The ISP Column
A monthly column on things Internet

February 2010

George Michaelson
Patrik Wallström
Roy Arends
Geoff Huston

Roll Over and Die?

In this month's column I have the pleasure of being joined by George Michaelson, Patrik Wallström and Roy Arends to present some critical results following recent investigations on the behaviour of DNS resolvers with DNSSEC. It's a little longer than usual, but I trust that its well worth the read

-- Geoff

It is considered good security practice to treat cryptographic keys with a healthy level of respect. The conventional wisdom appears to be that the more material you sign with a given private key the more clues you are leaving behind that could enable some form of effective key guessing. As RFC4641 states: "the longer a key is in use, the greater the probability that it will have been compromised through carelessness, accident, espionage, or cryptanalysis." Even though the risk is considered slight if you have chosen to use a decent key length, RFC 4641 recommends, as good operational practice, that you should "roll" your key at regular intervals. Evidently it's a popular view that fresh keys are better keys!

The standard practice for a "staged" key rollover is to generate a new key pair, and then have the two public keys co-exist at the publication point for a period of time, allowing relying parties, or clients, some period of time to pick up the new public key part. Where possible during this period, signing is performed twice, once with each key, so that the validation test can be performed using either key. After an appropriate interval of parallel operation the old key pair can be deprecated and the new key can be used for signing.

This practice of staged rollover as part of key management is used in X.509 certificates, and is also used in signing the DNS, using DNSSEC. A zone operator who wants to roll the DNSSEC key value would provide notice of a pending key change, publish the public key part of a new key pair, and then use the new and old private keys in parallel for a period. On the face of it, this process sounds quite straightforward.

What could possibly go wrong?

Detecting the Problem

As we are constantly reminded, the Internet can be a very hostile place, and public services are placed under constant pressure from a stream of probe traffic, attempting to exercise any one of a number of vulnerabilities that may be present at the server. In addition, there is the threat of

denial of service (DoS) attacks, where a service is subjected to an abnormally high traffic load that attempts to saturate and take it down.

The traffic signature in Figure 1 is a typical signature of an attempted DOS attack on a server, where the server is subjected to a sudden surge in queries. In this case the query log is from a server that is a secondary name server for a number of subdomains of the *in-addr.arpa*. zone, and the traffic surge shown here commenced on 16 December 2009. The traffic pattern shifted from a steady state of some 12Mbps to a new state of more than 20Mbps, peaking at 30Mbps.

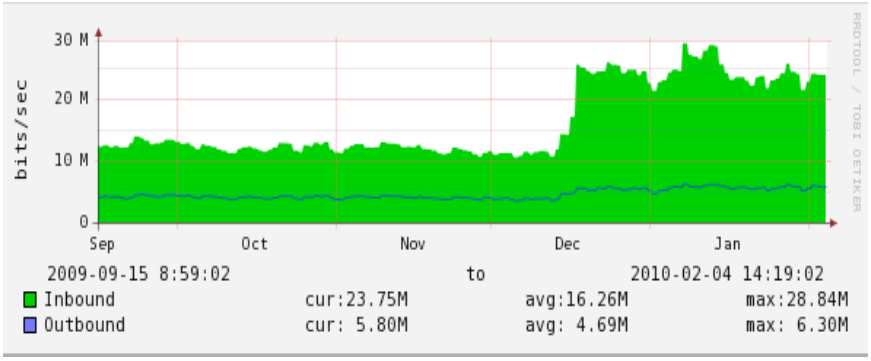

Figure 1 - Traffic Load in-addr.arpa server, December 2009 (provided by George Michaelson)

As the traffic shown in Figure 1 is traffic passed to and from a name server, the next step is to examine the DNS traffic on the name server, and in particular look at the type and rate of DNS queries that are being sent to the name server. The bulk of the additional query load is for DNSKEY Resource Records (RRs), as shown in Figure 2.

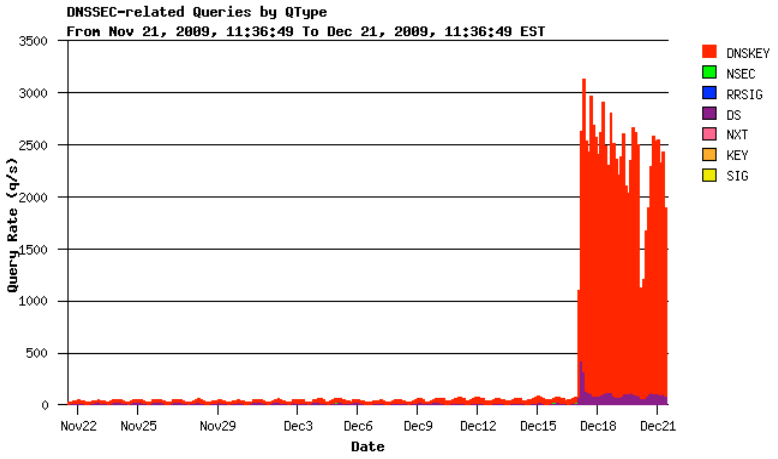

Figure 2 - Query rate for in-addr.arpa server, December 2009 (provided by George Michaelson)

As this is a DNSSEC signed zone then this query set will cause the server to send the DNSKEY RR and the related RRSIG RR in response to each query, which is a 1,188 byte response in this case. At a peak rate of some 3,000 DNS queries per second, that's a peak rate in excess of 35 Mbps of generated traffic in the DNS responses.

There are a number of possibilities as to what is going on here:

- This could be a DoS attack directed at the server, attempting to saturate the server by flooding it with short queries that generate a large response.
- This could be a DNS reflection DoS attack where the attacker is placing the address of the intended victim or victims in the source address of the DNS queries and attempting to overwhelm the victim with this DNS traffic.

While its good to be suspicious, its also useful to remember the old adage that one should be careful not to ascribe to malice what could equally be explained by incompetence, so a number of other explanations should also be listed here:

- This could be a DNS resolver problem, where the resolver is not correctly caching the response, and some local event is triggering repeated queries.
- This could be a bug in an application, where the application has managed to wedge itself in a state of rapid fire queries for DNSKEY RRs.

While it is not unusual for such anomalies to persist for days, it is more typical to see such events dissipate in the ensuing hours, and the tail of the query log in Figure 1 is a typical decay pattern. However, in this case this has not occurred, and Figure 3 shows the query rate for this server over the ensuing month. Not only does the query load remain over 2,000 queries per second over much of the period, it peaks at over 3,000 queries per second for 7 days in January 2010.

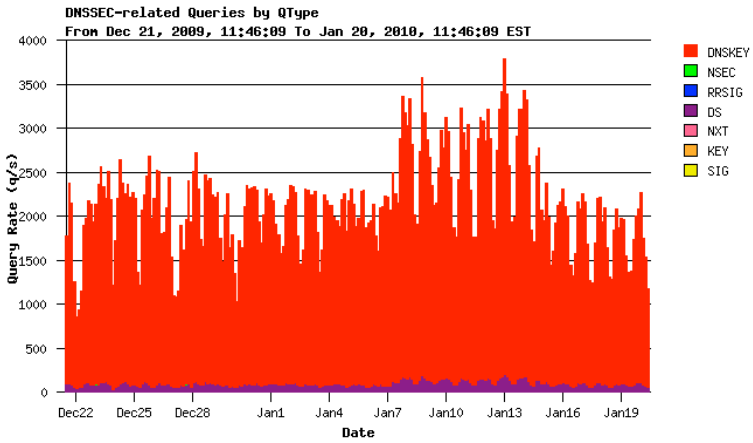

Figure 3 - Query rate for in-addr.arpa server, Dec 2009 / Jan 2010 (provided by George Michaelson)

From this data it appears that a query rate of a peak of some 2,000 queries per second is now part of the sustained "normal" load pattern, it is also possible that:

- This could be an instance of a new application being deployed that makes intensive use of validation of DNS responses.

The next step is to look at some of these queries in a little further detail. It makes some sense to look at the distribution of query source addresses to see if this load can be attributed to a small number of resolvers that are making a large number of queries, or if the load is spread across a much larger set of resolvers. The server in question typically sees of the order 500,000 to 1,000,000 distinct query sources per day. Closer inspection of the query logs indicates that the additional load is coming from a relatively small subset of resolvers of the order 1,000 distinct source addresses, with around 100 'heavy hitters', and the following sequence of queries from one such resolver is typical of the load being imposed on the server:

The following is an example of a dnscap (https://www.dns-oarc.net/tools/dnscap) *packet dis-assembly. Dnscap has a DNS parser which is able to present the binary DNS packet in readable form, aiding analysis of this situation. The first packet exchanges are shown in full, the remainder have been elided.*

Client requests the DS records for the 211.89.in-addr.arpa zone

```
[72] 03:06:46.755545
        [212.126.213.158].38868 [202.12.29.59].53
        dns QUERY,NOERROR,11038,cd
        1 211.89.in-addr.arpa,IN,TYPE43 0 0
        1 .,CLASS4096,OPT,32768,[0]
```

Reply says 'no such delegation', and sends RRSIG and NSEC from the parent zone, and surrounding records.

```
[577] 03:06:46.755634 [#57 eth2 0] [202.12.29.59].53 [212.126.213.158].38868  dns
QUERY,NOERROR,11038,qr|aa \
        1 211.89.in-addr.arpa,IN,TYPE43 0 \
        4 89.in-addr.arpa,IN,SOA,7200,ns-pri.ripe.net,
          dns-help.ripe.net,2010012011,3600,7200,1209600,7200 \
        89.in-addr.arpa,IN,TYPE46,172800,[185] \
```

```
        210.89.in-addr.arpa,IN,TYPE47,7200,[31] \
        210.89.in-addr.arpa,IN,TYPE46,7200,[185] 1 .,CLASS4096,OPT,0,[0]
```

Client requests DNS Key from the parent zone

```
[72] 2010-01-20 03:06:47.817759 [#60 eth2 0] \
        [212.126.213.158].24482 [202.12.29.59].53  \
        dns QUERY,NOERROR,226,cd \
        1 89.in-addr.arpa,IN,TYPE48 0 0 \
        1 .,CLASS4096,OPT,32768,[0]
```

Server sends DNSKEY and RRSIG set for the parent zone

```
[1188] 2010-01-20 03:06:47.817835 [#61 eth2 0] \
        [202.12.29.59].53 [212.126.213.158].24482  \
        dns QUERY,NOERROR,226,qr|aa \
        1 89.in-addr.arpa,IN,TYPE48 \
        5 89.in-addr.arpa,IN,TYPE48,3600,[264] \
        89.in-addr.arpa,IN,TYPE48,3600,[158] \
        89.in-addr.arpa,IN,TYPE48,3600,[158] \
        89.in-addr.arpa,IN,TYPE46,3600,[291] \
        89.in-addr.arpa,IN,TYPE46,3600,[185] 0 \
        1 .,CLASS4096,OPT,0,[0]
```

Having established an initial query state, and the DNSKEY and signature set over the original request, the client then paradoxically repeatedly re-queries the parent zone DNSKEY state. This has been elided below since the query and response do not differ during this exchange.

