DNS Extensions Working Group	W. Wijngaards	
Internet-Draft	NLnet Labs	
Intended status: Informational	February 24, 2009	
Expires: August 28, 2009		

TOC

Resolver side mitigations
draft-wijngaards-dnsext-resolver-side-mitigation-01

Status of This Memo

Copyright Notice

Abstract

This document describes a set of mitigations that stop the known variations of the Kaminsky cache poisoning attacks against the DNS system, for which only resolver side deployment is necessary.

Table of Contents

---

1. Introduction

TOC

[WW: These are the counter measures for the Kaminsky attack scenarios that I envision for the Unbound resolver (http://unbound.net). These are counter measures that require resolver side deployment only. Depending on working group input this document could remain an Unbound specific information document or can be made more generic, and move towards a BCP.]

This document describes the mitigations that a resolver can deploy on its own in the meantime, while a more comprehensive (read: DNSSEC) solution is being rolled out. For counter measures that require changes to authoritative and recursive servers everywhere, DNSSEC provides the most protection, followed by Nonce-based approaches (e.g. EDNS PING), followed by transport protocol games. Because Unbound implements DNSSEC validation already, and DNSSEC provides the most protection (e.g. against new unknown variations and also against full man-in-the-middle attacks), this is a good long term choice.

The solutions covered in this document hope to cover all of the variations in the recent Kaminsky-style attacks. However, it seems likely that other variations besides the ones described in this document are going to be discovered. For that reason a number of generic protections are included, chief amongst those is the use of extra entropy.

Since this document focuses on Unbound it is worth noting that although current versions implement these mitigations, they are not all turned on by default. Unbound should support the mitigations considered 'best' by the community. This means without weird, ill-considered, mitigations of its own. Hence this document.

It is assumed the reader is aware of, and implementing, the forgery-resilience [RFC5452] (Hubert, A. and R. van Mook, “Measures for Making DNS More Resilient against Forged Answers,” January 2009.) recommendations.

In Section 2 the criteria are listed. In Section 3 the various measures that can be used to mitigate threats are described. Section 4 enumerates Kaminsky-style attack variations, and shows what measures provide protection against each one of them. Section 5 discusses consequences caused by the mitigations.

TOC

2. Criteria

The first and foremost criterium is that these are resolver side solutions, thus only the resolver needs to be redeployed, or the software updated, for this to work. The reason behind this is that a short term deployment is possible. The idea is to provide some (partial) protection on the short term. On the long term it is possible to redeploy both authority and recursors, and the solution space is greatly increased (e.g. options range from EDNS PING, using TCP or SCTP, to DNSSEC deployment).
Many solutions in this document could also be used in stub resolvers. Stub resolvers are not mentioned specifically further on, the main focus is on the caching recursive server.
The solutions have to follow the DNS protocol.
The solutions have to be non disruptive, and non anti-social. Specifically, they must not put the costs of the solution with 3rd parties. For example, large scale fallback to TCP both uses a limited resource (TCP connections to authority servers), and disrupts deployment behind many middle boxes.
Solutions without an 'attack mode' are preferred. An 'attack mode' is a different state of behaviour that the resolver enters into after something anomalous is detected. It may be for only a subset of operations or only a limited time. One reason to avoid such modal design is that paranoia dictates that maximal protection should always be used. A second reason is that if a protection measure cannot be used always, it is likely to be disruptive (see above). Such an 'attack mode' complicates implementation, testing and especially security analysis.

---

3. Mitigations

TOC

Below, the resolver side mitigations are described.

3.1. Add Entropy

TOC

The mitigations in this section increase the transaction entropy above the 16 bits in the ID number. This is pretty close to the forgery-resilience [RFC5452] (Hubert, A. and R. van Mook, “Measures for Making DNS More Resilient against Forged Answers,” January 2009.) text, differences are in the rtt banding text and 0x20 consideration.

