Windows DNS Server Cache Poisoning

Amit Klein

March-June 2007

Abstract

The paper shows that Microsoft Windows DNS Server outgoing queries are predictable – i.e. that the source UDP port and DNS transaction ID can be effectively predicted, for the Windows DNS server (part of Microsoft Windows Server 2003 platforms and of Microsoft Windows 2000 Server platforms) in caching mode. A predictability algorithm is described that, in optimal conditions provides 8 possible guesses for the next transaction ID value, thereby overcoming whatever protection offered by the transaction ID mechanism. This enables a much more effective DNS cache poisoning than the currently known attacks against Windows DNS Server. The net effect is that pharming attacks are feasible against Windows caching DNS servers, without the need to directly attack neither DNS servers nor clients (PCs).

Table of Contents

1. Introduction

This paper describes a DNS cache poisoning attack on Windows DNS server. For a general historic overview of DNS cache poisoning, as well as relevant prior works and references, please refer to [1]. Hereby are listed only Windows DNS server - specific issues:

1. UDP source ports - UDP source ports are predictable for Windows DNS server implementation (the UDP port is unchanged, as briefly mentioned in [2] for Windows and was verified in the author's experiments). In fact, the source UDP port for outgoing queries from Windows DNS Server is static.

2. Historic notes and prior works pertaining specifically to Windows DNS Server:

- In April 1997, a paper (whose title is specific to BIND) was released [3], mentioning the problem of sequential transaction IDs in Windows NT 4.0 SP3 DNS server (this was reported to Microsoft by the authors of [3]). The issue was fixed in Windows NT 4.0 SP4 ([4]).
- In April 2001 a paper ([5]) was released, describing the use of a method called "attractors" to outline anomalies and predictability in numeric sequences. The paper mentioned that the Windows DNS server has a predictable transaction ID. This research was probably conducted for Windows NT Server 4.0 SP6a and/or Windows 2000 Server (those operating systems were used in the TCP sequence number research in the same paper). The "attractors" attack requires around 40,000 consecutive observations (transaction IDs) and 5,000 forged answers[1], which are usually not feasible (as explained in [1]), due to the combination of the attacker bandwidth limit and the race condition with the genuine DNS server.

To clarify: the rest of this discussion assumes Windows Server 2003 (or Windows 2000 Server SP4) DNS server, wherein all the old DNS vulnerabilities are fixed.

The attacks described in this paper make use of the predictable nature of Windows DNS Server transaction IDs to poison its cache. It is assumed that the DNS server can be forced to perform DNS queries using a malicious web page. This is a real-life condition, but of course it limits the attacker's activity scope – the attacker, for example, cannot force a burst of hundreds of queries all for the same hostname to be emitted from the same client. Nevertheless, it will be shown that since the transaction ID (and the UDP source port) is predictable enough, this suffices to mount a successful attack.

[1] Note that the original research in [5] made use of software which was later shown to contain 2 bugs. According to [5]'s author, those bugs turn out not to significantly affect the results of the paper (http://lcamtuf.coredump.cx/oldtcp/tcpseq/vseq-notice.txt).

2. Attacking the Windows DNS Server

1. Observations on the Windows DNS server

The Windows Server 2003 (and Windows 2000 Server) DNS Server (implemented as a service named "DNS Server") uses static UDP source port. This port number is acquired at the beginning of the service's run, and remains unchanged throughout the lifetime of the service. The DNS server generates a very predictable transaction ID. A partial analysis of the transaction ID was conducted, with the following results (results obtained for Windows 2003 Server Standard Edition SP2 and SP1, Windows 2003 for Small Business Server SP1):

Let *n* be a global counter, incremented on every DNS query to a new hostname. This counter is not advanced when the same hostname is queried in multiple name servers[2] (e.g. root name server, TLD name server, etc., as well as when the same query is sent to alternative name servers). The counter is set to 0 upon service startup and is advanced before it is polled for the ID calculation (so the first ID has *n* value 1)

Let *t* be the time (in integral seconds) at the DNS server.