Client repeats the request

```
[72] 2010-01-20 03:06:48.892744 [#66 eth2 0]  [212.126.213.158].59720
[202.12.29.59].53
```

Server repeats the response

```
[1188] 2010-01-20 03:06:48.892825 [#67 eth2 0][202.12.29.59].53
[212.126.213.158].59720
```

Repetition #2

```
[72] 2010-01-20 03:06:49.963730 [#70 eth2 0]  [212.126.213.158].41028
[202.12.29.59].53
[1188] 2010-01-20 03:06:49.963821 [#71 eth2 0][202.12.29.59].53
[212.126.213.158].41028
```

Repetition #3

```
[72] 2010-01-20 03:06:51.024725 [#74 eth2 0]  [212.126.213.158].32988
[202.12.29.59].53
[1188] 2010-01-20 03:06:51.024820 [#75 eth2 0][202.12.29.59].53
[212.126.213.158].32988
```

Repetition #4

```
[72] 2010-01-20 03:06:52.091457 [#78 eth2 0]  [212.126.213.158].35975
[202.12.29.59].53
[1188] 2010-01-20 03:06:52.091544 [#79 eth2 0][202.12.29.59].53
[212.126.213.158].35975
```

Repetition #5

```
[72] 2010-01-20 03:06:53.451761 [#84 eth2 0]   [212.126.213.158].6458
[202.12.29.59].53
[1188] 2010-01-20 03:06:53.451852 [#85 eth2 0] [202.12.29.59].53
[212.126.213.158].6458
```

Repetition #6

```
[72] 2010-01-20 03:06:54.556688 [#88 eth2 0]  [212.126.213.158].45030
[202.12.29.59].53
[1188] 2010-01-20 03:06:54.556779 [#89 eth2 0][202.12.29.59].53
[212.126.213.158].45030
```

Repetition #7

```
[72] 2010-01-20 03:06:55.335413 [#92 eth2 0]  [212.126.213.158].57584
[202.12.29.59].53
[1188] 2010-01-20 03:06:55.335556 [#93 eth2 0 [202.12.29.59].53
[212.126.213.158].57584
```

Repetition #8

```
[72] 2010-01-20 03:06:56.463737 [#96 eth2 0]  [212.126.213.158].37422
[202.12.29.59].53
[1188] 2010-01-20 03:06:56.463818 [#97 eth2 0][202.12.29.59].53
[212.126.213.158].37422
```

Figure 4 - DNS Query Sequence packet Capture (provided by George Michaelson)

Within a 10 second interval the client has made 11 queries. The first two queries are conventional DNS queries. The initial query is for the DS records for *211.89.in-addr.arpa*, with DNSSEC signature checking. The response is a "no such delegation" (NXDOMAIN) response, with the NSEC and RRSIG records from the parent zone attached. The client then wants to validate this signed response, and requests the DNSKEY from the parent zone, which the server then provides, as it is also authoritative for the parent zone. The client then enters into a tight loop where it repeats the request a further 8 times in the ensuing 8 seconds.

If this additional query load had appeared at the server over an extended period of time, then it would be possible to ascribe this to a faulty implementation of a DNS resolver, or a faulty client application. However, the sudden onset of the additional load on 16 December 2009 tends to suggest that something else is happening. The most likely explanation is that there was some external "trigger" event that exacerbated a latent behavioural bug in a set of DNS resolver clients. And the most likely external trigger event is in a change of the contents of the zones being served.

So we can now refine our set of possible causes to concentrate consideration on the possibility that:

- Something changed in the zones being served by this secondary server that triggered a pathological query response from a set of resolvers.

And indeed the zones' contents did change on 16 December, with a key change being implemented on that day when the old key was removed from the zone file and the zone was exclusively signed by the new key.

Key Management and DNSSEC

The RIPE NCC publish their DNSSEC keys at https://www.ripe.net/projects/disi/keys/, and their key management procedures at https://www.ripe.net/rs/reverse/dnssec/key-maintenance-procedure.html. The keys are rolled at six monthly intervals and, as shown in Figure 5, the most recent key rollover occurred on 16 December 2009.

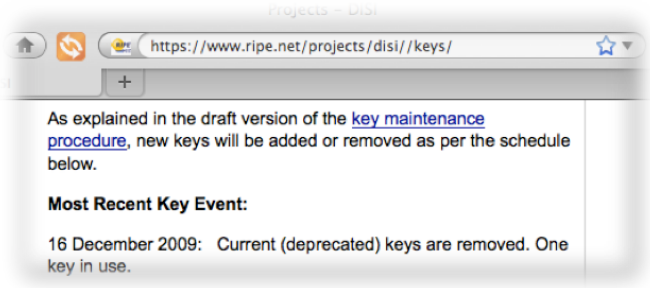

Figure 5 - Notice of Key Rollover in in-addr.arpa domains - RIPE NCC

This key rollover function should be a routine procedure, without any unintended side effects. Resolvers who are using DNSSEC should take care that they refresh their local cache of zone keys in synchronization with the published schedule, and ensure that they load a copy of the new key within the three month period when the two keys coexist. In this way when the old key is deprecated, responses from the zone's servers can be locally validated using the new key.

It should be noted that this manual procedure of key refresh of the keys associated with zones at levels below that of the root is more of an artifact of the current situation of piecemeal deployment of DNSSEC, and not an intentional part of the long term DNSSEC production environment. The long term intention is that of broad DNSSEC deployment, which would allow all DNSSEC keys to be validated via a single trust point at the root zone key.

For example, if a client is attempting to validate a response relating to a query for 211.89.in-addr.arpa and, hypothetically, every zone in the DNS was DNSSEC signed, then a key in *89.in-addr.arpa* has been signed over by the zone key for *in-addr.arpa*, which, in turn, was signed over by the zone key for *.arpa*, which in turn leads the client back to the signature over the root zone. In such a scenario each DNSSEC resolver client would only need to synchronize its local cache against the root keys and all other key rollovers would be picked up automatically by the client.

But this is not the case at present, and because *in-addr.arpa* is not yet a DNSSEC-signed zone then a client who wants to validate a response relating to *211.89.in-addr.arpa* will need to ensure that it has a local copy of the key for the *89.in-addr.arpa* zone. The associated implication of this piecemeal deployment scenario is that whenever a key rollover event occurs in one of these zones where the local resolver holds a copy of the key, then the local resolver needs to ensure that it obtains a copy of the new key before the key rollover operation at the zone has completed.

The question here is why did this particular key rollover for the signed zone cause the traffic load at the server to spike? And why is the elevated query rate sustained for weeks after the key rollover event? The key had changed six months earlier and yet the query load in early December was extremely low. And given that there is a diurnal cycle in query loads, is there some form of user behaviour that is driving this elevated query rate?

DNSSEC DNS Resolver Behaviour with Outdated Trust Keys

It's possible to formulate a theory as to what is going on here from the trace of Figure 4.

If could be that the DNS resolver client is using a local trust anchor that has been manually downloaded from the RIPE NCC earlier, prior to the most recent key rollover. When the key rollover occurred in December 2009 then the client could no longer validate the response with its locally stored trust anchor. Upon detecting an invalid signature in the response, the client reacted as if there was a "man-in-the middle" injection attempt, and immediately repeated the request in a effort to circumvent the supposed attack by rapidly repeating the query. If this were an instance of a man-in-the-middle injection attack this would be a plausible response, as there is the hope that the query will still reach the authoritative server and the client will receive a genuine response that can be locally validated. Given that this form of man-in-the-middle attack is directed at the client, the short burst of repeat queries from an individual client would be an isolated small scale event and should not trouble the server. However it is not a man-in-the-middle attack. This is a local failure to update the local trust anchor keys, and the DNS resolver is reacting as if it were attempting to circumvent some form of attack that is attempting to feed it invalid DNS information though repeated queries.

If one such client failed to update its local trusted key set, then the imposed server load on key rollover would be slight. However, if a larger number of clients were to be caught out in this

manner then the server load signature would look a lot like Figure 2. The additional load imposed on the server comes from the size of the DNSKEY and RRSIG responses, which, as shown in Figure 4, are 1,188 bytes per response.

