Network Working Group R. Gieben
Internet-Draft Google
Intended status: Informational W. Mekking
Expires: May 29, 2014 NLnet Labs
November 25, 2013

Authenticated Denial of Existence in the DNS
draft-gieben-auth-denial-of-existence-dns-05

Abstract

Authenticated denial of existence allows a resolver to validate that a certain domain name does not exist. It is also used to signal that a domain name exists, but does not have the specific RR type you were asking for. When returning a negative DNSSEC response, a name server usually includes up to two NSEC records. With NSEC3 this amount is three.

This document provides additional background commentary and some context for the NSEC and NSEC3 mechanisms used by DNSSEC to provide authenticated denial of existence responses

Table of Contents

1. Introduction

DNSSEC can be somewhat of a complicated matter, and there are certain areas of the specification that are more difficult to comprehend than others. One such area is "authenticated denial of existence".

Denial of existence is a mechanism that informs a resolver that a certain domain name does not exist. It is also used to signal that a domain name exists, but does not have the specific RR type you were asking for.

The first is referred to as an NXDOMAIN (non-existent domain) ([RFC2308] Section 2.1) and the latter a NODATA ([RFC2308] Section 2.2) response. Both are also known as negative responses.

Authenticated denial of existence uses cryptography to sign the negative response. However, if there is no answer, what is it that needs to be signed? To further complicate this matter, there is the desire to pre-generate negative responses that are applicable for all queries for non-existent names in the signed zone. See Section 3 for the details.

In this document, we will explain how authenticated denial of existence works. We begin by explaining the current technique in the DNS and work our way up to DNSSEC. We explain the first steps taken in DNSSEC and describe how NSEC and NSEC3 work. The NXT, NO, NSEC2 and DNSNR records also briefly make their appearance, as they have paved the way for NSEC and NSEC3.

To complete the picture, we also need to explain DNS wildcards as these complicate matters, especially combined with CNAME records.

Note: In this document, domain names in zone file examples will have a trailing dot, in the running text they will not. This text is written for people who have a fair understanding of DNSSEC. The following RFCs are not required reading, but they help in understanding the problem space.

o RFC 5155 [RFC5155] - Hashed Authenticated Denial of Existence;

o RFC 4592 [RFC4592] - The Role of Wildcards in the DNS.

And these provide some general DNSSEC information.

o RFC 4033, RFC 4034, RFC 4035 [RFC4033], [RFC4034], [RFC4035] - DNSSEC Specification;

- RFC 4956 [RFC4956] - DNS Security (DNSSEC) Opt-In. This RFC has experimental status, but is a good read.

These three drafts give some background information on the NSEC3 development.

- The NO record [I-D.ietf-dnsext-not-existing-rr];

- The NSEC2 record [I-D.laurie-dnsext-nsec2v2];

- The DNSNR record [I-D.arends-dnsnr].

2. Denial of Existence

We start with the basics and take a look at NXDOMAIN handling in the DNS. To make it more visible we are going to use a small DNS zone, with 3 names ("example.org", "a.example.org" and "d.example.org") and 3 types (SOA, A and TXT). For brevity, the class is not shown (defaults to IN) and the SOA record is shortened, resulting in the following zone file:

```
example.org.        SOA ( ... )
example.org.        NS  a.example.org.
a.example.org.      A 192.0.2.1
                    TXT "a record"
d.example.org.      A 192.0.2.1
                    TXT "d record"
```

Figure 1: The unsigned "example.org" zone.

2.1. NXDOMAIN Responses

If a resolver asks for the TXT type belonging to "a.example.org" to the name server serving this zone, it sends the following question: "a.example.org TXT"

The name server looks in its zone data and generates an answer. In this case a positive one: "Yes it exists and this is the data", resulting in this reply:

```
;; status: NOERROR, id: 28203

;; ANSWER SECTION:
a.example.org.      TXT "a record"

;; AUTHORITY SECTION:
example.org.        NS a.example.org.
```

The "status: NOERROR" signals that everything is OK, "id" is an integer used to match questions and answers. In the ANSWER section, we find our answer. The AUTHORITY section holds the names of the name servers that have information concerning the "example.org" zone. Note that including this information is optional.

If a resolver asks for "b.example.org TXT" it gets an answer that this name does not exist:

```
;; status: NXDOMAIN, id: 7042

;; AUTHORITY SECTION:
example.org.        SOA ( ... )
```

In this case, we do not get an ANSWER section and the status is set to NXDOMAIN. From this the resolver concludes that "b.example.org" does not exist. The AUTHORITY section holds the SOA record of "example.org" that the resolver can use to cache the negative response.

2.2. NODATA Responses

It is important to realize that NXDOMAIN is not the only type of does-not-exist. A name may exist, but the type you are asking for may not. This occurrence of non-existence is called a NODATA response. Let us ask our name server for "a.example.org AAAA", and look at the answer:

```
;; status: NOERROR, id: 7944

;; AUTHORITY SECTION:
example.org.        SOA ( ... )
```

The status NOERROR shows that the "a.example.org" name exists, but the reply does not contain an ANSWER section. This differentiates a NODATA response from an NXDOMAIN response, the rest of the packet is very similar. The resolver has to put these pieces of information together and conclude that "a.example.org" exists, but it does not have an "AAAA" record.

3. Secure Denial of Existence

The above has to be translated to the security aware world of DNSSEC. But there are a few principles DNSSEC brings to the table:

1. A name server is free to compute the answer and signature(s) on-the-fly, but the protocol is written with a "first sign, then load" attitude in mind. It is rather asymmetrical, but a lot of

the design in DNSSEC stems from fact that you need to accommodate authenticated denial of existence. If the DNS did not have NXDOMAIN, DNSSEC would be a lot simpler, but a lot less useful!

2. The DNS packet header is not signed. This means that a "status: NXDOMAIN" can not be trusted. In fact the entire header may be forged, including the AD bit (AD stands for Authentic Data, see RFC 3655 [RFC3655]), which may give some food for thought;

3. DNS wildcards and CNAME records complicate matters significantly. More about this in later sections (Section 5.3 and Section 5.4).

The first principle implies that all denial of existence answers need to be pre-computed, but it is impossible to pre-compute (all conceivable) non-existence answers.

