March 2024
Geoff Huston

# KeyTrap!

The National Research Center for Applied Cybersecurity ATHENE has uncovered a critical flaw in the design of DNSSEC, the Security Extensions of DNS (Domain Name System). DNS is one of the fundamental building blocks of the Internet. The design flaw has devastating consequences for essentially all DNSSEC-validating DNS implementations and public DNS providers, such as Google and Cloudflare. The ATHENE team, led by Prof. Dr. Haya Schulmann from Goethe University Frankfurt, developed "KeyTrap", a new class of attacks: with just a single DNS packet hackers could stall all widely used DNS implementations and public DNS providers. Exploitation of this attack would have severe consequences for any application using the Internet including unavailability of technologies such as web-browsing, e-mail, and instant messaging. With KeyTrap, an attacker could completely disable large parts of the worldwide Internet.
https://www.athene-center.de/en/news/press/key-trap, 13 Feb 2024

Really?

The language of the press release is certainly dramatic, with "devasting consequences" and the threat to "completely disable large parts of the worldwide Internet." If this is really so devastating then perhaps we should look at this in a little more detail to see what's going on, how this vulnerability works, and what the response has been.

But before we launch into the details of *KeyTrap* perhaps its useful to understand the respective roles of the DNS infrastructure.

When a stub resolver passes a query to a recursive resolver, then the recursive resolver must undertake some work to resolve a query. A recursive resolver is *primed* with the names and IP addresses of the authoritative name servers for the root zone of the DNS. The recursive resolver will then query one of these name servers with the original query name and query type. Assuming that the query is not relating to a record that is in the root zone, then the response from the name server is a *referral response* that names the delegated domain, the collection of names that are nameservers for the domain and normally some *glue records* that list the IP addresses of these name server. The recursive resolver then repeats the process with one of these servers. The number of iterative steps for the process is related to the number of delegation points that are used for this domain name. For example, for the query name `www.potaroo.net`, the worst case would be 4 steps, with queries to a root server, a `.net` server, a `potaroo.net` server and the `www.potaroo.net` server (Figure 1). In theory the amount of work that the recursive resolver needs to perform to resolve a name is limited by the label count in the query name.

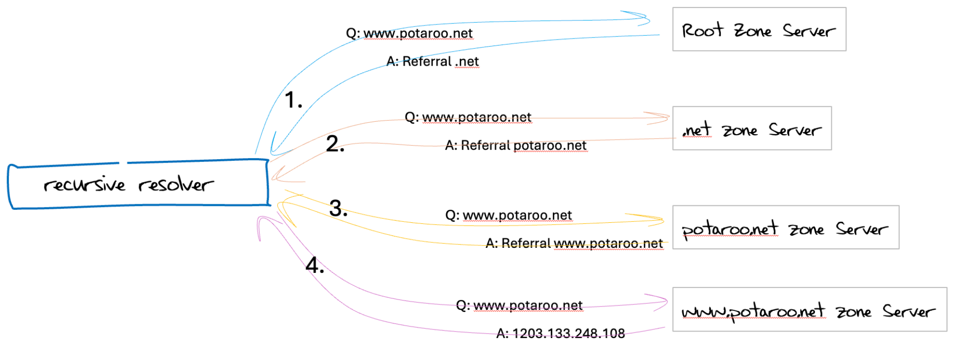


*Figure 1 – Iterative Name Resolution in the DNS*

However, is not necessarily the case, and this depends on the way that the DNS is configured for a zone. The *glue records* are not necessarily provided in the zone, and if they are not located in the zone then they will be omitted in the referral response. If that's the case, then the recursive resolver must suspend its current resolution task and re-apply the same process to find the IP address of the name servers for the name. This could also encounter the same *glueless* situation, requiring stacking up this incomplete resolution and working on the new name server name, and so on. Using a dynamic DNS server it's possible to make this an indefinite situation that will never stop, as described by Florian Maury at a DNS OARC workshop in May 2015.

The mitigation measures for this errant zone configuration are for the recursive resolver to limit the number of subsequent queries that the recursive resolver will perform for a single initial query, and potentially to limit the recursive depth of stacked query tasks. Related mitigations include limiting the total elapsed time spent in any single resolution task, and the "breadth" of subsequent queries ("breadth" is the number of nameservers for a zone that will be queried). It's a balance between performing an exhaustive set of queries to find an answer and protecting the resolver's resources by dropping the query and returning an error code, such as SERVFAIL.

The same kind of behaviour can be constructed with CNAME chains, where the name that is the *target* of a CNAME alias is itself a CNAME, and so on. The mitigation for this situation is the same as for the indefinite *glueless* situation, namely, to limit the number queries that a recursive resolver will perform to resolve a single query name.