Then the Transaction ID (16 bits quantity) is as following:

Bits 14…15 (2 bits) – zero (0)

Bits 11…13 (3 bits) – pseudo random data (M)

Bits 3…10 (8 bits) – n mod 256 (C)

Bits 0…2 (3 bits) – specified below (L).

That is, $\mathrm{ID}=M\cdot 2^{11}+C\cdot 2^{3}+L$

ID is serialized into the DNS ID header field as big-endian, i.e. bits 15…8 in the first byte and bits 7…0 in the second byte.

[2] This is an important observation. If DNS transaction IDs are extracted and analyzed by mathematical/statistical anomaly detection utility, these repeating values (due to the Windows DNS server reuse of the same transaction ID during the resolution of a single query, and due to retransmissions) may seriously skew the results and make them appear more anomalous (predictable) than they actually are. That happens because they "train" the utility to predict the same transaction ID for the next query. In real life, however, the attacker needs to predict the transaction ID of a query he/she never saw before, as opposed to the "follow-up" resolutions from an earlier query.

L is as following. Assume a 8×7 table $T[8][7]$ with values in the range 0…7. Then

$$L=(T[n \bmod 8][n \bmod 7]+n\cdot t) \bmod 8$$

Note that since $C = (n \bmod 256)$, it follows that $C \bmod 8 = n \bmod 8$, therefore:

$$L=(T[C \bmod 8][n \bmod 7]+(C \bmod 8)\cdot t) \bmod 8$$

The table T changes every few seconds. However, some invariants were observed (empirically), most notably among them is $T[0][i]=0$. This is a very important and useful result, since it guarantees that if (C mod 8)=0, then L=0.

Additional results are presented in appendix A. No significant results were obtained for the field M, so for the scope of this article, it can be treated as 3 random bits.

2. The basic attack

The attack target is an organization (e.g. a corporate, an ISP or an academic institute) with an internal Windows Server 2003 (or Windows 2000 Server) DNS caching server. This server does not answer DNS queries from the Internet, and no direct access to the internal network is available to the attacker. The goal of the attack is to poison the cache entry for the domain example.com. It is assumed that this domain is not yet cached (or that its cache entry has expired). The attacker needs to make the cache server cache the authoritative name server entry for example.com as the attacker's IP address, rather than the IP address of the real authoritative name server for example.com.

The attacker uses the predictability of the counter field (C), together with the predictability (in some cases) of the field L. Note that if, for example, the attacker can ensure that the DNS query to be spoofed is issued when (C mod 8) = 0 then it is guaranteed that L=0 in the query whose answer is to be spoofed. In the same way, if the attacker observes an L value of a DNS query with (C mod 8)=1, then the attacker knows that the next query will have an L value twice that much (mod 8), unless the server timer advanced by one second (or more). In information theoretic terms, when (C mod 8) = 7, the entropy of the next transaction ID (whose C mod 8 will be 0) is 3 bits (the 3 bits of M), instead of the theoretic maximum of 16 bits.

The attacker lures one of the network users to visit the attacker's web page. This page contains an image URL to, say, www1.attacker.com. Let's skip the part where the name server obtains the authoritative name-server for attacker.com and focus on the query for www1.attacker.com. It is sent to the attacker's name server. This name server observes the (C mod 8) value of the DNS transaction

ID. If it is not 7, it sends back a CNAME record for the next host name (i.e. a CNAME that points at www2.attacker.com). The Windows DNS server will then request www2.attacker.com with the next ID value. This process repeats itself (up to 7 times) until the (C mod 8) value is 7. At this point, the attacker name server returns a CNAME record that points to www.example.com. Note that altogether up to (and possibly including) 8 CNAME "redirections" were performed. Fortunately for the attacker, the Windows DNS server follows up to (and including) 9 CNAME redirections.

The above technique is called CNAME chains[3]. Windows DNS server handles CNAME chains (up to 9 "redirections") well, but will only return the first 8 CNAME records (i.e. the 9th CNAME will not be included in the response returned to the client). Therefore, when the chain contains up to (and including) 8 redirections, the response to the client will be functional, i.e. will include the IP address of the final CNAME.