So far we've been concentrating attention on the in-addr.arpa zone, where the operational data was originally gathered. However, it appears that this could happen to any DNSSEC signed domain where the zone's keys are published to allow clients to manually load them as trust points, and where the keys are rolled on a regular basis. Figure 6 shows a local trace from the perspective of a DNS resolver client of a DNSKEY exchange with the .se server following a key rollover by .se. This is presented using *tcpdump* (http://www.tcpdump.org), a more compressed representation than the *dnscap* viewed earlier, but substantively the same information is being presented.

```
06:50:18.045007 IP 94.254.84.99.6197 > 130.239.5.114.53: 34306% [1au] DNSKEY? se. (31)
06:50:18.059296 IP 94.254.84.99.7496 > 199.254.63.1.53: 8090% [1au] DNSKEY? se. (31)
06:50:18.088664 IP 94.254.84.99.10736 > 81.228.10.57.53: 62708% [1au] DNSKEY? se. (31)
06:50:19.075008 IP 94.254.84.99.18234 > 81.228.8.16.53: 8774% [1au] DNSKEY? se. (31)
06:50:19.078576 IP 94.254.84.99.57591 > 81.228.8.16.53: P 0:33(33) ack 1 win 78
        <nop,nop,timestamp 428416112 3213455965>58695% [1au] DNSKEY? se. (31)
06:50:19.081633 IP 94.254.84.99.50469 > 81.228.10.57.53: 36119% [1au] DNSKEY? se. (31)
06:50:20.685057 IP 94.254.84.99.24744 > 81.228.10.57.53: 60621% [1au] DNSKEY? se. (31)
06:50:20.711385 IP 94.254.84.99.42081 > 81.228.10.57.53: P 0:33(33) ack 1 win 78
        <nop,nop,timestamp 428416275 3215995958>15873% [1au] DNSKEY? se.(31)
06:50:20.727943 IP 94.254.84.99.18180 > 194.146.106.22.53: 639% [1au] DNSKEY? se. (31)
06:50:20.729360 IP 94.254.84.99.14715 > 130.239.5.114.53: 42673% [1au] DNSKEY? se.
(31)
06:50:20.740681 IP 94.254.84.99.8412 > 192.36.135.107.53: 43484% [1au] DNSKEY? se.
(31)
06:50:20.748103 IP 94.254.84.99.15091 > 199.7.49.30.53: 48966% [1au] DNSKEY? se. (31)
06:50:20.771257 IP 94.254.84.99.26304 > 199.254.63.1.53: 5749% [1au] DNSKEY? se. (31)
06:50:20.800054 IP 94.254.84.99.43607 > 192.36.144.107.53: 42701% [1au] DNSKEY?
se.(31)
06:50:20.801602 IP 94.254.84.99.38911 > 192.71.53.53.53: 48332% [1au] DNSKEY? se. (31)
06:50:20.803462 IP 94.254.84.99.60254 > 192.36.133.107.53: 644% [1au] DNSKEY? se. (31)
06:50:20.815856 IP 94.254.84.99.31948 > 81.228.8.16.53: 17008% [1au] DNSKEY? se. (31)
06:50:21.625015 IP 94.254.84.99.19712 > 81.228.10.57.53: 19280% [1au] DNSKEY? se. (31)
06:50:22.542092 IP 94.254.84.99.41822 > 81.228.8.16.53: 22224% [1au] DNSKEY? se. (31)
06:50:23.515012 IP 94.254.84.99.55978 > 81.228.10.57.53: 29847% [1au] DNSKEY? se. (31)
```

Figure 6 - DNSKEY queries on .se - (provided by Patrik Wallström)

Over an interval of 5.5 seconds, the client has made 17 UDP requests and 2 TCP requests to the .se servers for the DNSKEY records for .se.

Perhaps this re-query behaviour is quite widespread for DNSSEC-enabled resolvers. If that's the case then we can look back in the *in-addr.arpa* secondary logs and examine the logs at the time of the previous scheduled key rollover in June 2009, as shown in Figure 7.

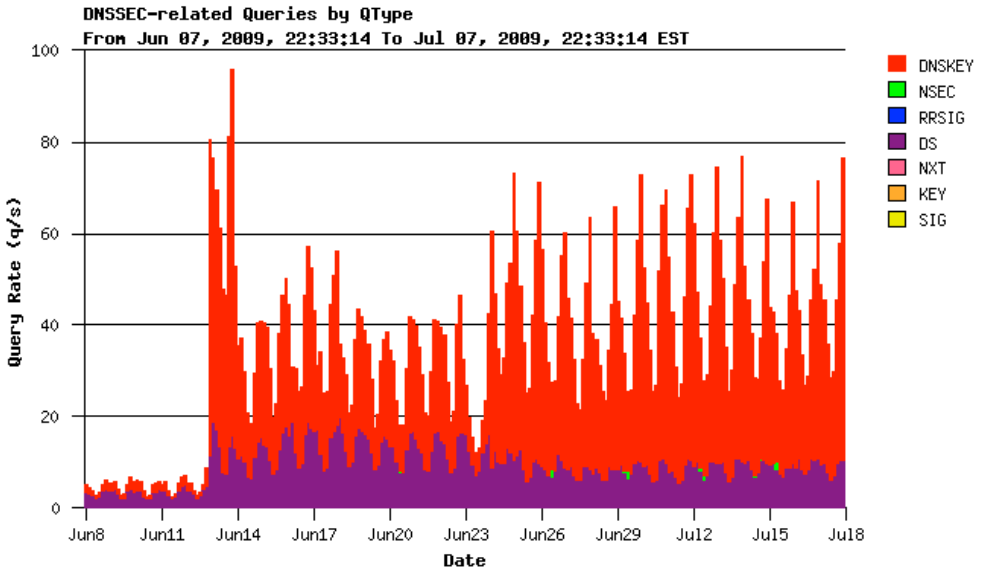

Figure 7 - DNS Query Load for in-addr-arpa secondary, June 2009 (provided by George Michaelson)

The spike in traffic at key rollover time in June 2009 is certainly there, but it rose from a query rate of less than 10 queries a second to a diurnal pattern that peaks at 60 queries per second. At the time of the next key rollover in December 2009, the pre-rollover query rate was a diurnal pattern that peaked at 60 - 70 queries per second. Perhaps the increased query rate never actually went away, in which case it is possible that the incremental query load seen in December 2009 will not dissipate either. The change in behaviour could be attributed more simply to the increased experimentation and trials of local trust anchors to complement the deployment of DNSSEC aware resolvers over the past six months, complementing the technical awareness campaign over the forthcoming DNSSEC signing of the root zone that also occurred in this period.

This looks to have been happening in *in-addr.arpa* for some time. The query log for June 2008 shows a similar spike to 60 - 80 queries per second at the time of key rollover (Figure 8).

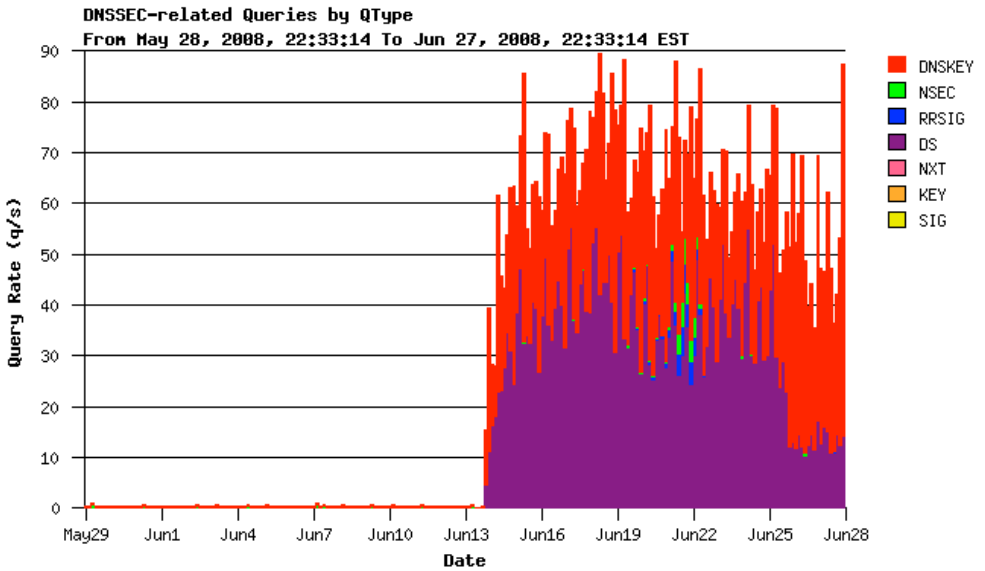

Figure 8 - DNS Query Load for in-addr-arpa secondary, June 2008 (provided by George Michaelson)

This load was not sustained and it dropped back to a quiescent level in two distinct steps, on 25 June 25 and 10 July 10, as shown in Figure 9.

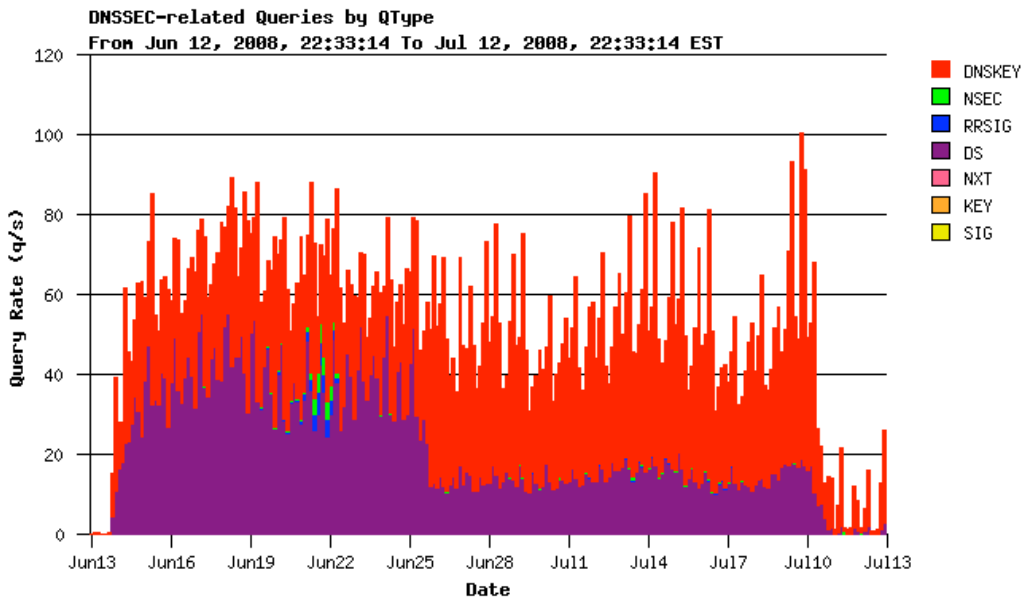

Figure 9 - DNS Query Load for in-addr-arpa secondary, June/July 2008 (provided by George Michaelson)

There is the strong implication from this data that when an individual resolver is operating from outdated local trust keys it is capable of imposing a load of 20 - 40 queries per second on the in-addr.arpa secondary servers.

