Dan Kaminsky's 2008 DNS Vulnerability

Sourcefire Vulnerability Research Team Report

SOURCEFIRE, INC

July 25, 2008

Authored by: Matthew Olney, Patrick Mullen, Kevin Miklavcic

Dan Kaminsky's 2008 DNS Vulnerability

Sourcefire Vulnerability Research Team Report

Executive Overview

This document reviews the currently understood details of the DNS spoofing attack developed by Dan Kaminsky. While Mr. Kaminsky has not provided the technical details of the bug, a leak by Matasano Security is widely believed to be an accurate representation of the attack. We will review the relevant parts of the DNS architecture and lay out the details of the attack as they are currently understood.

In short, the attack targets recursive name servers; that is, those name servers that will receive a request from a client and then forward it to an authoritative name server. The attack combines a previously undisclosed method to poison the cache of the recursive server to point to a fake authoritative name server and a new way to ensure that the poisoning attack succeeds quickly. A successfully executed attack will, from the point of view of the targeted name server, hijack an entire domain and allow the attacker to dictate where clients will go.

In this way, an attacker could hijack bigbank.com, and point all clients using the exploited name server to a fake server constructed by the attacker. This could allow further compromise, capture of username/password combinations, and other attacks. This is only one path an attacker could take: she could also grab all mail going to a particular domain, reroute chat traffic, or almost anything else that relies on domain name resolution to be successful.

DNS Architecture

There are two DNS packet types that need to be looked at for this vulnerability, the query and the response. Both of these packet types have the following data structure:

Field	Notes
Identification (TXID)	2-byte field uniquely identifying the query
Flags	2 byte field describing the nature of the packet
Number of questions	2-byte field with number of resources requested
Number of answer RRs	2-byte field with number of resource records
Number of authority RRs	2-byte field with number of authority RRs
Number of additional RRs	2-byte field with number of additional RRs
Questions	Variable length field with questions listed (name, type, class)
Answers	Variable length answer RR, variable # of records
Authority	Variable length authority RR, variable # of records
Additional information	Variable length additional RR, variable # of records

The DNS query packet is identified by the flags field. The flags typically are set to indicate that the message is not a response (and therefore a query), and that recursion is requested. Additionally, there will be no RRs set, and usually only one question. Finally, the actual question will populate the "Questions" field. The request will be tagged with a 16-bit random number in the identification field, referred to as the TXID.

A DNS response field will have a different set of flags, with the most significant bit set to indicate that the message is a response. From non-authoritative servers, the flags for a response will typically be set to indicate that the packet is a response, and include error codes, if necessary to indicate any problems with the answer. We check this flag field as part of our open source detection. (See "Detection Methodology", below). Additionally, the number of questions will be set to one and place the original question in the questions field. A list of the names of the authoritative name servers for the domain will be placed into the authority field and the IP addresses of those names will be placed in the authoritative field. As a simple example, see the dig for www.sourcefire.com below:

```
[vrt@server]$ dig www.sourcefire.com

; <<>> DiG 9.2.4 <<>> www.sourcefire.com
;; global options:  printcmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 21682
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 3

;; QUESTION SECTION:
;www.sourcefire.com.       IN      A

;; ANSWER SECTION:
www.sourcefire.com.  300    IN      A       68.177.102.22

;; AUTHORITY SECTION:
sourcefire.com.          300     IN      NS      ns1.sourcefire.com.
sourcefire.com.          300     IN      NS      ns2.sourcefire.com.
sourcefire.com.          300     IN      NS      ns3.sourcefire.com.

;; ADDITIONAL SECTION:
ns1.sourcefire.com.          76040   IN      A       68.177.102.19
ns2.sourcefire.com.          76040   IN      A       199.107.65.180
ns3.sourcefire.com.          76040   IN      A       64.214.53.11
```

As described above, the original question (www.sourcefire.com, type A, class IN) populates the question field. The answer to our question, that www.sourcefire.com can be found at 205.178.187.13, is available in the answers field. The authority section lists the three authoritative servers that handle sourcefire.com DNS traffic. Finally the resolved addresses of those authoritative servers are placed in the additional section.

The second piece to look at before we move to the actual attack is the role of recursive DNS servers on the Internet. Most clients are configured to use only 2, or perhaps 3, name servers. The name servers that are used by clients typically wouldn't have the answer to a question about sourcefire.com. Therefore they have to repeat the query (recurse) by asking a set of questions. The steps are:

1) Client asks recursive name server where www.sourcefire.com is.
2) Recursive name server asks a "DNS Root server" (which is responsible for pointing recursive name servers to the authoritative name servers for a domain) where to find authoritative name servers for sourcefire.com.
3) The root server points the recursive name server towards the authoritative DNS servers for sourcefire.com.
4) The recursive name server then asks the authoritative DNS servers for sourcefire.com for the address of www.sourcefire.com
5) The authoritative DNS servers for sourcefire.com then reply with the address for www.sourcefire.com (68.177.102.22).
6) The recursive server then replies to the client with the correct IP address of www.sourcefire.com