Once the attacker redirects to www.example.com, the second phase begins. The attacker knows that the next (C mod 8) value will be 0, and hence L will be 0. So now the attacker needs to prepare 8 possible DNS answers, corresponding to the 8 possible M values, with a counter containing the value ((C+1) mod 256), and with the same UDP destination port (which is copied from the query source port), with source port 53, destination IP address being the request's source IP address, and the source IP address should be that of the name server for the .COM gTLD (which will be queried by the DNS caching name server for the www.example.com resolution).

The attacker can start sending those 8 DNS responses, as rapidly as possible, cycling through them again and again. Even with a modest 256Kbit uplink and even 150 bytes per response it is possible to achieve a cycle in less than 40 milliseconds. This increases the likelihood that the spoofed response (from the attacker's server) will reach the DNS server before the genuine DNS response (from the gTLD server).

The Perl script in Appendix B can be used to calculate the 8 guesses. Its runtime is negligible.

3. Attack variants

[3] CNAME chains are discouraged per the DNS RFC 1034 ([8]), section 3.6.2. Indeed, "standard" name servers eliminate such indirections from a static DNS configuration by resolving CNAME chains internally and providing a consolidated result. At the same time, CNAME chaining is in use by many good and respectable domains, e.g. when a domain uses Content Delivery Network (CDN) services it typically points at the CDN host (on a different domain) via a CNAME record. Therefore, to implement the above CNAME chain it is advised to use a name server which provides user-controllable runtime configuration, such as [7].

The CNAME chain is probably the most effective way to force the DNS server to rapidly iterate through IDs. However, it is not the only way to achieve this goal. It's possible (but less efficient) to force ID iteration via HTTP redirection (www1.attacker.com redirects to www2.attacker.com at the HTTP level, and so forth).

The CNAME chain can be also established via a "ping-pong" between two domains (this enables static configuration of the two name servers involved. So www1.attacker1.com will have a CNAME pointing at www2.attacker2.com, which will have a CNAME pointing at www3.attacker1.com, and so on. Just as with a single name-server chain, the Windows DNS server will not follow more than 9 redirections. In this case, it's harder for security systems to detect that an anomalous DNS activity is taking place, what with the fact that some legitimate DNS servers do answer with a CNAME pointing at a name outside their authoritative domain (e.g. when content delivery network host is used to deliver the content).

The basic attack only uses one empiric observation on the table T, namely that $T[0][i]=0$, in order to predict the next transaction ID in the transition from (C mod 8)=7 to (C mod 8)=0. But by leveraging the additional empiric observations on T (see appendix A), it is possible to also use the transition from (C mod 8)=1 to (C mod 8)=2, since with the first one L=(t mod 8) and the second one is (2·t mod 8), so the next L is twice (mod 8) the last L seen. That depends, however, on the two queries (the last seen and the next one) being performed on the same second from the DNS server's perspective. At the price of doubling the number of guesses, it's possible to account for one second shift by guessing two L values for the next query: twice the last L value seen (modulo 8), 2 plus that amount (modulo 8).

If the gTLD server is too close to the Windows DNS server to be poisoned (i.e. the round trip to it is very short), or if example.com's authoritative name server record is already cached by the Windows DNS server, it may still be possible to poison the A record for say www.example.com. All the attacks above should work just the same.

The attack can be also applied to improve the DNS cache poisoning attack in the DNS forwarder scenario. The original attack is explained in [2] (and an explanation about DNS forwarding can be found in [6]), and the suggested solution there is similar to standard DNS poisoning solutions, namely to prevent DNS queries from the external (Internet) network to the child server. However, with this attack there's no need to spray the child DNS server with hundreds of DNS requests. Therefore, this attack might be performed from a browser which renders a malicious HTML page. The attack should work as following: the child server (Windows 2003) is asked (by the client) to resolve www1.attacker.com. It queries the parent DNS server, which queries the attacker's name server. The attacker's name server does not respond, so the child DNS server queries it directly (not through the parent DNS server). Now the attacker's DNS server responds with a CNAME record for www2.attacker.com. The child DNS server again queries the parent which queries the attacker's name server, which again does not respond, and so on until the attacker's name server observes that (C mod 8) value is 7. At this time, the attacker's name server redirects to wwwfinal.attacker.com, and starts sending DNS response packets with spoofed source IP of the parent DNS server, and with additional data specifying DNS records such as a name server for the target domain. All this time, the attacker name server should not respond to any DNS query (specifically from the parent DNS server). According to the above text, the child DNS server would send a DNS query to the parent DNS server (with a predictable transaction ID) and will wait for the parent DNS server to answer it. It will more likely get the attacker's spoofed DNS answers, and thus will be poisoned.