Its likely that one possible cause for this situation was in the way in which some DNSSEC distributions are packaged with operating systems. The following notice to the RIPE DNS Working Group points to a known issue with a Fedora Linux distribution:

> From: Anand Buddhdev <anandb@ripe.net>
> Date: February 5, 2010 2:23:51 PM GMT+01:00
> To: dns-wg@ripe.net
> Subject: [dns-wg] Outdated RIPE NCC Trust Anchors in Fedora Linux Repositories
>
> We have discovered that recent versions of the Fedora Linux distribution are shipping with a package called "dnssec-conf", which contains the RIPE NCC's DNSSEC trust anchors. This package is installed by default as a dependency of BIND, and it configures BIND to do DNSSEC validation.
>
> Unfortunately, the current version of this package (1.21) is outdated and contains old trust anchors.
>
> On 16 December 2009, we had a key roll-over event, where we removed the old Key-Signing Keys (KSKs). From that time, BIND resolvers running on Fedora Linux distributions could not validate any signed responses in the RIPE NCC's reverse zones.
>
> If you are running Fedora Linux with the standard BIND package, please edit the file "/etc/pki/dnssec-keys//named.dnssec.keys", and comment out all the lines in it containing the directory path "production/reverse". Then restart BIND.
>
> This will stop BIND from using the outdated trust anchors. If you do want to use the RIPE NCC's trust anchors to validate our signed zones, we recommend that you fetch the latest trust anchor file from our website and

> reconfigure BIND to use it instead of the ones distributed in the dnssec-conf package:
>
> https://www.ripe.net/projects/disi/keys/index.html
>
> Please remember to check frequently for updates to our trust anchor file, as we introduce new Key-Signing Keys (KSKs) every 6 months.
>
> Regards,
>
> Anand Buddhdev,
> DNS Services Manager, RIPE NCC

So there appears to be a combination of three factors that are causing this situation:

- The use of pre-packaged DNSSEC distributions that included pre-loaded keys in the distribution.
- The use of regular key rollover procedures by the zone administrator.
- Some implementations of DNS resolvers that react aggressively when there is a key validation failure by performing a rapid sequence of repeat queries, with either a very slow, or in some cases no apparent back-off in query load

This combination of circumstances makes the next scheduled key rollover for in-addr.arpa, scheduled for June 2010, look to be quite an "interesting" event. If there is the same level of increase in use of DNSSEC with manually managed trust keys over this current six month interval as we've seen in the pervious six months, and if there is the same proportion of clients who fail to perform a manual update prior to the next scheduled key rollover event, then the increase in the query load imposed on in-addr.arpa servers at the time of key rollover promises to be truly biblical in volume!

Signing the Root

There is an end in sight for this situation for the sub-zones of in-addr.arpa, and for all other such sub-zones that currently have to resort to various forms of distribution of their keys as putative trust keys for DNS resolver clients. ICANN has announced that on 1 July 2010 a signed root zone for the DNS will be fully deployed. On the possibly optimistic assumption that the .arpa. and .in-addr.arpa. zones will be DNSSEC signed in a similar timeframe, then the situation of escalating loads being imposed on the servers for delegated subdomains of in-addr.arpa at each successive key rollover event will be curtailed. It would then be possible to configure the client with a single trust key, being the public key signing key for the root zone, and allow the client to perform all signature validation without need to manually manage other local trust keys.

There are two potential problems with this.

The first is that for those clients who fail to remove the local trust anchor key set the problem does not go away. For both BIND and UNBOUND, when there are multiple possible chains of trust then the resolver will chose to attempt to validate the shortest chain. As an example, if a client has configured the DNSKEY for, say, *test.example.com* into their local trust anchor key set, and they then subsequently add the DNSKEY for *example.com*, the resolver client will attempt to validate all queries in *text.example.com* and its subzones using the *test.example.com* DNSKEY. A more likely scenario is where an operator has already added local trust anchor keys for, say, *.org* or *.se*. When the root of the DNS is signed then they may also add the keys for the root to their local trust anchor set. If they omit to remove the *.org* and *.se* trust anchor keys, in the belief that this root key value will override the *.org* and *.se* local keys, then they will encounter the same

validation failure behaviour. In such a case, when the local keys for these second level domains become stale, then their resolver will exhibit the same re-query behaviour, even when they maintain a valid local root trust anchor key.

> As a side note, the same behaviour may occur when DNSSEC Lookaside Validation (DLV) is being used. If the zone's key management procedures fall out of tight synchronization with the DLV repository then it is possible to open a window where the old key remains in the DLV repository, but is no longer in the zone file. This can lead to a window of vulnerability where the keys in the DLV repository are unable to validate the signed information in the zone file, which, in turn exercises the same problem with re-query.

The second potential problems lies with the phase-in approach of signing the root. The staged rollout of DNSSEC for the root zone envisages a sequenced deployment of DNSSEC across the root server clusters and throughout this sequence the root will be signed with a "DURZ" (Figure 10)

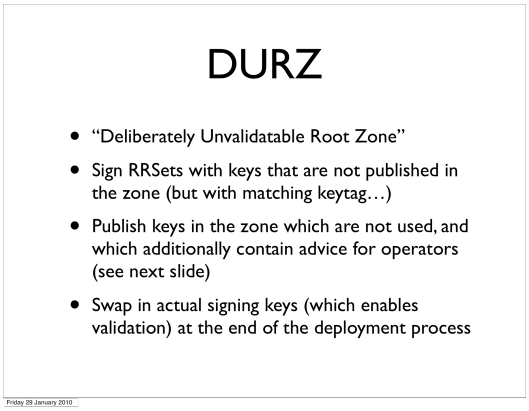

Figure 10 - Root Zone DNSSEC deployment - Joe Abley, ICANN, January 2010

The DNSKEY RR that will be used for this phase of the root zone deployment is an invalid key (Figure 11)

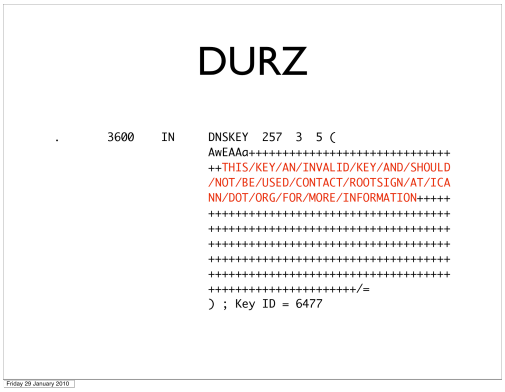

Figure 11 - Root Zone DNSKEY- Joe Abley, ICANN, January 2010

What happens when a client installs this key in their local trust anchor set, and performs a query into the root zone?

As an experiment, this key was installed into an instance of BIND 9.7.0rc2, with a single

upstream root, pointing at the L root, the only instance of the 13 authoritative root servers currently enabled with DNSSEC signed data. Figure 12 shows the *tcpdump* of this experiment.