*port randomisation

  As many as possible, using only 1000 or 2000 ports (as some commercial DNS products do) is not enough. A range of 59000 port numbers (15.8 bits) can be usefully achieved. This causes operational problems (NAT boxes using predictable port numbers), portability problems (bugs, features not available), and volume problems (using port number uses limited resource).

*0x20.

Breaks queries to some authorities, but more than 99.9% works. It is like a proposal that needs authority server deployment where the authority servers are already deployed to a large extent. [I-D.vixie-dnsext-dns0x20] (Vixie, P. and D. Dagon, "Use of Bit 0x20 in DNS Labels to Improve Transaction Identity," March 2008.).

*rtt banding

RTT banding refers to the method of picking a random nameserver for the query out of the set of nameservers that are within a RTT band (say at most 200 msec slower) from the fastest nameserver.

New attack opportunities can be created by sending a new fake question to be resolved by the resolver. Therefore the actual size of the roundtrip time window is not as important as the additional entropy gained by selecting randomly from a set of servers.

*IPv4 - IPv6

When both IPv4 and IPv6 are available, the protocol can be chosen randomly together with rtt banding to provide more entropy.

*source address randomisation

If the resolver has multiple public IP addresses these can be used to randomise with.

If all the above entropy settings are in use, it is estimated that Unbound can provide about 44 bits of entropy (16 ID, 15.8 port bits, about 8 0x20 bits, about 2 rtt banding + protocol bits and about 2 source address bits). Without user configuration or queries amenable to 0x20, 34 bits of entropy are likely, or even 18 if a NAT box kills the port randomisation. Entropy thus provides only limited protection.

3.2. Use Care with the Cache

TOC

*rfc2181 adherence

This means that RRsets are ranked in trustworthiness depending on whether they come from the answer section, or from another part of the message. The authoritative answers are preferred. [RFC2181] (Elz, R. and R. Bush, "Clarifications to the DNS Specification," July 1997.)

In addition, do not give data obtained from authority or additional sections in answer sections to clients.

*CNAME chain.

Only use first entry in answer section. Perform new lookups for remainder.

*DNAME chain.

Only use the first entry DNAME and its synthesized CNAME from the answer section. Perform new lookups for remainder.

*no DNAME from cache

Do not pick a DNAME RR out of the cache for a query for which that DNAME RR was not returned. Thus, a DNAME is only used for query names for which answers have been received from the authority server.

When the DNAME is signed with DNSSEC, it is allowed to synthesize new CNAMEs from it to answer new queries with it. This is because the zone owner whose zone is redirected is signing away his own zone.

3.3. Obtain Authoritative Data

TOC

*Authority query for NS after referral

The idea is to obtain authoritative data for the NS RRset instead of using data tacked along on another message. Care must be taken to avoid DoSing parent nameservers, and not break resolution in common cases where the NS RRsets in parent and child differ.

On a referral, the data from the referral may be used to continue answering the current query, but it is not stored in the cache. If the question equals the referred zone name and has qtype NS, then the NS RRset from the referral does get stored in the cache.

If the question is not that already, a new lookup is performed for the referred zone name with qtype NS. The results from that lookup are cached normally. The lookup has to start at a parent of the referred zone, so that a new referral is obtained.

The upshot is that RFC2181 adherence pins the NS RRset data in the cache because it is seen in the answer section, and tacked on data from other messages is ignored until the TTL expires. It should be noted that most infrastructure TTLs for NS records are very large.

It does not break existing disjoint RRsets, or servers that do not answer for qtype NS at all, or servers that are offline, because the referral is cached when making the qtype NS query. This is why the qtype NS query has to be made in such a way that it elicits a fresh referral from the parent server. This gives a once per TTL opportunity for spoofing the referral.

The NS RRset answered from the child side of the zone cut overrides the NS RRset picked up from the referral. This causes the same data to be used as today, where the authority section NS set sent along by the child server overrides the NS set seen from

the referral.