3. Conclusions

To quote from [1] with the necessary adaptations, it is saddening to realize that 10-15 years after the dangers of predictable DNS transaction ID were discovered, still one of the most popular DNS cache servers does not incorporate strong transaction ID generation. It is particularly surprising that the transaction ID mechanism in use by Microsoft Windows DNS server is not based on industrial grade cryptographic algorithms.

The paper demonstrated that the "classic" DNS poisoning attack is still applicable to Windows DNS server. The attack described is far more effective than any attack previously described for Windows DNS. The attack does not require "query access" to the DNS server (except for a single triggering query). This is in contrast to the birthday attack, which requires a burst of hundreds of queries, rendering the birthday attack almost ineffective when Split-Split DNS configuration is used.

Usage of industrial-strength cryptographic algorithms is recommended for the DNS transaction ID generation. Furthermore, to strengthen the DNS query-response security, it is highly recommended to (strongly) randomize the DNS query source port (as also noted in many sources). Together, this would yield 30 bits of highly unpredictable data that needs to be spoofed, thus making DNS cache poisoning much less (if at all) feasible.

4. Further work

During this research, no static/dynamic reverse engineering techniques were used. In fact, the research was a pure exercise in packet analysis (few thousand DNS queries were used obtained and analyzed). As a result, while the findings are useful as-is, the following question remains:

How are *M* and *T* determined?

The research was conducted for the Windows 2003 platform. The analysis is applicable to Windows 2000 as well (and to wit, Microsoft released a patch for Windows 2000 SP4), and maybe even Windows NT Server 4.0 SP4 and above (note though that Windows NT 4.0 is no longer supported as of 2005), but this was not explicitly tested.

5. Disclosure timeline

April 30th, 2007 – Microsoft Security Response Center (MSRC) were informed of this issue.

November 13th, 2007 – Microsoft issues a fix (Microsoft Security Bulletin MS07-062) for Windows Server 2003 and Windows 2000 Server SP4. The fix is downloadable at Microsoft's website. Simultaneously, Trusteer discloses the vulnerability to the public (in the form of this document).

6. Vendor/product status

Windows Server 2003 – all versions and variants to date are vulnerable. A fix is available from Microsoft for the currently supported versions and service packs.

Windows 2000 Server – all versions and variants to date are vulnerable. A fix is available from Microsoft (for SP4 only, as it is the only supported version to date).

MITRE tracks this issue as CVE-2007-3898.

Microsoft documented this issue as Knowledge Base Article 941672 (Microsoft Security Bulletin MS07-062).

7. References

[1] "BIND 9 DNS Cache Poisoning", Amit Klein (Trusteer), July 2007

http://www.trusteer.com/docs/bind9dns.html (HTML)

http://www.trusteer.com/docs/BIND_9_DNS_Cache_Poisoning.pdf (PDF)

[2] "Windows DNS Cache Poisoning by Forwarder DNS Spoofing" (BugTraq mailing list submission), Makoto Shiotsuki, April 16th, 2007

http://www.securityfocus.com/archive/1/465882

[3] "BIND Vulnerabilities and Solutions" (Secure Networks Inc. and CORE Seguridad de la Informacion Security Advisory), Ivan Arce and Emiliano Kargieman, April 22nd, 1997

http://www.openbsd.org/advisories/res_random.txt

[4] "Predictable Query IDs Pose Security Risks for DNS Servers" (Microsoft Knowledge-Base Article 167629), June(?) 1997

http://support.microsoft.com/kb/q167629/

[5] "Strange Attractors and TCP/IP Sequence Number Analysis", Michal Zalewski, April 21st, 2001

http://lcamtuf.coredump.cx/oldtcp/tcpseq.html#otherp

[6] "Understanding forwarders" (Microsoft TechNet article)

http://technet2.microsoft.com/WindowsServer/en/library/a3cf0184-0594-4e78-8247-609f038434381033.mspx?pf=true