Starting BIND with the DURZ key configured with root looks like this:

```
02:23:13.903200 IP 94.254.84.99.20845 > 192.36.148.17.53: 39812% [1au] DNSKEY? . (28)
02:23:13.904447 IP 94.254.84.99.29484 > 192.112.36.4.53: 5784% [1au] DNSKEY? . (28)
02:23:14.063617 IP 94.254.84.99.56185 > 202.12.27.33.53: 58470% [1au] DNSKEY? . (28)
02:23:14.096800 IP 94.254.84.99.19540 > 192.33.4.12.53: 63411% [1au] DNSKEY? . (28)
02:23:14.202476 IP 94.254.84.99.23210 > 128.63.2.53.53: 43288% [1au] DNSKEY? . (28)
02:23:14.302964 IP 94.254.84.99.61614 > 193.0.14.129.53: 60641% [1au] DNSKEY? . (28)
02:23:14.443820 IP 94.254.84.99.39117 > 128.8.10.90.53: 52235% [1au] DNSKEY? . (28)
02:23:14.580610 IP 94.254.84.99.1832 > 192.228.79.201.53: 41792% [1au] DNSKEY? . (28)
02:23:14.749730 IP 94.254.84.99.42450 > 192.203.230.10.53: 52903% [1au] DNSKEY? . (28)
02:23:14.934376 IP 94.254.84.99.32392 > 199.7.83.42.53: 48480% [1au] DNSKEY? . (28)
02:23:15.073805 IP 94.254.84.99.18993 > 192.5.5.241.53: 53794% [1au] DNSKEY? . (28)
02:23:15.083405 IP 94.254.84.99.18362 > 192.58.128.30.53: 32638% [1au] DNSKEY? . (28)
02:23:15.536684 IP 94.254.84.99.40824 > 198.41.0.4.53: 63668% [1au] DNSKEY? . (28)

On issuing the first query! (Any query will do!)
02:23:17.237648 IP 94.254.84.99.43118 > 192.36.148.17.53: 20348% [1au] DNSKEY? . (28)
02:23:17.497613 IP 94.254.84.99.26253 > 192.112.36.4.53: 27565% [1au] DNSKEY? . (28)
02:23:17.541230 IP 94.254.84.99.13293 > 128.8.10.90.53: 14401% [1au] DNSKEY? . (28)
02:23:17.677963 IP 94.254.84.99.12985 > 192.58.128.30.53: 21457% [1au] DNSKEY? . (28)
02:23:17.686715 IP 94.254.84.99.47565 > 202.12.27.33.53: 11950% [1au] DNSKEY? . (28)
02:23:17.719576 IP 94.254.84.99.52505 > 193.0.14.129.53: 27749% [1au] DNSKEY? . (28)
02:23:17.744421 IP 94.254.84.99.12667 > 192.203.230.10.53: 10018% [1au] DNSKEY? . (28)
02:23:17.929291 IP 94.254.84.99.4109 > 128.63.2.53.53: 46561% [1au] DNSKEY? . (28)
02:23:18.029820 IP 94.254.84.99.43702 > 192.228.79.201.53: 48403% [1au] DNSKEY? . (28)
02:23:18.200672 IP 94.254.84.99.63829 > 192.33.4.12.53: 10977% [1au] DNSKEY? . (28)
02:23:18.306366 IP 94.254.84.99.7642 > 192.5.5.241.53: 8983% [1au] DNSKEY? . (28)
02:23:18.315797 IP 94.254.84.99.39297 > 199.7.83.42.53: 55836% [1au] DNSKEY? . (28)
02:23:18.455164 IP 94.254.84.99.61774 > 198.41.0.4.53: 47958% [1au] DNSKEY? . (28)
02:24:09.124070 IP 94.254.84.99.40791 > 194.17.45.54.53: 28761% [1au] DNSKEY? iis.se. (35)
02:24:09.150048 IP 94.254.84.99.62256 > 192.36.133.107.53: 61571% [1au] DNSKEY? se. (31)
02:24:09.569264 IP 94.254.84.99.50097 > 192.36.148.17.53: 27683% [1au] DNSKEY? . (28)
02:24:09.570446 IP 94.254.84.99.54285 > 198.41.0.4.53: 45514% [1au] DNSKEY? . (28)
02:24:09.670769 IP 94.254.84.99.9913 > 192.112.36.4.53: 23467% [1au] DNSKEY? . (28)
02:24:09.715313 IP 94.254.84.99.2640 > 192.33.4.12.53: 52147% [1au] DNSKEY? . (28)
02:24:09.817716 IP 94.254.84.99.58291 > 128.8.10.90.53: 7659% [1au] DNSKEY? . (28)
02:24:09.953736 IP 94.254.84.99.38181 > 192.5.5.241.53: 8159% [1au] DNSKEY? . (28)
02:24:09.963126 IP 94.254.84.99.1905 > 199.7.83.42.53: 47251% [1au] DNSKEY? . (28)
02:24:10.106348 IP 94.254.84.99.62103 > 128.63.2.53.53: 10315% [1au] DNSKEY? . (28)
02:24:10.212582 IP 94.254.84.99.37972 > 192.58.128.30.53: 40089% [1au] DNSKEY? . (28)
02:24:10.400276 IP 94.254.84.99.42996 > 192.203.230.10.53: 17397% [1au] DNSKEY? . (28)
02:24:10.581850 IP 94.254.84.99.31677 > 192.228.79.201.53: 1277% [1au] DNSKEY? . (28)
02:24:10.740591 IP 94.254.84.99.49761 > 193.0.14.129.53: 26150% [1au] DNSKEY? . (28)
02:24:11.404268 IP 94.254.84.99.10249 > 202.12.27.33.53: 63604% [1au] DNSKEY? . (28)
02:24:11.458706 IP 94.254.84.99.24564 > 202.12.27.33.53: 204% [1au] DNSKEY? . (28)
02:24:11.607310 IP 94.254.84.99.15168 > 192.36.148.17.53: 10115% [1au] DNSKEY? . (28)
02:24:11.608500 IP 94.254.84.99.40661 > 192.33.4.12.53: 36614% [1au] DNSKEY? . (28)
02:24:11.991944 IP 94.254.84.99.43132 > 192.112.36.4.53: 22922% [1au] DNSKEY? . (28)
02:24:12.035845 IP 94.254.84.99.38528 > 128.8.10.90.53: 903% [1au] DNSKEY? . (28)
02:24:12.172145 IP 94.254.84.99.32505 > 128.63.2.53.53: 12626% [1au] DNSKEY? . (28)
02:24:12.278046 IP 94.254.84.99.11937 > 192.203.230.10.53: 57589% [1au] DNSKEY? . (28)
02:24:12.460078 IP 94.254.84.99.62363 > 192.228.79.201.53: 28558% [1au] DNSKEY? . (28)
02:24:12.623728 IP 94.254.84.99.53880 > 192.5.5.241.53: 43286% [1au] DNSKEY? . (28)
02:24:12.813330 IP 94.254.84.99.13181 > 198.41.0.4.53: 43192% [1au] DNSKEY? . (28)
02:24:12.912998 IP 94.254.84.99.25437 > 192.58.128.30.53: 39893% [1au] DNSKEY? . (28)
02:24:12.922437 IP 94.254.84.99.62132 > 199.7.83.42.53: 60602% [1au] DNSKEY? . (28)
02:24:13.061302 IP 94.254.84.99.38905 > 193.0.14.129.53: 46454% [1au] DNSKEY? . (28)
02:24:14.698210 IP 94.254.84.99.14584 > 192.36.148.17.53: 40183% [1au] DNSKEY? . (28)
02:24:14.699449 IP 94.254.84.99.19093 > 202.12.27.33.53: 39930% [1au] DNSKEY? . (28)
02:24:14.753222 IP 94.254.84.99.53340 > 192.112.36.4.53: 38911% [1au] DNSKEY? . (28)
02:24:15.028191 IP 94.254.84.99.39292 > 193.0.14.129.53: 26615% [1au] DNSKEY? . (28)
02:24:15.196278 IP 94.254.84.99.42742 > 128.8.10.90.53: 25979% [1au] DNSKEY? . (28)
02:24:15.332954 IP 94.254.84.99.36728 > 192.228.79.201.53: 28439% [1au] DNSKEY? . (28)
02:24:15.505941 IP 94.254.84.99.36551 > 192.33.4.12.53: 38868% [1au] DNSKEY? . (28)
02:24:15.608257 IP 94.254.84.99.14098 > 128.63.2.53.53: 23742% [1au] DNSKEY? . (28)
02:24:15.714463 IP 94.254.84.99.63128 > 192.203.230.10.53: 16222% [1au] DNSKEY? . (28)
02:24:16.075162 IP 94.254.84.99.52677 > 192.5.5.241.53: 61834% [1au] DNSKEY? . (28)
02:24:16.084444 IP 94.254.84.99.34196 > 192.58.128.30.53: 31026% [1au] DNSKEY? . (28)
02:24:16.093283 IP 94.254.84.99.24101 > 198.41.0.4.53: 29121% [1au] DNSKEY? . (28)
02:24:16.216303 IP 94.254.84.99.31773 > 199.7.83.42.53: 64270% [1au] DNSKEY? . (28)
02:24:16.601690 IP 94.254.84.99.33871 > 193.0.14.129.53: 31556% [1au] DNSKEY? . (28)
02:24:16.626838 IP 94.254.84.99.14702 > 192.36.148.17.53: 11600% [1au] DNSKEY? . (28)
02:24:16.650103 IP 94.254.84.99.33475 > 198.41.0.4.53: 29963% [1au] DNSKEY? . (28)
02:24:16.749918 IP 94.254.84.99.48750 > 192.58.128.30.53: 25666% [1au] DNSKEY? . (28)
02:24:16.759274 IP 94.254.84.99.40180 > 202.12.27.33.53: 39954% [1au] DNSKEY? . (28)
02:24:16.792218 IP 94.254.84.99.10986 > 128.63.2.53.53: 22159% [1au] DNSKEY? . (28)
02:24:16.898463 IP 94.254.84.99.55716 > 199.7.83.42.53: 44138% [1au] DNSKEY? . (28)
02:24:17.041216 IP 94.254.84.99.1394 > 128.8.10.90.53: 7667% [1au] DNSKEY? . (28)
02:24:17.177412 IP 94.254.84.99.3746 > 192.112.36.4.53: 27260% [1au] DNSKEY? . (28)
02:24:17.221656 IP 94.254.84.99.15528 > 192.228.79.201.53: 32483% [1au] DNSKEY? . (28)
02:24:17.395147 IP 94.254.84.99.23069 > 192.33.4.12.53: 10234% [1au] DNSKEY? . (28)
02:24:17.497258 IP 94.254.84.99.34658 > 192.203.230.10.53: 37578% [1au] DNSKEY? . (28)
```

```
02:24:17.702866 IP 94.254.84.99.31632 > 192.5.5.241.53: 35419% [1au] DNSKEY? . (28)
02:24:18.650866 IP 94.254.84.99.50167 > 81.228.8.16.53: 32542% [1au] DNSKEY? se. (31)
02:24:19.091562 IP 94.254.84.99.41520 > 192.36.148.17.53: 50842% [1au] DNSKEY? . (28)
02:24:19.092767 IP 94.254.84.99.13229 > 192.58.128.30.53: 63926% [1au] DNSKEY? . (28)
02:24:19.102158 IP 94.254.84.99.42637 > 192.112.36.4.53: 2401% [1au] DNSKEY? . (28)
02:24:19.261940 IP 94.254.84.99.25110 > 128.8.10.90.53: 48445% [1au] DNSKEY? . (28)
02:24:19.398002 IP 94.254.84.99.45560 > 192.33.4.12.53: 2984% [1au] DNSKEY? . (28)
02:24:19.678568 IP 94.254.84.99.30179 > 128.63.2.53.53: 62561% [1au] DNSKEY? . (28)
02:24:19.785383 IP 94.254.84.99.26524 > 199.7.83.42.53: 32847% [1au] DNSKEY? . (28)
02:24:19.928220 IP 94.254.84.99.8158 > 192.5.5.241.53: 15109% [1au] DNSKEY? . (28)