A generic denial record which can be used in all denial of existence proofs is not an option: such a record is susceptible to replay attacks. When you are querying a name server for any record that actually exists, a man-in-the-middle could replay that generic denial record that is unlimited in its scope and it would be impossible to tell whether the response was genuine or spoofed. In other words, the generic record can be replayed to falsely deny _all_ possible responses.

We could also use the QNAME in the answer and sign that; essentially signing an NXDOMAIN response. While this approach could have worked technically, it is incompatible with off-line signing.

The way this has been solved is by introducing a record that defines an interval between two existing names. Or to put it another way: it defines the holes (non-existing names) in the zone. This record can be signed beforehand and given to the resolver. Appendix A and Appendix B describe on-line signing techniques that are compatible with this scheme.

> Given all these troubles, why didn't the designers of DNSSEC go for the (easy) route and allowed for on-line signing? Well, at that time (pre 2000), on-line signing was not feasible with the then current hardware. Keep in mind that the larger servers get between 2000 and 6000 queries per second (qps), with peaks up to 20,000 qps or more. Scaling signature generation to these kind of levels is always a challenge. Another issue was (and is) key management, for on-line signing to work _each_ authoritative name server needs access to the private key(s). This is considered a security risk. Hence, the protocol required not to rely on on-line signing.

The road to the current solution (NSEC/NSEC3) was long. It started with the NXT (next) record. The NO (not existing) record was introduced, but never made it to RFC. Later on, NXT was superseded by the NSEC (next secure) record. From there it went through NSEC2/DNSNR to finally reach NSEC3 (next secure, version 3) in RFC 5155.

3.1. NXT

The first attempt to specify authenticated denial of existence was NXT (RFC 2535 [RFC2535]). Section 5.1 of that RFC introduces the record:

> "The NXT resource record is used to securely indicate that RRs with an owner name in a certain name interval do not exist in a zone and to indicate what RR types are present for an existing name."

By specifying what you do have, you implicitly tell what you don't have. NXT is superseded by NSEC. In the next section we explain how NSEC (and thus NXT) works.

3.2. NSEC

In RFC 3755 [RFC3755] all the DNSSEC types were given new names, SIG was renamed RRSIG, KEY became DNSKEY and NXT was renamed to NSEC and a minor issue was fixed in the process, namely the type bitmap was redefined to allow more than 127 types to be listed ([RFC2535], Section 5.2).

Just as NXT, NSEC is used to describe an interval between names: it indirectly tells a resolver which names do not exist in a zone.

For this to work, we need our "example.org" zone to be sorted in canonical order ([RFC4034], Section 6.1), and then create the NSECs. We add three NSEC records, one for each name, and each one covers a certain interval. The last NSEC record points back to the first as required by the RFC, and depicted in Figure 2.

1. The first NSEC covers the interval between "example.org" and "a.example.org";

2. The second NSEC covers "a.example.org" to "d.example.org";

3. The third NSEC points back to "example.org", and covers "d.example.org" to "example.org" (i.e. the end of the zone).

As we have defined the intervals and put those in resource records, we now have something that can be signed.

```
                                   example.org
                                      **
                                  +-- ** <--+
                             (1) /  .     .   \ (3)
                                /  .       .   \
                               |  .         .  |
                               v .           . |
                               **    (2)     **
                 a.example.org ** ---------> ** d.example.org
```

Figure 2: The NSEC records of "example.org". The arrows represent NSEC records, starting from the apex.

This signed zone is loaded into the name server. It looks like this:

```
   example.org.        SOA ( ... )
                       DNSKEY ( ... )
                       NS  a.example.org.
                       NSEC a.example.org. NS SOA RRSIG NSEC DNSKEY
                       RRSIG(NS) ( ... )
                       RRSIG(SOA) ( ... )
                       RRSIG(NSEC) ( ... )
                       RRSIG(DNSKEY) ( ... )
   a.example.org.      A 192.0.2.1
                       TXT "a record"
                       NSEC d.example.org. A TXT RRSIG NSEC
                       RRSIG(A) ( ... )
                       RRSIG(TXT) ( ... )
                       RRSIG(NSEC) ( ... )
   d.example.org.      A 192.0.2.1
                       TXT "d record"
                       NSEC example.org. A TXT RRSIG NSEC
                       RRSIG(A) ( ... )
                       RRSIG(TXT) ( ... )
                       RRSIG(NSEC) ( ... )
```

Figure 3: The signed and sorted "example.org" zone with the added NSEC records (and signatures). For brevity, the class is not shown (defaults to IN) and the SOA, DNSKEY and RRSIG records are shortened.

If a DNSSEC aware resolver asks for "b.example.org", it gets back a "status: NXDOMAIN" packet, which by itself is meaningless (remember that the DNS packet header is not signed and thus can be forged). To be able to securely detect that "b" does not exist, there must also be a signed NSEC record which covers the name space where "b" lives. The record:

```
a.example.org.      NSEC d.example.org. A TXT RRSIG NSEC
```

does precisely that: "b" should come after "a", but the next owner name is "d.example.org", so "b" does not exist.

Only by making that calculation, is a resolver able to conclude that the name "b" does not exist. If the signature of the NSEC record is valid, "b" is proven not to exist. We have authenticated denial of existence.

Note that a man-in-the-middle may still replay this NXDOMAIN response when you're querying for, say, "c.example.org". But it would not do any harm since it is provably the proper response to the query. In the future, there may be data published for "c.example.org". Therefore, the RRSIG's RDATA include a validity period (not visible in the zone above), so that an attacker cannot replay this NXDOMAIN response for "c.example.org" forever.

3.3. NODATA Responses

NSEC records are also used in NODATA responses. In that case we need to look more closely at the type bitmap. The type bitmap in an NSEC record tells which types are defined for a name. If we look at the NSEC record of "a.example.org", we see the following types in the bitmap: A, TXT, NSEC and RRSIG. So for the name "a" this indicates we must have an A, TXT, NSEC and RRSIG record in the zone.

