Rapid DNS Poisoning in djbdns

Kevin Day

February 09, 2009

Abstract

The popular DNS server package *djbdns* is vulnerable to DNS cache poisoning in considerably less time than previously believed. Dnscache (included with the djbdns suite) can be coerced into producing repeated simultaneous identical queries. This allows an attacker to exploit a "birthday attack", allowing poisoning of arbitrary DNS records in hours – or even minutes – using relatively low speed attacks.

Cache poisoning may lead to session hijacking, unauthorized information leakage, or other circumstances resulting from users or systems being redirected to systems under the attacker's control.

The Common Vulnerabilities and Exposures (CVE) project has assigned the name CVE-2008-4392 to this issue. This is a candidate for inclusion in the CVE list (http://cve.mitre.org), which standardizes names for security problems.

1 Introduction

The djbdns DNS server package has cache policy vulnerabilities which allow an attacker to significantly shorten the amount of time required to launch a brute force DNS poisoning attack. All of the vulnerabilities below are exploitable using a modified version of the 2008 "Kaminsky" technique[1] for cache poisoning, combined with the "Birthday Attack" vulnerabilities published in 2002 under US-CERT VU#457875.[2]

In each case, one or more vulnerabilities may allow an outside attacker to successfully poison a DNS server's cache many times faster than previously believed possible. Cache poisoning may lead to session hijacking, unauthorized information leakage, or other circumstances resulting from users or systems being redirected to systems under the attacker's control.

Resolving/recursive servers that are vulnerable to this attack may have records poisoned within minutes using a relatively low sized flood. This is extremely dangerous, allowing an attacker to redirect arbitrary hosts or domains to his control, including top-level domains such as ".com".

These attacks are possible by exploiting weaknesses in these servers' handling of multiple simultaneous identical queries, often referred to as a "birthday attack". To successfully convince a DNS server to accept an attacker's forged answer to a query, the attacker must successfully guess the server's random chosen source port number and packet ID that matches the server's open query. By coercing a DNS server to open multiple identical queries, the chances of a successful match between any of the server's open queries and the attacker's spoof attempt increase drastically.

2 Attack Summary

The currently shipping version of *djbdns*(1.05), contains a caching DNS server called *dnscache*. Dnscache has three distinct vulnerabilities allowing cache poisoning.

The first vulnerability is a failure to prevent simultaneous requests for the same question. The second is that SOA requests are considered completely uncacheable. Both allow an attacker to hold open 200 (or more) simultaneous queries for an identical question. This increases the likelihood of a spoofing

[1] http://www.doxpara.com/?p=1185
[2] http://www.kb.cert.org/vuls/id/457875

attempt to collide with an in-progress query by 20,000%, which reduces the number of attempts to poison dnscache from 2 billion to around 16 million, or even less if the server's default configuration has been changed.

The last official version of djbdns was released in 2001, but is still in heavy use on the Internet. As of 2007-12-28, djbdns has been placed completely in the public domain by its original creator, Daniel J. Bernstein. After discovering these vulnerabilities, several large-scale users of djbdns collaborated on creating and testing patches. Respecting Mr. Bernstein's original work, notification and copies of our proposed fix were made available to him immediately. While it's unclear when an updated version of djbdns will be released, we have freely released these patches to the public at http://www.your.org/dnscache.

The third vulnerability in djbdns is related to dnscache's outstanding query queue. By default, dnscache allows 200 queries in progress. If all 200 slots are full and another client request is received, dnscache silently drops the oldest query in the queue. Clients waiting for a reply to a query that was dropped receive nothing from the server at all, and recover through a timeout system. This is a reasonably well known Denial of Service attack. However, when combined with the above mentioned failure to combine identical queries, all queries (not just uncacheable SOA queries) are vulnerable to a sustained poisoning attack.

Altering dnscache's behavior in this manner is outside the scope of a third-party security patch. However, it is believed that the above patch to combine identical queries prevents the rapid poisoning vulnerability associated with dnscache's queue architecture.