02:24:19.960570 IP 94.254.84.99.57312 > 192.203.230.10.53: 27997% [1au] DNSKEY? . (28)
02:24:20.143191 IP 94.254.84.99.15841 > 192.228.79.201.53: 47336% [1au] DNSKEY? . (28)
02:24:20.416923 IP 94.254.84.99.5673 > 198.41.0.4.53: 1259% [1au] DNSKEY? . (28)
02:24:20.516689 IP 94.254.84.99.43887 > 202.12.27.33.53: 40750% [1au] DNSKEY? . (28)
02:24:20.956532 IP 94.254.84.99.49795 > 193.0.14.129.53: 33346% [1au] DNSKEY? . (28)
02:24:20.981541 IP 94.254.84.99.19266 > 202.12.27.33.53: 27952% [1au] DNSKEY? . (28)
02:24:21.014736 IP 94.254.84.99.42658 > 193.0.14.129.53: 64022% [1au] DNSKEY? . (28)
02:24:21.061652 IP 94.254.84.99.14251 > 192.58.128.30.53: 39129% [1au] DNSKEY? . (28)
02:24:21.071089 IP 94.254.84.99.9557 > 192.36.148.17.53: 25139% [1au] DNSKEY? . (28)
02:24:21.072224 IP 94.254.84.99.7751 > 198.41.0.4.53: 38349% [1au] DNSKEY? . (28)
02:24:21.171952 IP 94.254.84.99.22796 > 192.112.36.4.53: 46573% [1au] DNSKEY? . (28)
02:24:21.358998 IP 94.254.84.99.32549 > 128.63.2.53.53: 1168% [1au] DNSKEY? . (28)
02:24:21.465026 IP 94.254.84.99.64224 > 192.228.79.201.53: 59098% [1au] DNSKEY? . (28)
02:24:21.629747 IP 94.254.84.99.23750 > 192.203.230.10.53: 44773% [1au] DNSKEY? . (28)
02:24:21.811884 IP 94.254.84.99.16476 > 128.8.10.90.53: 14941% [1au] DNSKEY? . (28)
02:24:21.948733 IP 94.254.84.99.20059 > 192.33.4.12.53: 63422% [1au] DNSKEY? . (28)
02:24:22.252741 IP 94.254.84.99.40101 > 192.5.5.241.53: 33742% [1au] DNSKEY? . (28)
02:24:22.492797 IP 94.254.84.99.13114 > 199.7.83.42.53: 46215% [1au] DNSKEY? . (28)
02:24:24.231737 IP 94.254.84.99.13287 > 192.58.128.30.53: 27130% [1au] DNSKEY? . (28)
02:24:24.240988 IP 94.254.84.99.11302 > 202.12.27.33.53: 53827% [1au] DNSKEY? . (28)
02:24:24.295012 IP 94.254.84.99.52401 > 192.112.36.4.53: 51039% [1au] DNSKEY? . (28)
02:24:24.338740 IP 94.254.84.99.24675 > 192.36.148.17.53: 23882% [1au] DNSKEY? . (28)
02:24:24.455822 IP 94.254.84.99.48441 > 199.7.83.42.53: 30666% [1au] DNSKEY? . (28)
02:24:24.622057 IP 94.254.84.99.36735 > 128.63.2.53.53: 32851% [1au] DNSKEY? . (28)
02:24:24.728204 IP 94.254.84.99.3677 > 192.33.4.12.53: 56212% [1au] DNSKEY? . (28)
02:24:24.829849 IP 94.254.84.99.39834 > 128.8.10.90.53: 40586% [1au] DNSKEY? . (28)
02:24:24.966177 IP 94.254.84.99.25997 > 192.228.79.201.53: 8232% [1au] DNSKEY? . (28)
02:24:25.136065 IP 94.254.84.99.44025 > 192.203.230.10.53: 7558% [1au] DNSKEY? . (28)
02:24:25.611681 IP 94.254.84.99.30920 > 192.5.5.241.53: 64354% [1au] DNSKEY? . (28)
02:24:25.621239 IP 94.254.84.99.54622 > 198.41.0.4.53: 17493% [1au] DNSKEY? . (28)
02:24:25.863965 IP 94.254.84.99.53376 > 193.0.14.129.53: 7518% [1au] DNSKEY? . (28)
02:24:26.131782 IP 94.254.84.99.50817 > 192.36.148.17.53: 57910% [1au] DNSKEY? . (28)
02:24:26.133004 IP 94.254.84.99.19517 > 192.112.36.4.53: 35239% [1au] DNSKEY? . (28)
02:24:26.177496 IP 94.254.84.99.37643 > 193.0.14.129.53: 24338% [1au] DNSKEY? . (28)
02:24:26.202529 IP 94.254.84.99.52307 > 202.12.27.33.53: 36197% [1au] DNSKEY? . (28)
02:24:26.372261 IP 94.254.84.99.38358 > 192.33.4.12.53: 62066% [1au] DNSKEY? . (28)
02:24:26.473699 IP 94.254.84.99.50546 > 192.58.128.30.53: 25251% [1au] DNSKEY? . (28)
02:24:26.482875 IP 94.254.84.99.64029 > 128.8.10.90.53: 58902% [1au] DNSKEY? . (28)
02:24:26.619593 IP 94.254.84.99.58903 > 199.7.83.42.53: 41889% [1au] DNSKEY? . (28)
02:24:26.762160 IP 94.254.84.99.40222 > 192.228.79.201.53: 22137% [1au] DNSKEY? . (28)
02:24:26.926460 IP 94.254.84.99.62333 > 128.63.2.53.53: 65394% [1au] DNSKEY? . (28)
02:24:27.027743 IP 94.254.84.99.22093 > 192.203.230.10.53: 7432% [1au] DNSKEY? . (28)
02:24:27.411593 IP 94.254.84.99.55371 > 198.41.0.4.53: 42687% [1au] DNSKEY? . (28)
02:24:27.511283 IP 94.254.84.99.16106 > 192.5.5.241.53: 40790% [1au] DNSKEY? . (28)
02:24:28.139767 IP 94.254.84.99.47032 > 199.7.49.30.53: 14579% [1au] DNSKEY? se. (31)
02:24:28.897654 IP 94.254.84.99.43654 > 198.41.0.4.53: 54161% [1au] DNSKEY? . (28)
02:24:28.997536 IP 94.254.84.99.25918 > 192.33.4.12.53: 60127% [1au] DNSKEY? . (28)
02:24:29.099698 IP 94.254.84.99.49060 > 192.36.148.17.53: 3319% [1au] DNSKEY? . (28)
02:24:29.100813 IP 94.254.84.99.41907 > 202.12.27.33.53: 24686% [1au] DNSKEY? . (28)
02:24:29.276367 IP 94.254.84.99.57741 > 192.228.79.201.53: 18709% [1au] DNSKEY? . (28)
02:24:29.444457 IP 94.254.84.99.35651 > 128.63.2.53.53: 17366% [1au] DNSKEY? . (28)
02:24:29.550502 IP 94.254.84.99.46633 > 128.8.10.90.53: 31631% [1au] DNSKEY? . (28)
02:24:29.686722 IP 94.254.84.99.51817 > 192.203.230.10.53: 21841% [1au] DNSKEY? . (28)
02:24:29.868353 IP 94.254.84.99.38504 > 192.58.128.30.53: 62775% [1au] DNSKEY? . (28)
02:24:29.877735 IP 94.254.84.99.42031 > 192.112.36.4.53: 14691% [1au] DNSKEY? . (28)
02:24:29.924851 IP 94.254.84.99.26754 > 193.0.14.129.53: 64511% [1au] DNSKEY? . (28)
02:24:29.949864 IP 94.254.84.99.2531 > 192.5.5.241.53: 550% [1au] DNSKEY? . (28)
02:24:30.253624 IP 94.254.84.99.39685 > 199.7.83.42.53: 9481% [1au] DNSKEY? . (28)
02:24:30.750609 IP 94.254.84.99.56933 > 192.36.148.17.53: 51500% [1au] DNSKEY? . (28)
02:24:30.751839 IP 94.254.84.99.27448 > 202.12.27.33.53: 42804% [1au] DNSKEY? . (28)
02:24:30.784020 IP 94.254.84.99.41231 > 192.58.128.30.53: 30428% [1au] DNSKEY? . (28)
02:24:30.793702 IP 94.254.84.99.49739 > 193.0.14.129.53: 58392% [1au] DNSKEY? . (28)
02:24:30.934610 IP 94.254.84.99.48978 > 128.63.2.53.53: 47271% [1au] DNSKEY? . (28)
02:24:31.041492 IP 94.254.84.99.47918 > 192.33.4.12.53: 17719% [1au] DNSKEY? . (28)
02:24:31.165086 IP 94.254.84.99.5257 > 128.8.10.90.53: 18477% [1au] DNSKEY? . (28)
02:24:31.301195 IP 94.254.84.99.40812 > 192.112.36.4.53: 14613% [1au] DNSKEY? . (28)
02:24:31.345951 IP 94.254.84.99.2408 > 199.7.83.42.53: 3356% [1au] DNSKEY? . (28)
02:24:31.488513 IP 94.254.84.99.23684 > 192.203.230.10.53: 7238% [1au] DNSKEY? . (28)
02:24:31.670678 IP 94.254.84.99.25187 > 192.228.79.201.53: 46507% [1au] DNSKEY? . (28)
02:24:32.160626 IP 94.254.84.99.51092 > 198.41.0.4.53: 8985% [1au] DNSKEY? . (28)
02:24:32.503152 IP 94.254.84.99.27146 > 192.5.5.241.53: 27775% [1au] DNSKEY? . (28)
02:24:33.838154 IP 94.254.84.99.37560 > 192.5.5.241.53: 10852% [1au] DNSKEY? . (28)
02:24:33.870844 IP 94.254.84.99.64862 > 192.36.148.17.53: 360% [1au] DNSKEY? . (28)
02:24:33.872050 IP 94.254.84.99.1247 > 192.33.4.12.53: 26639% [1au] DNSKEY? . (28)
02:24:33.974399 IP 94.254.84.99.11003 > 128.63.2.53.53: 4046% [1au] DNSKEY? . (28)
02:24:34.081201 IP 94.254.84.99.53564 > 202.12.27.33.53: 27940% [1au] DNSKEY? . (28)
02:24:34.113975 IP 94.254.84.99.28350 > 198.41.0.4.53: 2696% [1au] DNSKEY? . (28)
02:24:34.472201 IP 94.254.84.99.3990 > 192.112.36.4.53: 12967% [1au] DNSKEY? . (28)
02:24:34.516196 IP 94.254.84.99.4686 > 128.8.10.90.53: 50131% [1au] DNSKEY? . (28)
```

```
02:24:34.652189 IP 94.254.84.99.24962 > 192.228.79.201.53: 62757% [1au] DNSKEY? . (28)
02:24:34.823875 IP 94.254.84.99.22691 > 192.203.230.10.53: 44720% [1au] DNSKEY? . (28)
02:24:35.029079 IP 94.254.84.99.64213 > 193.0.14.129.53: 34740% [1au] DNSKEY? . (28)
02:24:35.591274 IP 94.254.84.99.6733 > 192.58.128.30.53: 52678% [1au] DNSKEY? . (28)
02:24:35.600530 IP 94.254.84.99.56447 > 199.7.83.42.53: 23828% [1au] DNSKEY? . (28)
02:24:35.743135 IP 94.254.84.99.61441 > 192.36.148.17.53: 39443% [1au] DNSKEY? . (28)
02:24:35.767682 IP 94.254.84.99.49506 > 192.112.36.4.53: 56757% [1au] DNSKEY? . (28)
02:24:35.811374 IP 94.254.84.99.18699 > 192.5.5.241.53: 35094% [1au] DNSKEY? . (28)
02:24:35.820758 IP 94.254.84.99.2640 > 128.63.2.53.53: 1426% [1au] DNSKEY? . (28)
02:24:35.921474 IP 94.254.84.99.16880 > 199.7.83.42.53: 51500% [1au] DNSKEY? . (28)
02:24:36.292388 IP 94.254.84.99.10379 > 128.8.10.90.53: 53198% [1au] DNSKEY? . (28)
02:24:36.607536 IP 94.254.84.99.17679 > 192.228.79.201.53: 13992% [1au] DNSKEY? . (28)
```