With the type bitmap of the NSEC record, a resolver can establish that a name is there, but the type is not. For example, if a resolver asks for "a.example.org AAAA", the reply that comes back is:

```
;; status: NOERROR, id: 44638

;; AUTHORITY SECTION:
example.org.        SOA ( ... )
example.org.        RRSIG(SOA) ( ... )
a.example.org.      NSEC d.example.org. A TXT RRSIG NSEC
a.example.org.      RRSIG(NSEC) ( ... )
```

The resolver should check the AUTHORITY section and conclude that:

(1) "a.example.org" exists (because of the NSEC with that owner name) and;

(2) the type (AAAA) does not as it is not listed in the type bitmap.

The techniques used by NSEC form the basics of authenticated denial of existence in DNSSEC.

3.4. Drawbacks of NSEC

There were two issues with NSEC (and NXT). The first is that it allows for zone walking. NSEC records point from one name to another, in our example: "example.org", points to "a.example.org" which points to "d.example.org" which points back to "example.org". So we can reconstruct the entire "example.org" zone, thus defeating attempts to administratively block zone transfers ([RFC2065] Section 5.5).

The second issue is that when a large, delegation-centric ([RFC5155], Section 1.1), zone deploys DNSSEC, every name in the zone gets an NSEC plus RRSIG. So this leads to a huge increase in the zone size (when signed). This would in turn mean that operators of such zones who are deploying DNSSEC, face up front costs. This could hinder DNSSEC adoption.

These two issues eventually lead to NSEC3 which:

- Adds a way to garble the owner names, thus thwarting zone walking;

- Makes it possible to skip names for the next owner name. This feature is called Opt-Out (See Section 5.1). It means not all names in your zone get an NSEC3 plus ditto signature, making it possible to "grow into" your DNSSEC deployment.

Note that there are other ways to mitigate against zone walking. RFC 4470 [(#RFC4470) prevents zone walking by introducing minimally covering NSEC records. This technique is described in Appendix A.

Before we delve into NSEC3, let us first take a look at its predecessors: NO, NSEC2, and DNSNR.

4. Experimental and Deprecated Mechanisms: NO, NSEC2 and DNSNR

Long before NSEC was defined, the NO record was introduced. It was the first record to use the idea of hashed owner names, to fix the issue of zone walking that was present with the NXT record. It also fixed the type bitmap issue of the NXT record, but not in a space efficient way. At that time (around 2000) zone walking was not considered important enough to warrant the new record. People were also worried that DNSSEC deployment would be hindered by developing an alternate means of denial of existence. Thus the effort was shelved and NXT remained.

When the new DNSSEC specification [RFC4034] was written, people were still not convinced that zone walking was a problem that should be solved. So NSEC saw the light and inherited the two issues from NXT.

Several years after, NSEC2 was introduced as a way to solve the two issues of NSEC. The NSEC2 draft contains the following paragraph:

> "This document proposes an alternate scheme which hides owner names while permitting authenticated denial of existence of non-existent names. The scheme uses two new RR types: NSEC2 and EXIST."

When an authenticated denial of existence scheme starts to talk about EXIST records, it is worth paying extra attention. The EXIST record was defined as a record without RDATA that would be used to signal the presence of a domain name. From the draft:

> "In order to prove the nonexistence of a record that might be covered by a wildcard, it is necessary to prove the existence of its closest encloser. This record does that. Its owner is the closest encloser. It has no RDATA. If there is another RR that proves the existence of the closest encloser, this SHOULD be used instead of an EXIST record."

The introduction of this record led to questions on what wildcards actually mean (especially in the context of DNSSEC). It is probably not a coincidence that "The Role of Wildcards in the Domain Name System" ([RFC4592]) was standardized before NSEC3 was.

NSEC2 solved the zone walking issue by hashing (with SHA1 and a salt) the "next owner name" in the record, thereby making it useless for zone walking. But it did not have Opt-Out.

The DNSNR record was another attempt that used hashed names to foil zone walking and it also introduced the concept of opting out (called "Authoritative Only Flag") which limited the use of DNSNR in delegation-centric zones.

All these proposals didn't make it, but did provide valuable insights. To summarize:

o The NO record introduced hashing, but this idea lingered in the background for a long time;

o The NSEC2 record made it clear that wildcards were not completely understood;

o The DNSNR record used a new flag field in the RDATA to signal Opt-Out;

5. NSEC3

From the experience gained with NSEC2 and DNSNR, NSEC3 was forged. It incorporates both Opt-Out and the hashing of names. NSEC3 solves any issues people might have with NSEC, but it introduces some additional complexity.

NSEC3 did not supersede NSEC, they are both defined for DNSSEC. So DNSSEC is blessed with two different means to perform authenticated denial of existence: NSEC and NSEC3. In NSEC3 every name is hashed, including the owner name. This means that NSEC3 chain is sorted in hash order, instead of canonical order. Because the owner names are hashed, the next owner name for "example.org" is unlikely to be "a.example.org". Because the next owner name is hashed, zone walking becomes more difficult.

To make it even more difficult to retrieve the original names, the hashing can be repeated several times each time taking the previous hash as input. To prevent the reuse of pre-generated hash values between zones a (per zone) salt can also be added. In the NSEC3 for "example.org" we have hashed the names thrice ([RFC5155], Section 5) and use the salt "DEAD". Lets look at typical NSEC3 record:

```
15bg9l6359f5ch23e34ddua6n1rihl9h.example.org. (
   NSEC3 1 0 2 DEAD A6EDKB6V8VL5OL8JNQQLT74QMJ7HEB84
        NS SOA RRSIG DNSKEY NSEC3PARAM )
```

On the first line we see the hashed owner name: "15bg9l6359f5ch23e34ddua6n1rihl9h.example.org", this is the hashed name of the fully qualified domain name (FQDN) "example.org" encoded as Base32 ([RFC4648]). Note that even though we hashed "example.org", the zone's name is added to make it look like a domain name again. In our zone, the basic format is "Base32(SHA1(FQDN)).example.org". The next hashed owner name "A6EDKB6V8VL5OL8JNQQLT74QMJ7HEB84" (line 2) is the hashed version of "d.example.org", represented as Base32. Note that "d.example.org" is used are the next owner name, because in the hash ordering, its hash comes after the hash of the zone's apex. Also note that ".example.org" is not added to the next hashed owner name, as this name always falls in the current zone.