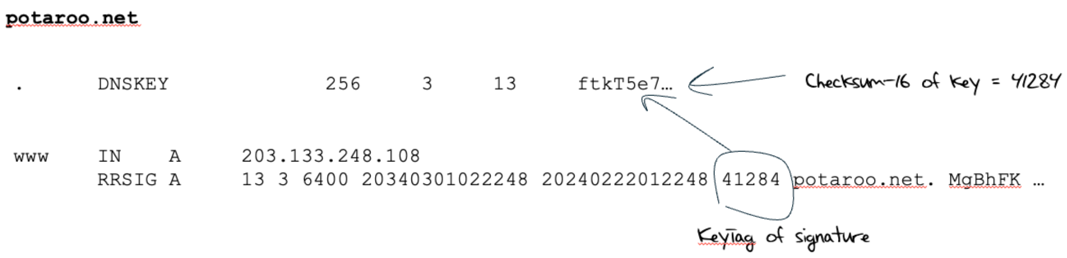
So, it's possible to create situations in a DNS zone, or construct dynamic behaviours in authoritative servers where the resolution of a DNS name sets off a large number of consequent queries, or even a non-terminating set of queries. The mitigation is for the recursive resolver to impose a limit on these consequent queries when resolving a name.

Is it possible to impose an unreasonable load on recursive resolvers when attempting to resolve a name that does not also involve additional queries? Recursive resolvers that perform DNSSEC validation of signed responses provide such a scenario, and the *KeyTrap* situation shows a way to do this.

In a minimal model of DNSSEC, a zone is signed with a *Key*. The value of this key is given in the DNSKEY resource record for the zone, and all signatures in this zone (RRSIG resource records) are generated by using the private part of this key. When a recursive resolver wants to validate a response from this zone, it takes the DNSKEY value and applies it to the signature value to determine if the corresponding private key was used to generate the signature for this resource record.

However, it's not uncommon for a zone to have a number of keys. There is the convention used to separate the key that is passed between the parent and child zones (the Key-Signing Key, or KSK) and the key used to generate the signatures for records in the zone (the Zone-Signing Key, or ZSK). There may also be multiple keys in a zone for certain situations. If a zone wants to support multiple algorithms, or while it is transitioning from one algorithm to another, the zone may have multiple keys and each record may have multiple RRSIG records. In this case when validating record, the recursive resolver will have to use each RRSIG value and find a DNSKEY value with a matching algorithm field, and for each match determine if this key (its private counterpart) was used to generate the signature in seeking to find a match between a signature record and a key.. There is no limit in the DNSSEC specification to the number of keys in a zone, nor the number of RRSIG records that may be associated with a resource record in a signed zone.

In attempting to verify a response the recursive resolver has to compare each RRSIG entry against each DNSKEY key until it finds a match. The cases where the key and the signature use different algorithms can be readily discarded, but the remainder can present a significant computational load. Asymmetric crypto is expense, so performing a full algorithm check between each key and each signature record is combinatorically expensive. To avoid this cost, DNSSEC uses the concept of a *KeyTag*. This value is the 16-bit checksum of the public key value. The tag value is not stored in the DNSKEY resource record but is computed by the recursive resolver when performing DNSSEC validation. Each RRSIG record contains a *KeyTag* value, which is this same 16-bit checksum. The recursive resolver will only perform the more expensive crypto check if the computed key tag value of the key matches the tag value contained in the RRSIG record (Figure 2).



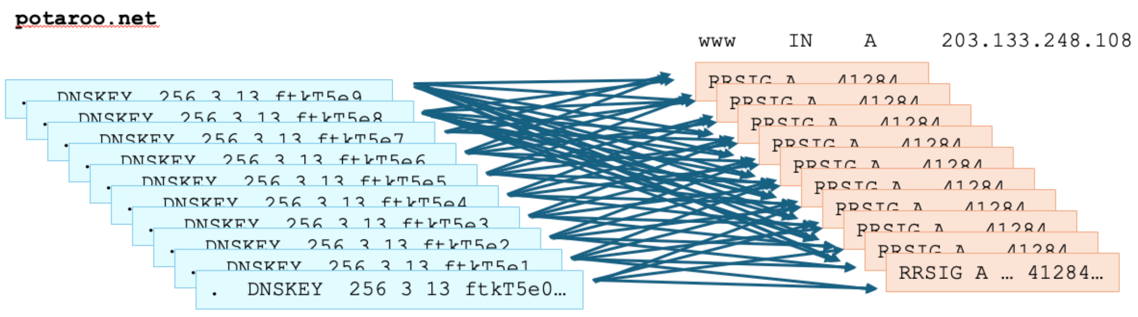
```
potaroo.net

.       DNSKEY           256     3     13      ftkT5e7…

www     IN    A     203.133.248.108
        RRSIG A     13  3  6400  20340301022248  20240222012248  41284  potaroo.net.  MgBhFK …
```

*Figure 2 – Simple Zone Signing*

The problem is that 16-bit checksums are not a terribly good hashing function, and it is possible to generate a large collection of input strings all of which have the same checksum value. If a zone is signed with multiple keys, and each key has the same 16-bit checksum, then the recursive resolver has no choice but to perform the crypto to attempt to find a key that matches any of the RRSIG signature values.