Figure 12 - root queries for the DNSKEY for .se se - (provided by Patrik Wallström)

This should look familiar, as it is precisely the same query pattern as happened with the in-addr.arpa servers and the .se servers, although the volume of repeated DNSKEY queries is somewhat alarming. When the client receives a response from a subdomain that needs to be validated against the root, and when the queries to the root are not validatable against the local trust key, and the client goes into a tight loop of repeated queries against the authoritative listed name servers of the .se zone. By anchoring the local resolver in a single root, with a key state which invalidates the signatures of all authoritative servers of the zone, but which authoritatively (absent DNSSEC) confirms them as valid servers of the zone, places the DNS resolver client instance in a loop: no authoritative name server that it can query has a signature that the client can accept by local validation, but the root informs it only these name servers can be used.

It appears that the client is being quite inventive with its attempts to find a signature that it can validate against its local trust key, and queries all servers, and does not limit itself to the single server that was the first to answer. Further tests have also shown that the client does not cache the information that the DNSKEY cannot be validated, and reinitiates this spray of repeated queries against the name servers when a subsequent DNSSEC query is made in a subzone. Therefore the behaviour is promiscuous in two distinct ways. Firstly, from the in-addr.arpa traces made at a single NS, it is clear that any NS so queried is repeatedly queried. From the .se tests, its clear that all NS of a zone can be queried.

The clients tested here are BIND 9.5.0 and BIND 9.7.0rc2 (https://www.isc.org/software/bind).

An Examination of BIND

When BIND validates, it does bottom up, depth first search on all possible paths to establish the trust-chain from the initial received query, to a configured trust anchor.

As an example, assume a TXT RRset for *test.example.com* in a signed *example.com* zone. The zone *example.com* resides on 2 name server addresses. The *example.com* zone has a KSK, which is referred to by the DS record in the .com zone. The .com zone is signed, and resides on 14 addresses, (11 IPv4 and 3 IPv6). The *.com* zone has a KSK, which is referred to by the DS record in the root zone, or by the trust anchor in the resolver's local configuration.

Assume that either the DS record for *.com* in the root, or the locally held trust anchor for *.com* in the resolver has become stale. That is, the DS record for *.com* in the root zone validates, but there are no DNSKEYs in *.com* that matches the DS record in the root zone.

Now lets assume that BIND is resolving a query relating to *test.example.com*. The following depth first search occurs:

- BIND resolves the *test.example.com* RRset. It attempts to validate it. To do so, it needs the *example.com* DNSKEY RRset.

- It resolves the DNSKEY RRset for *example.com*. It attempts to validate it. To do so, it needs the *example.com* DS RRset.

- It resolves the DS RRset for *example.com*. It attempts to validate it. To do so, it needs the .com DNSKEY RRset.

- It resolves the DNSKEY RRset for .com. It attempts to validate it. It does this with the configured trust anchor.

However, the resolver can't validate the .com DNSKEY RRset, as it has not the proper trust anchor for it. It will query all remaining 13 servers for the DNSKEY RRset for .com. After this, the resolver still does not have the proper *.com* DNSKEY, and will track back one level:

- It resolves the DS RRset for *example.com* from the next authoritative server. It attempts to validate it. To do so, it needs the .com DNSKEY RRset. The depth first search goes forward again.

- It resolves the DNSKEY RRset for *.com*. It attempts to validate it. It does this with the configured trust anchor.

Since the DNSKEY has RRset for .com hasn't changed, this will fail as well.

The complete depth first search comprises of:
- TXT records on 2 servers, signed by:
- DNSKEY records on 2 servers, referred to by:
- DS records on 14 servers, signed by:
- DNSKEY records on 14 servers, supposedly matching the trust-anchor.

When all possible paths have been exhausted, BIND will have sent:
- 784 (2*2*14*14) .com DNSKEY requests to 14 .com name servers,
- 56 (2*2*14) *example.com* DS requests to 14 .com name servers,
- 4 (2*2*) *example.com* DNSKEY requests to 2 *example.com* name servers.

The ratio between DS and DNSKEY (to the .com name servers) is 14. This is exactly the amount of name servers available.

Consider the case where the local trust anchor for root has become stale. There are 19 root server addresses:
- 283024 (2*2*14*14*19*19) root DNSKEY requests to 19 root name servers,
- 14896 (2*2*14*14*19) .com DS requests to 19 root name servers,
- 784 (2*2*14*14) .com DNSKEY requests to 14 .com name servers,
- 56 (2*2*14) *example.com* DS requests to 14 .com name servers,
- 4 (2*2) *example.com* DNSKEY requests to 2 *example.com* name servers.

Note that the level of depth is important here: the longer the broken chain, the more queries that will be sent. In the first example, the level is 2 deep, from *test.example.com* to .com. In the latter example, the level is 3 deep. It's worthwhile noting in this context that reverse trees and enum trees in the *.arpa* zone are longer on average. Though delegations in those sub-trees might span several labels, it is not uncommon to delegate per label.

Note also that the entire effort is done per incoming query!

Though the examples show an enormous query load, there are a few ceilings. Every run is done in serial. The run is stopped after 30 seconds. This value is hardcoded in BIND and can't be configured:

"lib/dns/resolver.c" line 3586:
```
/*
 * Compute an expiration time for the entire fetch.
 */
```

```
isc_interval_set(&interval, 30, 0);             /* XXXRTH constant */
```

However, the scope of these ceilings are per query. Even with a small amount of incoming queries, the resulting load is very large.

Conclusion: BIND tries to construct the chain of trust in an exhaustive and aggressive way, when there is a stale link somewhere in the chain. The longer the trust chain is between the stale link and the target qname, the more queries will be send. The more name server addresses available for a domain, the more queries will be send.

Analysis by Roy Arends

The Unbound client (http://www.unbound.net/) also appears to have a similar behaviour, although it is not as intense because of the cache behaviour in this implementation. Unbound will "remember" the query outcome for a further 60 seconds, so repeated queries for the same name will revert to the cache. But the DNSSEC key validation failure is per zone, and further queries for other names in the same zone will still exercise this re-query behaviour. In effect, for a zone which has sufficient 'traffic' of DNS load in sub-zones or instances inside that zone, the chain of repeated queries is constantly renewed and kept alive.

An Examination of UNBOUND

UNBOUND basically consists of a validator and iterator module. First, the iterator module resolves the data. Then the validator module validates the data. The validator asks the iterator for missing data. The cache is a message cache: messages are stored using qname/qclass/qtype lookup key. The associated RRsets are stored separately to avoid redundancy. When no validation happens (no trust anchor configured), or when validation succeeds, messages are stored for the duration of the lowest TTL in the response, whereas RRSets are stored for the duration of their TTL. So far, so good!

When validation fails, the validator will tell the iterator to re-fetch the DNSKEY set from a different server. It will retry this 5 times. This value is hardcoded (VAL_MAX_RESTART_COUNT).

When validation fails after VAL_MAX_RESTART_COUNT is exhausted, the data is stored for the duration of the configurable VAL_BOGUS_TTL, which has a default of 60 seconds, or for NULL_KEY_TTL (hardcoded to 900 seconds), whichever of the two is smaller.

The default value of 60 seconds causes UNBOUND to restrain itself. However, since its a per-message cache, it only restrains itself for that qname/qclass/qtype tuple. Hence, if a different query is asked, UNBOUND needs to validate the response, sees a bogus DNSKEY in the cache and starts to re-fetch the DNSKEY keyset. In other words, a lame root key will cause DNSKEY queries for every unique query seen per 60 second window.

Additionally, there is no way for the validator to know if a DS record is lame (i.e. validates, but points to non-existent keys), or if the KSK at the child is lame. Hence, when a DS record somewhere in the chain is lame, UNBOUND behaves the same as if a configured trust anchor is lame.

A possible resolution of this is for UNBOUND to mark lame keys as being lame for the entire zone, and not have these keys re-fetched and re-validated per unique query.

Note that during this entire process, none of the signatures over records are broken, timed out, etc. They all match up. What has triggered this behavior is a misconfiguration of a trust-anchor by the client, or some stale DNSKEY, or DS record.

This can be used to make some conjectures about the traffic implications.

> The fact that VAL_BOGUS_TTL is configurable is of little comfort. A single resolver with a large constituency can receive 3000 unique queries per minute, or 50 queries per second (qps). A misconfigured root-key will thus result in a query storm of at least 50 times 5 DNSKEY requests for root, i.e. 250 qps. Considering a root zone DNSKEY response size of 736 bytes (current DNSKEY response size from l-root), this will cause a bandwidth consumption of 1.4 Mbps. This is then distributed over the 13 root-servers. Assuming that the bulk of the traffic is divided over 13 IPv4 root addresses, that's a stream of 110 Kbps per server (or about 20 qps per server) from a single resolver.
>
> *Analysis by Roy Arends*

The emerging picture is that misconfigured local trust keys in a DNS resolver for a zone can cause large increases in the DNS query load to the Name Servers of that zone, where the responses to these additional queries are themselves large, being of the order of 1,000 bytes in every response. And this can occur for any DNSSEC signed zone.

The conditions for the client to revert to a rapid re-query behaviour are where:

- The DO bit is honoured by the server,
- The DNS data appears to be signed, and
- The signature check fails.

The client response is to aggressively attempt to re-fetch the DNSKEY RRs for the zone from any of the zone's servers to see if the attack can be mitigated via a rapid spread of the queries to a broader set of servers. The other part of the client response is not to cache validation failure for the zone in the event that this repeated query phase does not provide the client with a locally validated key. After all, the data is provably false, so caching it would be to retain something which has been "proven" to be wrong.

The conditions being set up for the DURZ approach for signing the root are:

- The DO bit is honoured by the server,
- The DNS data appears to be signed, and
- The signature check fails.

What's to stop the DNS root servers being subjected to the same spike in the query load?

The appropriate client behaviour for this period of DNSSEC deployment at the root is not to enable DNSSEC validation in the resolver. While this is sound advice, it is also the case that many resolvers have already enabled validation in their resolvers, and are probably not going to turn it off for the next six months while the root servers gradually deploy DNSSEC using DURZ. But what if a subset of the client resolvers start to believe that these unvalidatable root keys should be validated?

What If...