Next, we need to understand one of Dan Kaminsky's prior tricks, CNAME-forced cache updating. A CNAME (canonical name) entry simply means that one name is pointed to another. For example, ftp.sourcefire.com could point to www.sourcefire.com. Dan discovered that you could force a cache update by using the CNAME record type. When a CNAME request is made to an authoritative name server, it sends out a response that includes the alternate name the CNAME is pointed to, as well as the IP resolution to that name. For example, if a.sourcefire.com had a CNAME entry pointing to www.sourcefire.com, the response would be:

a.sourcefire.com	CNAME	IN	www.sourcefire.com
www.sourcefire.com	A	IN	68.177.102.22

The trick here is that even if the recursive name server has a non-expired entry in its cache for www.sourcefire.com, the server will honor the information provided and update the cache entry for www.sourcefire.com. An attacker could forge a response to any request, replying in CNAME fashion and forcing a cache update to a fake server. Dan calls this attack "CNiping". One of the additional pieces of information that has come out with this attack is that this cache poisoning will also work with name server entries, which use the NS record type, similar in structure to a CNAME record:

sourcefire.com.	300	IN	NS	ns1.sourcefire.com.
ns1.sourcefire.com.	76040	IN	A	68.177.102.19

These entries are always paired with an A record resolution in responses. This means that a successfully forged response can rewrite the location of the name servers in cache to point to a fake name server.

The final piece of the DNS architecture we need to touch on is the cache. This is the target of the attack. In the above scenario, the recursive name server stores the information it has learned in a cache. Subsequent questions about www.sourcefire.com will result in the server retrieving the known value 68.177.102.22 from the cache. More importantly, with regard to this attack, the server also stores the information about the domain name servers in the cache. This way if a question about ftp.sourcefire.com is received, it doesn't have to go back to the DNS root servers for the information again.

Kaminsky's Vulnerability

The vulnerability as it currently is understood revolves around the recursive name server. If an attacker can somehow forge a packet correctly, and beat the true authoritative server's response, then the attacker can use the recursive server to point the client to an arbitrary IP address. There are a number of things that stand in the way of the attack. In order for the attack to be successful the following things must match up in a forged reply packet:

1) Source and destination IP
2) Source and destination port
3) Transaction ID
4) Question

Because we're dealing with UDP, getting the source and destination IP addresses correct is trivial. Because we're dealing with DNS, getting the ports right is also easy, because the source port will always be 53 and many DNS servers (at least up until the recent patches) would use a static or easily guessable destination port. Since we can target a given host, and trigger a query at will on the recursive server, we can also get the question easily. That leaves us to guess a 16-bit random number.

Another problem to overcome is that if we want to spoof www.sourcefire.com, it is likely that the server has the information in its cache already. If the recursive server is queried for www.sourcefire.com and it is in the cache, it won't ask the authoritative name server. So we have to ask a question that we know isn't in the cache. But if the record isn't in the cache, then it's probably not a very desirable target. However, by combining the cache-poisoning with the ability to force cache updates through NS and CNAME records we have something much more useful.

Kaminky's discovery seems to hinge on three things:

1) Given today's Internet speeds, the attacker can send enough guesses about a Transaction ID to render that protection useless
2) The recursive name server will always contact the authoritative name server if the attacker looks up a random (and most likely, non-existent) host, such as alksdjfaldf.sourcefire.com.
3) If the attacker correctly spoofs the packet, then the attacker can overwrite the cache with values in the additional RR field.

Here is how the attack would work:

1) Attacker figures out what the destination port needs to be, this can be done fairly easily, for example, here is an output from Dan Kaminsky's DNS test at www.doxpara.com:

```
a.b.c.d:4318 TXID=29841
a.b.c.d:4318 TXID=55271
a.b.c.d:4318 TXID=63125
a.b.c.d:4318 TXID=34926
a.b.c.d:4318 TXID=55054
```

   This server is judged to be vulnerable because, in all likelihood, the next time the server asks a question, the response will probably be to port 4318.

2) The attacker would send a very large number of DNS queries to the recursive server, each individual request asking about a randomly named host, flkapoias.sourcefire.com for example. It does this to ensure the server will make a request out to the authoritative name server.
3) The attacker then immediately follows each request with a forged reply to the anticipated destination port with a guess at the TXID. While for any given packet the odds of correctly guessing a 16-bit number are low, given the volume of traffic, estimates are that with a speedy Internet connection and a vulnerable DNS server, the attack can be successful within 10 seconds.