[7] "Stanford::DNSserver - A DNS Name Server Framework for Perl", Rob Riepel and other contributors (see http://www.stanford.edu/~riepel/Stanford-DNSserver/DNSserver.html#contributions)

http://www.stanford.edu/~riepel/Stanford-DNSserver/

[8] "DOMAIN NAMES - CONCEPTS AND FACILITIES" (IETF RFC 1034), Paul Mockapetris, November 1987

http://www.ietf.org/rfc/rfc1034.txt

Appendix A – Additional **empiric** properties of *T* and *M*

1. Properties of the table *T*

For i=0,…,7, the following holds:

$T[0][i]=0$

$T[1][i]=0$

$T[2][i]=0$

$2 \mid T[3][i]$ (i.e. $T[3][i]$ is always even)

$T[4][i]=(2 \cdot T[3][i]) \bmod 8$

$T[5][i] \neq 6$

$T[6][i]=f(T[7][i])$ where f is defined as

$f(0)=0$; $f(1)=4$; $f(2)=6$; $f(3)=2$; $f(4)= 0$; $f(5)=0$; $f(7)=6$

Also, there seems to be a very strong correlation between $T[3]$, $T[5]$ and $T[7]$.

Overall, the author managed to reduce the entropy of *T* from the theoretic maximum of 168 bits (3 bits per entry × 56 table entries) to less than 40 bits (the combined entropy of $T[3]$, $T[5]$ and $T[7]$ is slightly less than 40 bits). It is still quite a lot – this would theoretically require at least 13 DNS queries to fully reconstruct the table, and in reality much more (each cycle of 8 queries doesn't provide more than 8 bits of information, so around 5 such cycles are needed, i.e. 40 queries. It is probably reasonable to simply observe 56 consecutive IDs and easily extract the *T* table in fullness). Due to time constraints, it has not been possible to complete the research and fully understand how *T* is constructed, and how (and when) it changes over time. Yet the results obtained so far enable a very effective attack, as will be seen below.

For example, notice that for two samples L_j and L_{j+56} at counter values j and $j+56$ respectively, and at times t_j and t_{j+56} respectively, the following holds (unless the table *T* changed in between the samples):

$$(L_{j+56} - j \cdot t_{j+56}) \bmod 8 = (L_j - j \cdot t_j) \bmod 8$$

In other words, there's a cycle of length 56 for $(L - j \cdot t) \bmod 8$.

2. Properties of the field *M*

A cyclic behavior (cycles of 56) was observed for the field *M*, but it’s more complicated than the cycles of *L* (above). Again, for lack of time, a full research was not conducted, although it should be quite possible to reduce the entropy in its case as well.

Appendix B – prediction script for Windows DNS server

This Perl script accepts one command line argument (the transaction ID, as a decimal number) and produces the list of 8 guesses for the next transaction ID, provided that the transaction ID’s *C* obeys (*C* mod 8)=7.

```
$TRXID=$ARGV[0];
$zero=$TRXID>>14;
if ($zero!=0)
{
        print "Highest two bits are not 0.\n";
        print "Is this really Windows DNS server? check endian issues!\n";
        exit(0);
}
$M=($TRXID>>11) & 7;
$C=($TRXID>>3) & 0xFF;
$L=$TRXID & 7;
if (($C % 8)!=7)
{
        print "C mod 8 is not 7 - can't predict next TRXID.\n";
        print "Wait for C mod 8 to become 7\n";
        exit(0);
}

print "Next TRXID is one of the following 8 values:\n";
for ($m=0;$m<8;$m++)
{
        print "".(($m<<11)|((($C+1) % 256)<<3))." ";
}
print "\n";
```