3 Unaffected Packages

3.1 BIND

At this time, it is believed that all recent BIND versions are not vulnerable to this type of attack.

3.2 OpenDNS

OpenDNS is a free service offering open resolving nameservers to the public. OpenDNS's proprietary server software is originally based on djbdns's dnscache. Due to specific countermeasures recently implemented by OpenDNS's development team, OpenDNS is not vulnerable to this type of attack.

4 Impact

4.1 Assumptions

Assume an attacker is attempting to poison the cache of one of the affected servers listed below. To avoid attention, the attacker limits the attack to 10 megabits/sec. This allows for roughly 15,000 attempts per second during the attack.

4.2 Rapid Results

An attacker can cause dnscache to open a high number of simultaneous identical queries. By default, an administrator setting called *MAXUDP* allows 200 simultaneous queries. The likelihood of each attempt an attacker makes is multiplied by the number of outstanding identical queries. This improves the odds of success of each spoofed packet from $\frac{1}{65536*64512}$ (1 in 4 billion) to $\frac{200}{65536*64512}$(roughly 1 in 32 million). [3]

[3] 65536 possible Query ID numbers, and 64512 ports. dnscache does not attempt to use UDP port numbers 0 through 1023.

[4] Note that dnscache was not architected for queues above 200. The linear search algorithms used do not scale well to huge MAXUDP values, causing severe performance degradation on current CPU speeds. Values above 2000 are theoretical and assume the CPU is capable of keeping up.

MAXUDP	Attack Bandwidth	Attack Packets/sec	Packets until success	Success time
200	10Mbps	15000	~16.3M	18 mins
200	100Mbps	150000	~16.3M	109 secs
2000	10Mbps	15000	~1M	70 secs
2000	100Mbps	150000	~1M	7 secs
64000[4]	10Mbps	15000	~33K	2 secs
64000	100Mbps	150000	~33K	0.2 secs

Table 1: Depending on how the administrator has configured MAXUDP, the impact can be drastic.

4.3 Comparison to other DNS servers

BIND makes use of countermeasures specifically designed to avoid birthday attacks such as these. BIND will never allow more than one identical request to be sent to another server at once. This would require an attacker to guess the correct combination of Query ID (16-bit number, 2^{16} possibilities) and port number (64512 ports on default configurations). This means there are (65536 * 64512) = 4,227,858,432 possibilities. At 15000 guesses per second, this takes nearly 2 days of sustained attack to have a 50% chance of success.

Server	Success packets	Success time
djbdns using Kaminsky's attack	~2B	40 hours
djbdns using this attack (MAXUDP=200)	~16.3M	18 mins
djbdns using this attack (MAXUDP=2000)	~1M	70 secs
BIND with source port randomization	~2B	40 hours

Table 2: Time until 50% chance of success, using a 10mbps attack.

5 Technical Details

5.1 dnscache allows duplicate simultaneous queries.

As mentioned in section 4.2, if a DNS server allows multiple identical queries at once, each DNS poisoning attempt has a chance to collide with any open query. If an attacker can convince a nameserver to open 10 identical requests to another name server, spoofed packets appearing to be from the correct source could potentially collide with any open request. This multiplies the attacker's chances of successfully colliding by 10.

dnscache's protection against this attack is through its caching mechanism. After the first response from the remote nameserver is accepted, all subsequent requests are served purely from the cache – eliminating the opportunity for poisoning. However, there is a window between the time the first request is sent until the first reply is received. Any additional requests from a client that are identical to the request already in progress will result in another query being opened to the remote name server.

While this window is brief, it is possble to quickly send 200 identical requests back-to-back to the victim nameserver. This will open 200 identical queries on dnscache. An attacker then may send a volley of spoofed packets hoping to collide with any of those 200 open queries. Each spoofed packet received before the responses come in has a roughly 200x greater chance of successfully poisoning dnscache than it would through a normal attack.