The "1 0 2 DEAD" section of the NSEC3 states:

- o Hash Algorithm = 1 (SHA1, this is the default, no other hash algorithms are currently defined for use in NSEC3);

- o Opt-Out = 0 (disabled);

o Hash Iterations = 2, this yields three iterations, as a zero value is already one iteration;

o Salt = "DEAD".

At the end we see the type bitmap, which is identical to NSEC's bitmap, that lists the types present at the original owner name. Note that the type NSEC3 is absent from the list in the example above. This is due to the fact that the original owner name ("example.org") does not have the NSEC3 type. It only exists for the hashed name.

Names like "1.h.example.org" hash to one label in NSEC3, "1.h.example.org" becomes: "117gercprcjgg8j04ev1ndrk8d1jt14k.example.org" when used as an owner name. This is an important observation. By hashing the names you lose the depth of a zone - hashing introduces a flat space of names, as opposed to NSEC.

The name used above ("1.h.example.org") creates an empty non-terminal. Empty non-terminals are domain names that have no RRs associated with them, and exist only because they have one or more sub-domains that do ([RFC5155], Section 1.3). The record:

```
1.h.example.org.    TXT "1.h record"
```

creates two names:

1. "1.h.example.org" that has the type: TXT;

2. "h.example.org" which has no types. This is the empty non-terminal.

An empty non-terminal will get an NSEC3 record, but not an NSEC record. In Section 5.5 is shown how the resolver uses these NSEC3 records to validate the denial of existence proofs.

Note that NSEC3 might not always be useful. For example, highly structures zones, like the reverse zones ip6.arpa and in-addr.arpa, can be walked even with NSEC3 due to their structure. Also the names in small, trivial zones can be easily guessed. In these cases, it does not help to defend against zone walking, but does add the computational load on authoritative servers and validators.

5.1. Opt-Out

Hashing mitigates the zone walking issue of NSEC. The other issue, the high costs of securing a delegation to an insecure zone, is

tackled with Opt-Out. When using Opt-Out, names that are an insecure delegation (and empty non-terminals that are only derived from insecure delegations) don't require an NSEC3 record. For each insecure delegation, the zone size can be decreased (compared with a fully signed zone without using Opt-Out) with at least two records: one NSEC3 record and one corresponding RRSIG record. If the insecure delegation would introduce empty non-terminals, even more records can be omitted from the zone.

Opt-Out NSEC3 records are not able to prove or deny the existence of the insecure delegations. In other words, those delegation do not benefit from the cryptographic security that DNSSEC provides.

A recently discovered corner case ([RFC5155-errata3441]) shows that not only those delegations remain insecure, also the empty non-terminal space that is derived from those delegations are insecure. Because the names in this empty non-terminal space do exist according to the definition in [RFC4592], the server should respond to queries for these names with a NODATA response. However, the validator requires an NSEC3 record proving the NODATA response ([RFC5155], Section 8.5):

> "The validator MUST verify that an NSEC3 RR that matches QNAME is present and that both the QTYPE and the CNAME type are not set in its Type Bit Maps field."

A way to resolve this contradiction in the specification is to always provide empty non-terminals with an NSEC3 record, even if it is only derived from an insecure delegation.

5.2. Loading an NSEC3 Zone

Whenever an authoritative server receives a query for a non-existing record, it has to hash the incoming query name to determine into which interval between two existing hashes it falls. To do that it needs to know the zone's specific NSEC3 parameters (hash iterations and salt).

One way to learn them is to scan the zone during loading for NSEC3 records and glean the NSEC3 parameters from them. However, it would need to make sure that there is at least one complete set of NSEC3 records for the zone using the same parameters. Therefore, it would need to inspect all NSEC3 records.

A more graceful solution was designed. The solution was to create a new record, NSEC3PARAM, which must be placed at the apex of the zone. Its role is to provide a fixed place where an authoritative name server can directly see the NSEC3 parameters used, and by putting it

in the zone it allows for easy transfer to the secondaries. If NSEC3 were designed in the early days of DNS (+/- 1984) this information would probably have been put in the SOA record.

5.3. Wildcards in the DNS

So far, we have only talked about denial of existence in negative responses. However, denial of existence may also occur in positive responses, i.e., where the ANSWER section of the response is not empty. This can happen because of wildcards.

Wildcards have been part of the DNS since the first DNS RFCs. They allow to define all names for a certain type in one go. In our "example.org" zone we could for instance add a wildcard record:

```
*.example.org.      TXT "wildcard record"
```

For completeness, our (unsigned) zone now looks like this:

```
example.org.        SOA ( ... )
example.org.        NS  a.example.org.
*.example.org.      TXT "wildcard record"
a.example.org.      A 192.0.2.1
                    TXT "a record"
d.example.org.      A 192.0.2.1
                    TXT "d record"
```

Figure 4: The example.org zone with a wildcard record.

If a resolver asks for "z.example.org TXT", the name server will respond with an expanded wildcard, instead of an NXDOMAIN:

```
;; status: NOERROR, id: 13658

;; ANSWER SECTION:
z.example.org.      TXT "wildcard record"
```

Note however that the resolver can not detect that this answer came from a wildcard. It just sees the answer as-is. How will this answer look with DNSSEC?

```
;; status: NOERROR, id: 51790

;; ANSWER SECTION:
z.example.org.      TXT "wildcard record"
z.example.org.      RRSIG(TXT) ( ... )

;; AUTHORITY SECTION:
d.example.org.      NSEC example.org. A TXT RRSIG NSEC
d.example.org.      RRSIG(NSEC) ( ... )
```

Figure 5: A response with an expanded wildcard and with DNSSEC.

The RRSIG of the "z.example.org" TXT record indicates there is a wildcard configured. The RDATA of the signature lists a label count [RFC4034], Section 3.1.3., of two (not visible in the answer above), but the owner name of the signature has three labels. This mismatch indicates there is a wildcard "*.example.org" configured.

An astute reader may notice that it appears as if a "z.example.org" RRSIG(TXT) is created out of thin air. This is not the case. The signature for "z.example.org" does not exist. The signature you are seeing is the one for "*.example.org" which does exist, only the owner name is switched to "z.example.org". So even with wildcards, no signatures have to be created on the fly.