This is a combinatorial issue. If a record has 100 signature records, all with the same key tag value, and the zone has 100 keys all of which have the same 16-bit checksum value, and if there is no match to be found, then the recursive resolver would need to perform 10,000 crypto operations to prove to itself that there is no match. Obviously, that will take time and also processing resources on the part of the recursive resolver, but it will not need to make further queries while it is doing so.

The *KeyTag* attack starts with a specially crafted zone that has a large number of keys, all of which have the same tag value. It also has a record with a large number of attached signature records, all of which have the same key tag value. None of the signatures and keys "match". A query for this query name, when sent to a DNSSEC-validating resolver, will cause the resolver to expend a large amount of processing time to attempt to fine a "match" between a key and a signature (Figure 3).



potaroo.net
www IN A 203.133.248.108
. DNSKEY 256 3 13 ftkT5e9
. DNSKEY 256 3 13 ftkT5e8
. DNSKEY 256 3 13 ftkT5e7
. DNSKEY 256 3 13 ftkT5e6
. DNSKEY 256 3 13 ftkT5e5
. DNSKEY 256 3 13 ftkT5e4
. DNSKEY 256 3 13 ftkT5e3
. DNSKEY 256 3 13 ftkT5e2
. DNSKEY 256 3 13 ftkT5e1
. DNSKEY 256 3 13 ftkT5e0…
RRSIG A 41284
RRSIG A … 41284…

*Figure 3 – KeyTrap zone configuration*

It's by no means "devasting" for the DNS, and the fix is much the same as the previous fix. As well as limiting the number of queries that a resolver can generate to resolve a queried name, a careful resolver will limit both the elapsed time and perhaps the amount of the resolver's processing resources that are used to resolve any single query name.

It's also not a novel discovery by the ATHENE folk. The vulnerability was described five years ago by a student at the University of Twente. I guess the issue was that the student failed to use a sufficient number of hysterical adjectives in describing this DNS vulnerability in the paper!

It's not the only DNSSEC potential vulnerability, of course. There is also the situation in DNSSEC when a validating recursive resolver is configured with an incorrect root zone Key Signing Key (KSK) value. The default behaviour is for the recursive resolver to query along all possible validation paths leading to the root zone to see if there is a path where the keys interlock correctly. In situations where the intermediate zones are configured with a generous collection of name servers, then the number of paths to be queried, and the number of cryptographic tests to be perform becomes a multiplication problem. A validation path of depth 3, with 13 nameservers at each level has some 2,197 distinct paths between the RRSIG and the Root KSK. Local caching reduces the query volume considerably, but the number of crypto operations to validate each path remains the same. This was reported under the generic name of "Roll over and Die".

Then there is the behaviour of recursive resolvers in attempting to validate a NSEC3 non-existence record when the zone is configured with a high iteration count for NSEC3. The additional applications of the hash function add nothing to the opacity of the NSEC3 record but places a high processing burden on a recursive resolver when it attempts to validate the NSEC3 record. This can be exercised by directing a high volume of random query name queries to a recursive resolver when the zone in question is signed using a high hash iteration count. The primary advice to zone admins is "just don't specify high iteration counts" (RFC 9276), as it has no crypto benefit. Again, the advice to recursive resolvers is to use query rate and processing limitations on the incoming query streams. I would suggest that the longer-term advice is the walk away from NSEC3 records completely! If you want to use an efficient front end signer, or prevent zone enumeration, or both, then perhaps the more effective path is to use a compact denial of existence approach to NSEC responses.

I am generally surprised at just how tolerant many recursive resolver implementations are when they encounter misconfigurations in DNS zones. They often exercise significant effort to find an answer in the most adverse of conditions of zone misconfigurations. But such diligence is hardly a virtue. It is, in fact, a weakness, as all an attacker need do is to craft a zone with such properties and then emit queries for the triggering domain names to vulnerable recursive resolvers. It is completely unrealistic to expect that all misconfiguration in DNS zones can be eliminated, particularly when such misconfiguration can be a deliberate and intended outcome as part of an attack on DNS infrastructure. I suspect that this extreme diligence on the part of these resolvers to seek a response is the root of such problems, and resolvers should be more far more willing to abandon their resolution efforts if the response is not readily forthcoming.

Sometimes "I don't know" is the best answer for DNS resolvers!

## Disclaimer

## Author

*Geoff Huston* AM, B.Sc., M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region.

*www.potaroo.net*