Additional queries are sent for this solution. This increases resolver and authority server load and bandwith usage.

*Authority queries for nameserver addresses, A and AAAA.

Same idea, like NS query above. You ask for A or AAAA records directly at the authoritative server. It is not necessary to elicit the referral again, the query can be directed at the best server.

Additional queries are sent for this solution. This increases resolver and authority server load and bandwith usage.

A bonus when using the above methods to obtain authoritative data is that when using DNSSEC, the data can be validated, and thus spoofed infrastructure data can be detected and handled appropriately. This protects DNSSEC, where the referral contains unsigned NS, A and AAAA records from spoofed infrastructure data. Of course, DNSSEC is designed to protect end-user data anyway, whether or not the referral data was poisoned. It simply adds the opportunity to add another layer of defense.

3.4. Detection

TOC

*trouble counter

This is a simple detection method. It counts all packets that were not asked for. The only thing noted about the packet is that it is a query reply (QR bit) and was not asked for.

This may show false positives due to UDP packet duplicates, delayed responses (delayed for longer than the implementation cares to keep track of what it asks for). The idea is that false positives are probably a low amount. Conversely, some unasked for packets may not be noticed because the implementation may not be listening to particular ports, or whatever implementation choices.

When a particular threshold is met, the cache is wiped clean.

The threshold is set so that denial of service does not become all that much easier, and that false positives do not (often) result in cache wipes. A threshold in the range of 10 million is proposed. This many packets itself is already a sizable denial of service attack, and also, the amount of data sent gets close to the cache size of the resolver to keep amplification towards the authority servers low.

Since this mitigation is meant to protect against hitherto unknown variations, it does not help to examine the packets any further than the QR bit (and the fact that they were not used for regular processing).

The result of this is that the probability that there is a poisoned item present in the cache is capped at some maximum. The exact value depends on the entropy per message and the threshold.

---

4. Variants to Protect against

TOC

In the descriptions below a short title is given to quickly summarize the exploit. The query 'q:' is what the attacker sends as fake question to the resolver to answer. The answer, authority 'auth:' and additional 'add:' sections list the content that the spoofer provides. The mitigation strategy, and sometimes discussion, is provided in the 'protected:' line.
The real target is example.com or www.example.com or ns1.example.com, which is the real nameserver for example.com here. The domain evil.example.net is under control of the attacker and 192.0.2.66(evil) is an IP address under control of the attacker. The label 'bad123' is used in place of a label that the attacker varies every attempt to obtain new spoofing windows.

```
Glue with new DNS server
q: bad123.example.com.
answer: bad123.example.com. A whatever
auth: example.com. NS evil.example.com.
add: evil.example.com. A 192.0.2.66(evil)
protected: 2181 adherence plus NS record pinned by NS query.
Also name error or no data answers could be used, instead of
this answer section.

Glue for DNS server
q: bad123.example.com.
answer: bad123.example.com. A whatever
auth: example.com. NS ns1.example.com. (normal entry)
add: ns1.example.com. A 192.0.2.66(evil)
protected: 2181 adherence plus NS record pinned by NS query,
plus A record pinned by glue query.
Also name error or no data answers could be used, instead of
this answer section.

Glue for Web server
q: bad123.example.com.
answer: bad123.example.com. A whatever
auth: example.com. NS www.example.com.
add: www.example.com. A 192.0.2.66(evil)
protected: 2181 adherence plus NS record pinned by NS query.

Glue smaller
q: bad123.example.com.
answer: bad123.example.com. A 192.0.2.66(evil)
auth: example.com. NS bad123.example.com.
protected: 2181 adherence plus NS record pinned by NS query.

NS change
q: bad123.example.com.
answer: bad123.example.com. A whatever
auth: example.com. NS evil.example.net.
protected: 2181 adherence plus NS record pinned by NS query.

NS server migration
q: bad123.example.com.
answer: bad123.example.com. A whatever
auth: example.com. NS ns1.example.com. (normal entry)
auth: example.com. NS ns2.example.com.evil.example.net.
      (evil, looks like typo in server migration)
protected: 2181 adherence plus NS record pinned by NS query.

CNAME
q: bad123.example.com.
answer: bad123.example.com. CNAME www.example.com.
answer: www.example.com. A 192.0.2.66(evil)
protected: CNAME chain cutoff.

DNAME one message
q: www.bad123.example.com.
answer: bad123.example.com. DNAME example.com.
answer: www.bad123.example.com. CNAME www.example.com.
answer: www.example.com. A 192.0.2.66(evil)
```

```
protected: DNAME chain cutoff.

DNAME whole zone
q: bad123.example.com.
answer: example.com. DNAME evil.example.net.
answer: bad123.example.com. CNAME bad123.evil.example.net.
answer: bad123.evil.example.net. A whatever
protected: no DNAME from cache.

New Delegation - rigged
q: bad123.www.example.com.
answer: (empty)
auth: www.example.com. NS www.example.com.
add: www.example.com. A 192.0.2.66(evil)
protected: the NS queries that ask referral confirmation
together with glue queries.

New Delegation - looks normal
q: bad123.www.example.com.
answer: (empty)
auth: www.example.com. NS ns1.evil.example.net.
auth: www.example.com. NS ns2.evil.example.net.
protected: the NS queries that ask referral confirmation
together with glue queries.

New Delegation - for glue
q: bad123.example.com.
answer: (empty)
auth: bad123.example.com. NS ns1.example.com.
additional:  ns1.example.com. A 192.0.2.66(evil)
protected: rfc2181 adherence.

Another hitherto unknown variation
These are a lot of variations and it is very likely that other
people can come up with better, different ideas.
protected: by entropy measures, by the count-and-wipe measure.
Long term solutions (PING, TCP, DNSSEC) also aim to protect
against these much more thoroughly.
```