The DNSSEC standard mandates that an NSEC (or NSEC3) is included in such responses. If it wasn't, an attacker could mount a replay attack and poison the cache with false data: Suppose that the resolver has asked for "a.example.org TXT". An attacker could modify the packet in such way that it looks like the response was generated through wildcard expansion, even though there exists a record for "a.example.org TXT".

The tweaking simply consists of adjusting the ANSWER section to:

```
;; status: NOERROR, id: 31827

;; ANSWER SECTION:
a.example.org.      TXT "wildcard record"
a.example.org.      RRSIG(TXT) ( ... )
```

Figure 6: A forged response without the expanded wildcard.

Note the subtle difference from Figure 5 in the owner name. In this response we see a "a.example.org TXT" record, for which a record with different RDATA (See Figure 4) exist in the zone.

Which would be a perfectly valid answer if we would not require the inclusion of an NSEC or NSEC3 record in the wildcard answer response. The resolver believes that "a.example.org TXT" is a wildcard record, and the real record is obscured. This is bad and defeats all the security DNSSEC can deliver. Because of this, the NSEC or NSEC3 must be present.

Another way of putting this is that DNSSEC is there to ensure the name server has followed the steps as outlined in [RFC1034], Section 4.3.2 for looking up names in the zone. It explicitly lists wildcard look up as one of these steps (3c), so with DNSSEC this must be communicated to the resolver: hence the NSEC(3) record.

5.4. CNAME Records

So far, the maximum number of NSEC records a response will have is two: one for the denial of existence and another for the wildcard. We say maximum, because sometimes a single NSEC can prove both. With NSEC3, this is three (as to why, we will explain in the next section).

When we take CNAME wildcard records into account, we can have more NSEC(3) records. For every wildcard expansion, we need to prove that the expansion was allowed. Lets add some CNAME wildcard records to our zone:

```
example.org.        SOA ( ... )
example.org.        NS  a.example.org.
*.example.org.      TXT "wildcard record"
a.example.org.      A 192.0.2.1
                    TXT "a record"
*.a.example.org.    CNAME w.b
*.b.example.org.    CNAME w.c
*.c.example.org.    A 192.0.2.1
d.example.org.      A 192.0.2.1
                    TXT "d record"
w.example.org.      CNAME w.a
```

Figure 7: A wildcard CNAME chain added to the "example.org" zone.

A query for "w.example.org A" will result in the following response:

```
;; status: NOERROR, id: 4307

;; ANSWER SECTION:
w.example.org.      CNAME w.a.example.org.
w.example.org.      RRSIG(CNAME) ( ... )
w.a.example.org.    CNAME w.b.example.org.
w.a.example.org.    RRSIG(CNAME) ( ... )
w.b.example.org.    CNAME w.c.example.org.
w.b.example.org.    RRSIG(CNAME) ( ... )
w.c.example.org.    A 192.0.2.1
w.c.example.org.    RRSIG(A) ( ... )

;; AUTHORITY SECTION:
*.a.example.org.    NSEC *.b.example.org. CNAME RRSIG NSEC
*.a.example.org.    RRSIG(NSEC) ( ... )
*.b.example.org.    NSEC *.c.example.org. CNAME RRSIG NSEC
*.b.example.org.    RRSIG(NSEC) ( ... )
*.c.example.org.    NSEC d.example.org. A RRSIG NSEC
*.c.example.org.    RRSIG(NSEC) ( ... )
```

The NSEC record "*.a.example.org" proves that wildcard expansion to "w.a.example.org" was appropriate: "w.a." falls in the gap "*.a" to "*.b". Similar, the NSEC record "*.b.example.org" proves that there was no direct match for "w.b.example.org" and "*.c.example.org" denies the direct match for "w.c.example.org".

DNAME records and wildcard names should not be used as reiterated in [RFC6672] Section 3.3.

5.5. The Closest Encloser NSEC3 Record

We can have one or more NSEC3 records that deny the existence of the requested name and one NSEC3 record that deny wildcard synthesis. What do we miss?

The short answer is that, due to the hashing in NSEC3 you loose the depth of your zone: Everything is hashed into a flat plane. To make up for this loss of information you need an extra record.

To understand NSEC3, we will need two definitions:

Closest encloser: Introduced in [RFC4592], "The closest encloser is the node in the zone's tree of existing domain names that has the most labels matching the query name (consecutively, counting from the root label downward)." In our example, if the query name is "x.2.example.org" then "example.org" is the "closest encloser";

Next closer name: Introduced in the NSEC3 RFC, this is the closest encloser with one more label added to the left. So if "example.org" is the closest encloser for the query name "x.2.example.org", "2.example.org" is the "next closer name".

An NSEC3 "closest encloser proof" consists of:

1. An NSEC3 record that *matches* the "closest encloser". This means the unhashed owner name of the record is the closest encloser. This bit of information tells a resolver: "The name you are asking for does not exist, the closest I have is this".

2. An NSEC3 record that *covers* the "next closer name". This means it defines an interval in which the "next closer name" falls. This tells the resolver: "The next closer name falls in this interval, and therefore the name in your question does not exist. In fact, the closest encloser is indeed the closest I have".

These two records already deny the existence of the requested name, so we do not need an NSEC3 record that covers the actual queried name: By denying the existence of the next closer name, you also deny the existence of the queried name.

Note that with NSEC, the existence of all empty non-terminals between the two names are denied, hence implicitly contains the closest encloser.

For a given query name, there is one (and only one) place where wildcard expansion is possible. This is the "source of synthesis", and is defined ([RFC4592], Section 2.1.1 and Section 3.3.1) as:

<asterisk label>.<closest encloser>

In other words, to deny wildcard synthesis, the resolver needs to know the hash of the source of synthesis. Since it does not know beforehand what the closest encloser of the query name is, it must be provided in the answer.

Take the following example. We take our zone, and put two TXT records to it. The records added are "1.h.example.org" and "3.3.example.org". It is signed with NSEC3, resulting in the following unsigned zone.

```
                   example.org.        SOA ( ... )
                   example.org.        NS  a.example.org.
                   1.h.example.org.    TXT "1.h record"
                   3.3.example.org.    TXT "3.3 record"
```

Figure 8: The added TXT records in example.org. These records create two empty non-terminals: h.example.org and 3.example.org.