Here, any successful match of ID and port number will work, since all of the spoofed replies are to identical name/type queries. The attacker just has to win one race, but it doesn't even matter which of the 200 simultaneous races going on is won.

Fix: A patch has been created to prevent multiple identical questions from being sent at once. Subsequent identical requests "piggy-back" onto the first request.

5.2 dnscache does not cache SOA records.

dnscache intentionally never caches a "SOA" type record. SOA (Start of Authority) records can be used for multiple DNS servers belonging to the same organization to coordinate updates between each other (zone transfers), as well as describing the zone's cache timeouts and contact information. One of the purposes of a SOA record is for a server to say how recent its information is, through a "serial number". If you're asking a server how recent its data is, you generally don't want a resolver to get in the way and give you its old version of that. However, this opens dnscache up to an attack.

As mentioned in 5.1, dnscache relies on its cache to prevent sustained floods of simultaneous requests. But, SOA records are never cached. An attacker can send a constant flood of SOA queries to dnscache, keeping the simultaneous query window open indefinitely. The new attack is as follows:

- An attacker sends 200 SOA requests for example.com, the domain he intends to poison.
- dnscache opens 200 requests at once to example.com's servers to look up the SOA record for example.com
- The attacker now has 200 possible responses to try to collide with.
- While the attack is going on, the attacker periodically throws in more SOA requests to the victim, to ensure 200 requests stay open at all times.

This attack requires approximately 16 million spoofed packets to have a 50% chance of winning. This is also slightly more efficient than the traditional "spoof with added glue" attack, since the question in the spoofed replies stays the same throughout the attack. It is no longer necessary to try to guess how long the window stays open, allowing the attack to have no wasted packets or missed opportunities. While small, it increases the efficiency of a poisoning attack by several percent.

Fix: A patch has been made available to make SOA records cacheable, but only in response to a direct SOA request.

5.3 dnscache's head-drop query list allows harmful query eviction.

Making SOA responses cacheable is a relatively simple change to dnscache, but it doesn't completely prevent the problem. When a request is received that would cause dnscache to exceed its open query limit, the oldest open query is dropped. If new queries for any question (cacheable or not, as long as it isn't currently cached) are being received faster than the oldest reply is received dnscache will never accept any answers from the real server. Example:

- Attacker sends 200 back-to-back requests off to a dnscache server, for any query not currently cached. Any current requests the server was working on are dropped, and now 200 requests are sent out.
- Attacker sends a few hundred spoofed replies to the victim, then 5 more queries for the same resource.
- The 5 oldest queries from the initial 200 are dropped, and 5 more get sent out.
- The attacker continues alternating between a load of spoofed replies and a few new queries.
- The real replies start trickling in, but the first 5 queries are already invalid on dnscache's side so they're dropped.

As long as the attacker can send new queries faster than the real responses are coming in, the window of 200 queries will stay open until the attacker succeeds ($\sim$16 million tries, with a high packet rate), the real server accidentally collides with a new query (1 in 4 billion chance, with a lower packet rate), or the attacker slips up and a real reply makes it in before the query was evicted. In that case the attacker can just change to a different question name and start again.

Fix: Changing the behavior of dnscache's queue management is beyond the scope of a third-party security patch. However, it is believed that the identical-query-merging patch from section 5.1 closes

the poisoning attack. Further research is needed to determine if alternate queuing strategies would be beneficial during high queries-per-second floods.

Patches for dnscache are available at http://www.your.org/dnscache.

6 Proof of Concept

Below is a description of how dnscache was successfully exploited. The domain to be hijacked has two nameservers that are authoritative for it, NS1 and NS2. They are both running tinydns from djbdns-1.05, the latest currently available.