The problem with key rollover and local management of trust keys appears to be found in around 1 in every 1,500 resolvers in the in-addr.arpa zones. With a current client population of some 1.5 million distinct resolver client addresses each day for these in-addr.arpa zones, there are some 1,000 resolvers who have lapsed into this repeated query mode following the most recent key rollover of December 2009. There are 6 NS records in each subzone of in-addr.arpa, and all servers see this pathological re-query behaviour following key rollover.

The root servers see a set of some 5 million distinct resolver addresses each day and a comparable population of non-updated resolvers would be in the order of some 3,000 resolvers querying 13 zone servers, where each zone server would see an incremental load of some 75Mbps.

Since the re-query behaviour is driven by the client being forced to reject the supposedly authoritative response because of an invalid key, and since DURZ is by definition an invalid key, the risk window for this increased load is the period during which DURZ is enabled, which for the current state of the root signing deployment is from the present date until July 2010. Since not all root servers have DNSSEC content or respond to the DO bit, and therefore do not return the unvalidatable signatures, the risk is limited to the set of DNSSEC enabled roots, which is increasing on a planned, staged rollout. A decision to delay deployment of the DNSSEC/DURZ sign state to the "A" root server instance was made because this receives a noted higher query load for the so-called "priming" queries, made when a resolver is re-initialized and uses the offline root "hints" file to bootstrap more current knowledge. It is therefore likely that the "A" root server would also see increased instances of this particular query model, should the priming query be implicated in this form of traffic.

Arguably, this is an unlikely situation. For most patterns of DNS query, failure to validate is immediately apparent. After all, where previously, you receive an answer, you now see your DNS queries time out and fail.

However, because the typical situation for a client host (including DHCP initialized hosts in the customer network space, the back office, etc.) is to have more than one listed resolver, there is the possibility of a misconfiguration being unnoticed, during the period of a rolling deployment of DNSSEC enabled services. In this situation if only one of the resolver's 'nserver' entries is DNSSEC enabled, either it is not queried, or it is queried, but then passed over by the resolver timeout setting. Users see slower DNS resolution, but can put this down to network delay, or other local problems.

A second argument is that installation of hand-trust material is not normal, so the servers in question will be immediately known because a non-standard process has to be invoked. Unfortunately, this is demonstrably not true. For example, the *Fedor*a release of Linux (http://fedoraproject.org/) has included a simple DNSSEC enabling process including a pre-configured trust file, covering the RIPE NCC reverse-DNS ranges. Because a previous release of this software included now stale keys (which have since been withdrawn in subsequent releases), any instance of this particular release of *Fedora* being enabled will not only be unable to process reverse-DNS, it may also invoke this re-query mode of operation that places the server under repeated load of DNSKEY requests. Because reverse-DNS is the 'infrastructure' DNS query which is typically logged, but not otherwise used, unless the server in question is configured to block service on failing reverse (which is unlikely, given that over 40% of reverse-DNS delegations are not made for the currently allocated IP address ranges) this behaviour simply might never be noticed by the end user. The use of so-called "Live CDs" can exacerbate this problem of pre-primed software releases that include key material that falls out of date. Even when the primary release is patched, the continued use of older releases in the field is inevitable. So perhaps this second argument is not quite so robust as originally thought.

Lastly, distinct from hand-installed local trust, is the use of DNSSEC look-aside validation which is known as 'DLV'. This is a privately managed DNS namespace, which has been using the ICANN maintained "interim trust anchor repository" or ITAR. The DLV service is configured to permit resolvers to query it, in place of the root, to establish trust over subzones which exist in a signed state, but cannot be seen as signed from the root downwards before the deployment of a signed root. There is now evidence that part of this query space exists, covering zones of interest to this situation. *.se* for instance, is in the ITAR, and so includes ranges which can be found in the DLV namespace, as do the *in-addr.arpa* spaces signed by the RIPE NCC.

Evidence suggests that if the DLV chain is being used, and a key rollover takes place, some variants of BIND fail to re-establish trust over the new keys until rebooted with a clean cache state. This is difficult to confirm because as each resolver is restarted, the stale trust state is wiped out and the local failure is immediately resolved.

Post DURZ

Of course this is a transitory phase, and even if there are issues in terms of DURZ and queries to the root servers, all will be resolved once the root key is rolled to a validatable key on 1 July 2010.

Yes?

Maybe not.

Lets go back to the ICANN material relating to signing the root, and look at the frequency of trust key rollover (Figure 13).

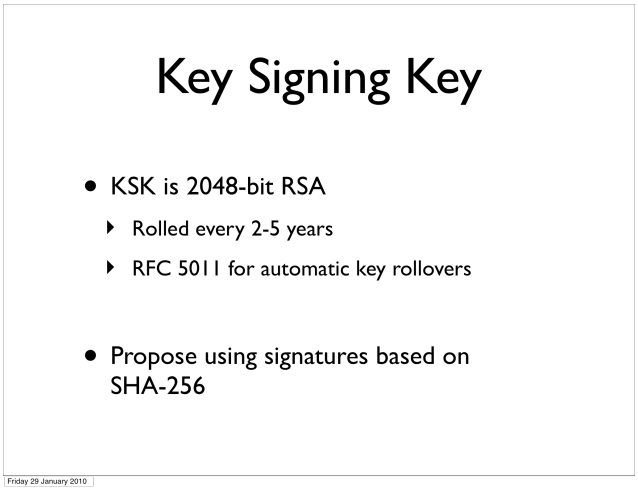

Figure 13 - Root Zone Key Rollover - Joe Abley, ICANN, January 2010

The implication is that sometime every 2 to 5 years, all DNS resolvers will need to ensure that they have fetched a new root trust key and loaded it into their resolver's local trust key cache.

If this local update of the root's trust key does not occur then the priming query for such DNSSEC enabled resolvers will encounter this problem of an invalid DNSKEY when attempting to validate the priming response from the root servers. The fail safe option here for the resolver client is to enter a failure mode and shutdown, but there is a strong likelihood that the resolver client will try as hard as it can to fetch a validatable DNSKEY for the root before taking the last resort of a shutdown, and in so doing will subject the root servers to this intense repeated query load that we are seeing on the in-addr.arpa zone.

A reasonable question to ask is: "Are there any procedural methods to help prevent stale keys being retained during key rollover?" Reassuringly, the answer is "Yes." There is a relatively recent RFC, "Automated Updates of DNS Security (DNSSEC) Trust Anchors," RFC5011 (http://tools.ietf.org/html/rfc5011), which addresses this problem.

RFC5011 provides a mechanism for both signalling that a key rollover needs to take place, and for forward declaring the use of keys to sign over the new trust set, to permit in-band distribution of the new keys. Resolvers are required to be configured with additional keying, and a level of trust is placed on this mechanism to deal with normal key rollover. RFC5011 does not solve initial key distribution problems, which of course must be made out of band, nor does it attempt to address multiple key failures. Cold standby equipment, or decisions to return to significantly older releases of systems (e.g., if a major security compromise to an operating system release demands a rollback) could still potentially deploy resolvers with invalid, outdated keys. However RFC5011 will prevent the more usual process failures, and provides an elegant in-band re-keying method which obviates a manual process of key management which all too often fails through neglect or ignorance of the appropriate maintenance procedures to follow.

It is unfortunate that RFC5011 compliant systems are not widely deployed during the lifetime of the DURZ deployment of the root, since we are definitely going to see at least one key rollover at the end of the DURZ deployment, and can expect a followup key rollover within a normal operations window. The alternative is that no significant testing of root trust rollover takes place until we are committed to validation as a normal operational activity, which invites the prospect of production deployment across the entire root set while a number of production operational processes associated with key rollover remain untested. The evidence from past issues in resolver behaviour is that older deployments have a very long lifetime for any feature under consideration, and since BIND 9.5 and older pre-release BIND 9.7 systems can be expected to persist in the field in significant numbers for some years to come, it is likely a significant level of pathological resolver behaviour in re-querying the root services by active resolvers will have to be tolerated for some time.

It is also concerning that aspects of the packet traces for the in-addr.arpa zone suggest that for all key rollovers, albeit at very low levels of query load, some amount of the resolvers have simply failed to take account of the new keys, and may never do so. Therefore, with increasing deployment of validation, it is possible that a substantial new traffic class which grows, peaks, and then declines, but always declines to a slightly higher value than before has to be borne, and factored into deployment scaling and planning. Since this traffic is large, generating a kilobyte of response per query, and potentially generally prevalent, it has the capability to exceed the normal response requirements for "normal" DNS query loads by at least one, if not two orders of magnitude. This multiplication factor of load is defined by the size of the resolver space, and number of listed NS for the affected zone.

Mitigation at the server side is possible if this proves to become a major problem. The pattern of re-query here (the sequence of repeated queries for DNSKEY RRs) appears a potential signature for this kind of problem. Given that for any individual server the client times its repeat queries on the reception of the response from the previous query, delaying the server's response to the repeated query will further delay the client making its repeated query to this server. If the server were in a position to delay such repeated responses, using a form of exponential increase in the delay timer or similar form of time penalty, then the worst effects of this form of client behaviour in terms of threats to the integrity of the server's ability to service the "legitimate" client load could be mitigated.

It is an inherent quality of the DNSSEC deployment that in seeking to prevent lies, an aspect of DNS ‘stability’ has been weakened. When a client falls out of synchronization with the current key state of DNSSEC then it will mistake the current truth for an attempt to insert a lie. The client's subsequent efforts to perform a rapid search for what it believes to be a truthful response by both stepping up the frequency and breadth of repeat queries could reasonably be construed as a legitimate response if indeed this was an instance of an attack on that particular client. Indeed, to do otherwise would be to permit the DNS to remain an un-trustable source of information. However, in this situation of slippage of synchronized key state between client and server, the effect is both local failure and the generation of excess load on external servers, which, if this situation is allowed to become a common state, has the potential to broaden the failure state to a more general DNS service failure through load saturation of critical DNS servers.

This is an unavoidable aspect of a qualitative change of the DNS, and places a strong imperative on DNS operations, and the community of the 5 million current and uncountable future DNS resolvers, to understand that "set and forget" is not the intended mode of operation of DNSSEC.

Authors

George Michaelson is a Research Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific Region.

Patrik Wallström has been working on DNSSEC and the development of the registry system at .SE for seven years, and with computer security and open source for over 15 years.

Roy Arends is Senior Researcher at Nominet UK, the Internet Registry for .uk domain names.

Geoff Huston the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region.