The resolver asks the following: "x.2.example.org TXT". This leads to an NXDOMAIN response from the server, which contains three NSEC3 records. A list of hashed owner names can be found in Appendix C. Also see Figure 9 the numbers in that figure correspond with the following NSEC3 records:

```
15bg9l6359f5ch23e34ddua6n1rihl9h.example.org. (
 NSEC3 1 0 2 DEAD 1AVVQN74SG75UKFVF25DGCETHGQ638EK NS SOA RRSIG
        DNSKEY NSEC3PARAM )

1avvqn74sg75ukfvf25dgcethgq638ek.example.org. (
    NSEC3 1 0 2 DEAD 75B9ID679QQOV6LDFHD8OCSHSSSB6JVQ )

75b9id679qqov6ldfhd8ocshsssb6jvq.example.org. (
 NSEC3 1 0 2 DEAD 8555T7QEGAU7PJTKSNBCHG4TD2M0JNPJ TXT RRSIG )
```

If we would follow the NSEC approach, the resolver is only interested in one thing. Does the hash of "x.2.example.org" fall in any of the intervals of the NSEC3 records it got?

```
                       example.org
                          **
                      +-- ** . . . . . . . . . . .
                 (1) /  . ^ .                    .
                    /  .  |   .                   .
                   |  .   |    .                   .
                   v .    |     .                   .
                   **     | (2)  **                 ++
   h.example.org ** ----+----> ** 3.example.org    ++ 2.example.org
                   .     /        . |                .
                   .    / (5)     . | (3)            .
                   .   /          . |                .
                   .  /           . v                .
 1.h.example.org **             **                 ++
                   ** <--------- ** 3.3.example.org  ++ x.2.example.org
                              (4)
```

Figure 9: x.2.example.org does not exist. The five arrows represent the NSEC3 records, the ones numbered (1), (2) and (3) are the NSEC3s returned in our answer. 2.example.org is covered by (3) and

x.2.example.org is covered by (4).

The hash of "x.2.example.org" is "ndtu6dste50pr4a1f2qvr1v31g00i2i1". Checking this hash on the first NSEC3 yields that it does not fall in between the interval: "15bg9l6359f5ch23e34ddua6n1rihl9h" and "1avvqn74sg75ukfvf25dgcethgq638ek". For the second NSEC3 the answer is also negative: the hash sorts outside the interval described by "1avvqn74sg75ukfvf25dgcethgq638ek" and "75b9id679qqov6ldfhd8ocshsssb6jvq". And the third NSEC3, with interval "75b9id679qqov6ldfhd8ocshsssb6jvq" to "8555t7qegau7pjtksnbchg4td2m0jnpj" also isn't of any help.

What is a resolver to do? It has been given the maximum amount of NSEC3s and they all seem useless.

So this is where the closest encloser proof comes into play. And for the proof to work, the resolver needs to know what the closest encloser is. There must be an existing ancestor in the zone: a name must exist that is shorter than the query name. The resolver keeps hashing increasingly shorter names from the query name until an owner name of an NSEC3 matches. This owner name is the closest encloser.

When the resolver has found the closest encloser, the next step is to construct the next closer name. This is the closest encloser with the last chopped label from query name pre-pended to it: "<last chopped label>.<closest encloser>". The hash of this name should be covered by the interval set in any of the NSEC3 records.

Then the resolver needs to check the presence of a wildcard. It creates the wildcard name by pre-pending the asterisk label to the closest encloser: "*.<closest encloser>", and uses the hash of that.

Going back to our example, the resolver must first detect the NSEC3 that matches the closest encloser. It does this by chopping up the query name, hashing each instance (with the same number of iterations and hash as the zone it is querying) and comparing that to the answers given. So it has the following hashes to work with:

x.2.example.org: "ndtu6dste50pr4a1f2qvr1v31g00i2i1", last chopped label: "<empty>";

2.example.org: "7t70drg4ekc28v93q7gnbleopa7vlp6q", last chopped label: "x";

example.org: "15bg9l6359f5ch23e34ddua6n1rihl9h", last chopped label: "2";

Of these hashes only one matches the owner name of one of the NSEC3

records: "15bg9l6359f5ch23e34ddua6n1rihl9h". This must be the closest encloser (unhashed: "example.org"). That's the main purpose of that NSEC3 record: tell the resolver what the closest encloser is.

When using Opt-Out, it is possible that the actual closest encloser to the QNAME does not have an NSEC3 record. If so, we will have to do with the closest provable encloser, which is the closest enclosing authoritative name that does have a NSEC3 record. In the worst case, this is the NSEC3 record corresponding to the apex, this name must always have an NSEC3 record.

With the closest (provable) encloser, the resolver constructs the next closer, which in this case is: "2.example.org"; "2" is the last label chopped, when "example.org" is the closest encloser. The hash of this name should be covered in any of the other NSEC3s. And it is, "7t70drg4ekc28v93q7gnbleopa7vlp6q" falls in the interval set by: "75b9id679qqov6ldfhd8ocshsssb6jvq" and "8555t7qegau7pjtksnbchg4td2m0jnpj" (this is our second NSEC3).

So what does the resolver learn from this?

- "example.org" exists;

- "2.example.org" does not exist.

And if "2.example.org" does not exist, there is also no direct match for "x.2.example.org". The last step is to deny the existence of the source of synthesis, to prove that no wildcard expansion was possible.

The resolver hashes "*.example.org" to "22670trplhsr72pqqmedltg1kdqeolb7" and checks that it is covered: in this case by the last NSEC3 (see Figure 9), the hash falls in the interval set by "1avvqn74sg75ukfvf25dgcethgq638ek" and "75b9id679qqov6ldfhd8ocshsssb6jvq". This means there is no wildcard record directly below the closest encloser and "x.2.example.org" definitely does not exist.

When we have validated the signatures, we reached our goal: authenticated denial of existence.

5.6. Three To Tango

One extra NSEC3 record plus additional signature may seem a lot just to deny the existence of the wildcard record, but we cannot leave it out. If the standard would not mandate the closest encloser NSEC3 record, but instead required two NSEC3 records: one to deny the query name and one to deny the wildcard record. An attacker could fool the

resolver that the source of synthesis does not exist, while it in fact does.

