctly, the remote host is considered vulnerable       #
# to the BIND Birthday Attack.  It does not send any replies,       #
# so it is not capable of carrying out an actual attack             #
# (sorry script kiddies)                                            #
#####################################################################

use IO::Socket;
use strict;

$| = 1;
my $usage = "Usage: $0 [ip address] [number of packets]\n";

my $target = $ARGV[0];
my $m = $ARGV[1];

die "$usage" unless $ARGV[0] && $ARGV[1];
die "$usage" if $m !~ /^\d+$/ || $m == 0;

my $domain = "mydomain.com"; # domain name with NS RR pointing to us
my $r = 0xdead;              # initial transaction ID for our queries
my @spoofingset;             # list of our guesses
my $total = 65536;           # total possible packets
my $maxlen = 1500;
my $collisions = 0;
my $datagram;

printf "Probability of success using $m packets: %.2f%%\n",
        100 - (((1 - (1 / $total)) ** (($m * ($m - 1)) / 2)) * 100);

for (0..($m - 1)) {
        $spoofingset[$_] = sprintf("%x", int(rand($total - 1)));
}

#print "Spoofing set: ", join(" ", @spoofingset), "\n";

my $server = IO::Socket::INET->new(LocalPort => 53,
                                   Proto     => "udp")
    or die "Couldn't be a udp server on port 53 : $@\n";

my $client = IO::Socket::INET->new(PeerAddr => $target,
                                                PeerPort => 53,
                                   Proto   => "udp")
    or die "Couldn't be a udp client on port 53 : $@\n";

my ($second, $top) = split(/\./, $domain);
my $findlabel = chr(length($second)) . $second;

my $request = "\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x03" .
              "www$findlabel\x03$top\x00\x00\x01\x00\x01";

for (0..($m - 1)) {
  ## send query with incrementing transaction ID
  $client->send(pack("H*", sprintf("%x",$r++)) . $request);
}
print "Sent $m DNS initial queries to target...\n";
```

```
$SIG{'ALRM'} = sub { die "timeout"};
my $count = 0;
eval {
  alarm(30);
  while ($count < $m) {
    $server->recv($datagram, $maxlen);
    my $tid = sprintf("%x", hex(unpack("H*", substr($datagram,0,2))));
    printf "Received recursive query with transaction ID: $tid\r";
    for (@spoofingset) {
        if ($tid eq $_) {
                print "\nMatched TID $tid in spoofing set. Success.\n";
                $collisions++;
        }
    }
    $count++;
  }
  alarm(0);
};

if ($@) { if ($@ !~ /timeout/) { alarm(0); die $!; } }

print "\nReceived $count recursive queries for $m initial queries\n";

if ($count == 0) {
  print "Target not listening or does not answer recursive queries\n";
}
if ($count == 1) {
  print "Target does not appear to be vulnerable\n";
}
if ($collisions > 0) {
  print "Spoofing attack would have been successful with " ,
         "these parameters.\n";
} else {
  print "Spoofing attack unsuccessful in this run.\n";
}
```

About the Author

Joe Stewart, GCIH, is a Senior Intrusion Analyst with LURHQ Corporation, a managed security services provider located in Myrtle Beach, South Carolina.

References

Albitz, Paul, and Cricket Liu. DNS and Bind. 3rd ed. Sebastopol, CA: O'Reilly & Associates, Inc,1998.

Mockapetris, Paul. RFC 882 - DOMAIN NAMES - CONCEPTS and FACILITIES. Nov. 1983 <http://www.faqs.org/rfcs/rfc882.html>

Mockapetris, Paul. RFC 883 - DOMAIN NAMES - IMPLEMENTATION and SPECIFICATION. Nov. 1983 <http://www.faqs.org/rfcs/rfc883.html>

Schuba, Christoph. Addressing Weaknesses in the Domain Name System Protocol. Aug. 1993 <http://ftp.cerias.purdue.edu/pub/papers/christoph-schuba/schuba-DNS-msthesis.pdf>.

Sax, Doug. DNS Spoofing (Malicious Cache Poisoning). 12 Nov. 2000. 15 Dec 2002 <http://rr.sans.org/firewall/DNS_spoof.php>.

Internet Software Consortium.15 Dec. 2002 <http://www.isc.org/products/BIND/>.

Sacramento, Vagner. Vulnerability in the sending requests control of BIND versions 4 and 8 allows DNS spoofing. 19 Nov. 2002. 15 Dec. 2002 <http://www.rnp.br/cais/alertas/2002/cais-ALR-19112002a.html>

CERT/CC Vulnerability Note VU#457875. 5 Dec. 2002. 15 Dec. 2002 <http://www.kb.cert.org/vuls/id/457875>.

CERT Advisory CA-1997-22 BIND - the Berkeley Internet Name Daemon. 26 May 1998. 15 Dec. 2002 <http://www.cert.org/advisories/CA-1997-22.html>.

How does the “birthday paradox” work? 15 Dec. 2002 <http://www.howstuffworks.com/question261.htm>.

What is a Birthday attack? 15 Dec. 2002 <http://www.x5.net/faqs/crypto/q95.html>.

Schneier, Bruce. Applied Cryptography. 2nd ed. New York: John Wiley & Sons, Inc, 1996.

DNSSEC - Securing the Domain Name System. 15 Dec 2002 <http://www.dnssec.net/>

Zalewski, Michal. Strange Attractors and TCP/IP Sequence Number Analysis. 21 Apr. 2001. 15 Dec. 2002 <http://razor.bindview.com/publish/papers/tcpseq.html>.

D.J. Bernstein. djbdns: Domain Name System tools. 15 Dec. 2002 <http://cr.yp.to/djbdns.html>