They publish one A record, the correct IP for www.example.com, 1.2.3.4:

```
;; QUESTION SECTION:
;www.example.com. IN A

;; ANSWER SECTION:
www.example.com. 86400 IN A 1.2.3.4

;; AUTHORITY SECTION:
example.com. 259200 IN NS ns1.example.com.
example.com. 259200 IN NS ns2.example.com.
```

The intended victim is running dnscache from djbdns-1.05. The victim takes approximately 120ms to get a response from the authoritative servers, due to their geographic distance and network latency. The attacker is physically located in a different datacenter in the same city, with approximately 20ms latency between it and the victim. This simulates a customer/ISP relationship, showing how a customer could poison their own ISP's server relatively easily.

6.1 Attack

The attacker sends 200 SOA requests to the victim, in order to immediately fill as many outgoing UDP slots as possible. It then begins sending SOA requests and spoofed SOA replies in a pattern; 75 spoofed SOA replies, followed by another SOA request. The exact numbers don't matter, but the goal is to keep all 200 UDP requests on the victim busy as close to 100% of the time as possible. The spoofs alternate between the source IPs for NS1 and NS2.

A legit reply for the SOA record contains:

```
;; QUESTION SECTION:
;example.com. IN SOA

;; ANSWER SECTION:
example.com. 2560 IN SOA ns1.example.com. hostmaster.example.com. 1217307566 16384 2048 1048576 2560

;; AUTHORITY SECTION:
example.com. 259200 IN NS ns1.example.com.
example.com. 259200 IN NS ns2.example.com.

;; ADDITIONAL SECTION:
ns1.example.com. 86400 IN A 1.4.4.4
ns2.example.com. 86400 IN A 1.5.5.5
```

A spoofed reply for the SOA record contains:

```
;; QUESTION SECTION:
;example.com. IN SOA
```

```
;; ANSWER SECTION:
example.com. 2560 IN SOA ns1.example.com. hostmaster.example.com. 1217307566 16384 2048 1048576 2560

;; AUTHORITY SECTION:
example.com. 259200 IN NS ns1.example.com.

;; ADDITIONAL SECTION:
ns1.example.com. 86400 IN A 2.1.1.1
www.example.com. 86400 IN A 2.2.2.2
```

Here, we replace www.example.com with an IP of our choosing, and for good measure replace one of the nameservers with our own IP too. We could easily replace this with replacing both nameservers, replacing several other hosts, MX records to redirect email or whatever we choose. After a successful poisoning, dnscache believes the new addresses listed in the spoof section, and will cache them for as long as we ask.

6.1.1 Results

Run	Success after	Time
1	32,100,000 spoofs	36 minutes
2	21,700,000 spoofs	24 minutes
3	38,100,000 spoofs	41 minutes
4	31,000,000 spoofs	34 minutes
5	24,500,000 spoofs	26 minutes

Table 3: Results after 5 test runs, at ∼15000pps.

An average of 29.48 million requests were needed to successfully poison dnscache. Theoretically, ∼16 million tests are required to get a 50% chance of colliding on one server. With two servers being randomly queried (both ns1 and ns2 getting requests), this should have at least doubled the trials to at over 32 million tests. The results received are approximately equal to what is predicted.

7 Workarounds

Install patches to correct these vulnerabilities as soon as they are made available from your vendor.

While none of these strategies is completely effective, each may increase the complexity in launching a successful attack.

- Do not modify parameters such as MAXUDP without carefully considering the consequences. The default limit prevents the impact of these weaknesses from being much worse.
- Enable recursion only to networks within your organization's control.
- Filter traffic at your network's borders to prevent outsiders from spoofing your own authoritative servers, or spoofed queries hitting your recursive servers.
- Employ the recommendations in BCP38/RFC2827 to prevent your network's users from launching spoofed attacks at others.

8 Acknowledgements

The author of this document would like to thank David Dagon, Adam Getchell, Dan Kaminsky, Jeff King, David Ulevitch and many others for their assistance in responsibly bringing this problem into public view.