Suppose the wildcard record does exist, so our unsigned zone looks like this:

```
example.org.        SOA ( ... )
example.org.        NS  a.example.org.
*.example.org.      TXT "wildcard record"
1.h.example.org.    TXT "1.h record"
3.3.example.org.    TXT "3.3 record"
```

The query "x.2.example.org TXT" should now be answered with:

```
x.2.example.org.    TXT "wildcard record"
```

An attacker can deny this wildcard expansion by calculating the hash for the wildcard name "*.2.example.org" and searching for an NSEC3 record that covers that hash. The hash of "*.2.example.org" is "fbq73bfkjlrkdoqs27k5qf81aqqd7hho". Looking through the NSEC3 records in our zone we see that the NSEC3 record of "3.3" covers this hash:

```
8555t7qegau7pjtksnbchg4td2m0jnpj.example.org. (
    NSEC3 1 0 2 DEAD 15BG9L6359F5CH23E34DDUA6N1RIHL9H TXT RRSIG )
```

This record also covers the query name "x.2.example.org" ("ndtu6dste50pr4a1f2qvr1v31g00i2i1").

Now an attacker adds this NSEC3 record to the AUTHORITY section of the reply to deny both "x.2.example.org" and any wildcard expansion. The net result is that the resolver determines that "x.2.example.org" does not exist, while in fact it should have been synthesized via wildcard expansion. With the NSEC3 matching the closest encloser "example.org", the resolver can be sure that the wildcard expansion should occur at "*.example.org" and nowhere else.

Coming back to the original question: why do we need up to three NSEC3 records to deny a requested name? The resolver needs to be explicitly told what the "closest encloser" is and this takes up a full NSEC3 record. Then, the next closer name needs to be covered in an NSEC3 record, and finally an NSEC3 must say something about whether wildcard expansion was possible. That makes three to tango.

6. Security Considerations

DNSSEC does not protect against denial of service attacks, nor does it provide confidentiality. For more general security considerations

related to DNSSEC, please see RFC 4033, RFC 4034, RFC 4035 and RFC 5155 ([RFC4033], [RFC4034], [RFC4035] and [RFC5155]).

These RFCs are concise about why certain design choices have been made in the area of authenticated denial of existence. Implementations that do not correctly handle this aspect of DNSSEC, create a severe hole in the security DNSSEC adds. This is specifically troublesome for secure delegations: If an attacker is able to deny the existence of a DS record, the resolver cannot establish a chain of trust, and the resolver has to fall back to insecure DNS for the remainder of the query resolution.

This document aims to fill this "documentation gap" and provide would-be implementors and other interested parties with enough background knowledge to better understand authenticated denial of existence.

7. IANA Considerations

This document has no actions for IANA.

8. Acknowledgments

This document would not be possible without the help of Ed Lewis, Roy Arends, Wouter Wijngaards, Olaf Kolkman, Carsten Strotmann, Jan-Piet Mens, Peter van Dijk, Marco Davids, Esther Makaay, Antoin Verschuren, Lukas Wunner, Joe Abley, Ralf Weber, Geoff Huston, Dave Lawrence, Tony Finch and Mark Andrews. Also valuable was the source code of Unbound. ("validator/val_nsec3.c") [Unbound].

Extensive feedback for early versions was received from Karst Koymans.

9. References

9.1. Normative References

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

[RFC2065] Eastlake, D. and C. Kaufman, "Domain Name System Security Extensions", RFC 2065, January 1997.

[RFC2308] Andrews, M., "Negative Caching of DNS Queries (DNS NCACHE)",

RFC 2308, March 1998.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, March 2005.

[RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, March 2005.

[RFC4592] Lewis, E., "The Role of Wildcards in the Domain Name System", RFC 4592, July 2006.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5155] Laurie, B., Sisson, G., Arends, R., and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence", RFC 5155, March 2008.

[RFC6672] Rose, S. and W. Wijngaards, "DNAME Redirection in the DNS", RFC 6672, June 2012.

9.2. Informative References

[I-D.arends-dnsnr] Arends, R., "DNSSEC Non-Repudiation Resource Record", draft-arends-dnsnr-00 (work in progress), July 2004.

[I-D.ietf-dnsext-not-existing-rr] Josefsson, S., "Authenticating denial of existence in DNS with

minimum disclosure", draft-ietf-dnsext-not-existing-rr-01 (work in progress), November 2000.

[I-D.laurie-dnsext-nsec2v2] Laurie, B., "DNSSEC NSEC2 Owner and RDATA Format", draft-laurie-dnsext-nsec2v2-00 (work in progress), December 2004.

[RFC2535] Eastlake, D., "Domain Name System Security Extensions", RFC 2535, March 1999.

[RFC3655] Wellington, B. and O. Gudmundsson, "Redefinition of DNS Authenticated Data (AD) bit", RFC 3655, November 2003.

[RFC3755] Weiler, S., "Legacy Resolver Compatibility for Delegation Signer (DS)", RFC 3755, May 2004.

[RFC4470] Weiler, S. and J. Ihren, "Minimally Covering NSEC Records and DNSSEC On-line Signing", RFC 4470, April 2006.

[RFC4956] Arends, R., Kosters, M., and D. Blacka, "DNS Security (DNSSEC) Opt-In", RFC 4956, July 2007.

[RFC5155-errata3441] Lewis, E., "Technical Errata against RFC 5155 (not acknowledged)", January 2013.

[Unbound] NLnet Labs, "Unbound: a validating, recursive, and caching DNS resolver", 2006, <http://unbound.net>.

[phreebird] Kaminsky, D., "Phreebird: a DNSSEC proxy", January 2011, <http://dankaminsky.com/phreebird/>.

Appendix A. On-line Signing: Minimally Covering NSEC Records

An NSEC record lists the next existing name in a zone, and thus makes it trivial to retrieve all the names from the zone. This can also be

done with NSEC3, but an adversary will then retrieve all the hashed names. With DNSSEC on-line signing, zone walking can be prevented by faking the next owner name.

To prevent retrieval of the next owner name with NSEC, a different, non-existing (according to the existence rules in []#RFC4592, Section 2.2) name is used. However, not just any name can be used because a validator may make assumptions on the size of the span the NSEC record covers. The span must be large enough to cover the QNAME, but not too large that it covers existing names.