Once the attack succeeds what have we gained? We have spoofed a randomly generated host, for example Bu09aVJLC1r.sourcefire.com. But the key is what additional information we gave the server. In each response, we forged an authoritative name server entry:

```
sourcefire.com    NS      IN      attacker
```

We also forged an additional record for attacker:

attacker A IN [attackers DNS address]

Now, any time the server needs to ask a question about sourcefire.com, it will query the attacker's DNS server. In essence, the attack hijacks the entire domain from the standpoint of the exploited recursive name server. All future requests for names within the targeted domain will be answered by the attacker's DNS server, sending clients to any arbitrary IP address specified by the attacker.

The attack is a combination of each of these methods. By using today's faster Internet speeds, combined with an understanding of the weakness in source port selection on vulnerable DNS servers, the attacker radically reduces the time for a successful cache-poisoning attempt. By choosing to query randomized host names, the attacker avoids the problem of the host already being in the cache, and can query as quickly as possible by cycling through random names. Finally, by using the Additional Resource Record section to overwrite previous entries in the cache, the attacker is able to target important hosts, such as authoritative name servers, and potentially hijack an entire domain from the perspective of a single recursive name server. Customers should be aware that code to exploit this vulnerability has been made public, including a module for the Metasploit Framework.

Detection Methodology

From a detection standpoint, we currently offer SID: 13667, a closed source SO rule that detects DNS cache poisoning attempts. We have supplemented this detection with a new open-source rule:

```
alert udp $EXTERNAL_NET 53 -> $HOME_NET any (msg:"DNS large number of
NXDOMAIN replies - possible DNS cache poisoning"; byte_test:1,&,2,3;
byte_test:1,&,1,3; byte_test:1,&,128,2; threshold:type threshold, track
by_src, count 200, seconds 30; metadata:policy balanced-ips drop, policy
security-ips drop, service dns; reference:cve,2008-1447;
reference:url,www.kb.cert.org/vuls/id/800113; classtype:misc-attack;
sid:13948; rev:1;)
```

This rule actually detects the "backscatter" of the attack. Backscatter refers to traffic from an external source not directly associated with the attack. In this case, we are keying in on the very high number of NXDOMAIN replies that will be seen from the actual authoritative domain server. An NXDOMAIN response is sent with a query is made for a non-existent host. Because the attack sending queries for randomly generated hosts, this will probably generate a large number of NXDOMAIN responses from the valid authoritative server. A network that sees a sudden flood of NXDOMAIN replies may be the target of this attack, and should check their recursive name servers. Also note that the threshold may need to be adjusted, depending on the nature of normal DNS traffic in a given network.

As a note, this rule should be set to "alert" only, not drop. If this rule were set to drop, it would be dropping legitimate traffic that wasn't actually part of the attack, and therefore would provide no actual defense.

Some companies have indicated a desire to attempt to track spoof attempts against domains for which they are authoritative. In this case you would reverse the above rule, and track the number of NXDOMAIN responses your servers are providing. In a case of a large increase in responses with NXDOMAIN, a recursive server on the Internet may be the target of an attack involving the domain you are authoritative for. Here is the published rule, note that it is disabled by default:

```
# alert udp $HOME_NET 53 -> $EXTERNAL_NET any (msg:"DNS excessive outbound
NXDOMAIN replies - possible spoof of domain run by local DNS servers";
byte_test:1,&,2,3; byte_test:1,&,1,3; byte_test:1,&,128,2; threshold:type
threshold, track by_dst, count 200, seconds 30; metadata:policy security-ips
drop, service dns; reference:cve,2008-1447;
reference:url,www.kb.cert.org/vuls/id/800113; classtype:misc-attack;
sid:13949; rev:2;)
```

Conclusion

By combining previously known weaknesses with new attack vectors and the ability to manipulate the DNS cache, Kaminsky has identified a serious issue with DNS. Vendors have released patches for their DNS systems and have urged customers to immediately patch. Snort has coverage available to detect the attack, but with details slow to emerge, and an upcoming presentation from Dan Kaminsky at Black Hat, the VRT may release additional detection in the coming days.

Additional References:

DNS bug leaks by matasano. (2008, July 25). Retrieved July 25, 2008, from Beezari: http://beezari.livejournal.com/141796.html

http://www.matasano.com/log/mtso/. (2008, July 25). Retrieved July 25, 2008, from Matasano Security: http://www.matasano.com/log/mtso/

Kaminsky (finally) provides DNS flaw details. (2008, JULY 25). Retrieved July 25, 2008, from News - Security - CNET News.com: http://news.cnet.com/8301-1009_3-9998906-83.html

Kaminsky, D. (2008, July 25). *DoxPara Research*. Retrieved July 25, 2008, from DoxPara Research: http://www.doxpara.com/

Mook, A. H. (2008, July 28). *Measures for making DNS more resilient against forged answers.* Retrieved July 28, 2008, from IETF DNS Extensions: http://tools.ietf.org/pdf/draft-ietf-dnsext-forgery-resilience-05.pdf

Acknowledgments:

Thanks to John Pritchard for pointing out some consistency issues in the provided examples. Based on this feedback, a revision to this document was released August 25, 2008.