---

5. Security Considerations

TOC

All of the mitigations aim to provide more security. But, several of these mitigations have adverse effects on performance and bandwith. The CNAME, DNAME, NS and nameserver address mitigations all require that additional lookups be performed. The CNAME and DNAME target lookups cause the answer to the client to be delayed. The NS set and nameserver address lookups cause a higher load on both authority and resolver servers.

The detection mechanism is susceptible to denial of service attacks. A small, calculated, amount of additional DoS leverage is provided. This changes some spoof attacks into a denial of service.

The NS set and nameserver address lookups cause the NS, A and AAAA RRsets to be pinned in the cache until the TTL expires. This provides cache overwriting protection, but at the cost of not picking up updates

to these RRsets in the course of normal resolution. Changes to these RRsets are then no longer seen on the next query, but only after the TTL times out. This adversely affects the coherency of the DNS server infrastructure, as it becomes more likely that resolvers operate using out of date nameserver data.

6. IANA Considerations

TOC

None.

7. Acknowledgments

TOC

Thanks to Nicholas Weaver (ICSI Berkeley) and Olaf Kolkman (NLnet Labs).

8. Informative References

TOC

[I-D.vixie-dnsext-dns0x20]	Vixie, P. and D. Dagon, "Use of Bit 0x20 in DNS Labels to Improve Transaction Identity," draft-vixie-dnsext-dns0x20-00 (work in progress), March 2008 (TXT).
[RFC2181]	Elz, R. and R. Bush, "Clarifications to the DNS Specification," RFC 2181, July 1997 (TXT, HTML, XML).
[RFC5452]	Hubert, A. and R. van Mook, "Measures for Making DNS More Resilient against Forged Answers," RFC 5452, January 2009 (TXT).

Author's Address

TOC

	Wouter Wijngaards
	NLnet Labs
	Science Park 140
	Amsterdam 1098 XG
	The Netherlands
Phone:	+31-20-888-4551
EMail:	wouter@nlnetlabs.nl