[RFC4470] introduces a scheme for generating minimally covering NSEC records. These records use a next owner name that is lexically closer to the NSEC owner name than the actual next owner name, ensuring that no existing names are covered. The next owner name can be derived from the QNAME with the use of so-called epsilon functions.

For example, to deny the existence of "b.example.org" in the zone from Section 3.2, the following NSEC record could have been generated:

```
a.example.org.      NSEC c.example.org. RRSIG NSEC
```

This record also proves that "b.example.org" also does not exist, but an adversary _cannot_ use the next owner name in a zone walking attack. Note the type bitmap only has the RRSIG and NSEC set, because [RFC4470] states:

> The generated NSEC record's type bitmap MUST have the RRSIG and NSEC bits set and SHOULD NOT have any other bits set.

This is because the NSEC records may appear at names that did not exist before the zone was signed. In this case however, "a.example.org" exists with other RR types and we could have also set the A and TXT types in the bitmap.

Because DNS ordering is very strict, the span should be shortened to a minimum. In order to do so, the last character in the leftmost label of the NSEC owner name needs to be decremented and the label must be filled with octets of value 255 until the label length reaches the maximum of 63 octets. The next owner name is the QNAME with a leading label with a single null octet added. This gives the following minimally covering record for "b.example.org":

```
a\255\255\255\255\255\255\255\255\255\255\255\255\255\255\255\255\255
 \255\255\255\255\255\255\255\255\255\255\255\255\255\255\255\255\255
 \255\255\255\255\255\255\255\255\255\255\255\255\255\255\255\255\255
 \255\255\255\255\255\255\255\255\255\255\255.example.org. (
   NSEC \000.b.example.org. RRSIG NSEC )
```

Appendix B. On-line Signing: NSEC3 White Lies

The same principle of minimally covering spans can be applied to NSEC3 records. This mechanism has been dubbed "NSEC3 White Lies" when it was implemented in Phreebird [phreebird]. Here, the NSEC3 owner name is the hash of the QNAME minus one and the next owner name is the hash of the QNAME plus one.

The following NSEC3 white lie denies "b.example.org" (recall this hashes to "iuu8l5lmt76jeltp0bir3tmg4u3uu8e7"):

```
iuu8l5lmt76jeltp0bir3tmg4u3uu8e6.example.org. (
   NSEC3 1 0 2 DEAD IUU8L5LMT76JELTP0BIR3TMG4U3UU8E8 )
```

The type bitmap is empty in this case. If the hash of "b.example.org" - 1 is a collision with an existing name, the bitmap should have been filled with the RR types that exist at that name. This record actually denies the existence of the next closer name (which is conveniently "b.example.org"). Of course the NSEC3 records to match the closest encloser and the one to deny the wildcard are still required. These can be generated too:

```
# Matching `example.org`: `15bg9l6359f5ch23e34ddua6n1rihl9h`
15bg9l6359f5ch23e34ddua6n1rihl9h.example.org. (
   NSEC3 1 0 2 DEAD 15BG9L6359F5CH23E34DDUA6N1RIHL9I NS SOA RRSIG
        DNSKEY NSEC3PARAM )

# Covering `*.example.org`: `22670trplhsr72pqqmedltg1kdqeolb7`
22670trplhsr72pqqmedltg1kdqeolb6.example.org.(
   NSEC3 1 0 2 DEAD 22670TRPLHSR72PQQMEDLTG1KDQEOLB8 )
```

Appendix C. List of Hashed Owner Names

The following owner names are used in this document. The origin for these names is "example.org".

Original Name	Hashed Name
"a"	"04sknapca5al7qos3km2l9tl3p5okq4c"
"1.h"	"117gercprcjgg8j04ev1ndrk8d1jt14k"
"@"	"15bg9l6359f5ch23e34ddua6n1rihl9h"
"h"	"1avvqn74sg75ukfvf25dgcethgq638ek"
"*"	"22670trplhsr72pqqmedltg1kdqeolb7"
"3"	"75b9id679qqov6ldfhd8ocshsssb6jvq"
"2"	"7t70drg4ekc28v93q7gnbleopa7vlp6q"
"3.3"	"8555t7qegau7pjtksnbchg4td2m0jnpj"
"d"	"a6edkb6v8vl5ol8jnqqlt74qmj7heb84"
"*.2"	"fbq73bfkjlrkdoqs27k5qf81aqqd7hho"
"b"	"iuu8l5lmt76jeltp0bir3tmg4u3uu8e7"
"x.2"	"ndtu6dste50pr4a1f2qvr1v31g00i2i1"

Table 1: Hashed owner names for "example.org" in hash order.

Appendix D. Changelog

[This section should be removed by the RFC editor before publishing]

D.1. -00

1. Initial document.

D.2. -01

1. Style and language changes;
2. Figure captions;
3. Security considerations added;
4. Fix erroneous NSEC3 RR;
5. Section on CNAMEs added;
6. More detailed text on closest encloser proof.

D.3. -02

1. Lowercase NSEC3 hashed ownernames and add reference to Base32;
2. Process the comments from Joe Abley and Geoff Huston.

* Added section about Opt-Out;

* Move experimental records in their own section;

* Added DNAME reference with respect to wildcards;

* Clarify the difference between the wildcard answers;

* Add more context about the NO record;

* Elaborate more about the EXIST records and its problems;

* Added more text about the NSEC3PARAM records;

* Apply assorted fixes throughout the document;

* Moved table with hashed owner names to appendix.

D.4. -03

1. Changed affiliation for R. Gieben;

2. Some minor updates.

D.5. -04

1. Added NS record in all zone examples;

2. Some tweaks in the text regarding on-line signing;

3. Add more text on a non-working "generic non-existence records".

4. Add appendix on on-line signing;

5. Add text on usefulness of NSEC3.

D.6. -05

1. Minor fixes and adjustments.

Authors' Addresses

R. (Miek) Gieben
Google

Phone:
EMail: miek@google.com
URI:

W. (Matthijs) Mekking
NLnet Labs
Science Park 400
Amsterdam,   1098 XH
NL

Phone:
EMail: matthijs@nlnetlabs.nl
URI: http://www.nlnetlabs.nl/