Network Working Group
Request for Comments: 5011
Category: Standards Track

M. StJohns
Independent

September 2007

Automated Updates of DNS Security (DNSSEC) Trust Anchors

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

This document describes a means for automated, authenticated, and authorized updating of DNSSEC "trust anchors". The method provides protection against N-1 key compromises of N keys in the trust point key set. Based on the trust established by the presence of a current anchor, other anchors may be added at the same place in the hierarchy, and, ultimately, supplant the existing anchor(s).

This mechanism will require changes to resolver management behavior (but not resolver resolution behavior), and the addition of a single flag bit to the DNSKEY record.

Table of Contents

1. Introduction

As part of the reality of fielding DNSSEC (Domain Name System Security Extensions) [RFC4033] [RFC4034] [RFC4035], the community has come to the realization that there will not be one signed name space, but rather islands of signed name spaces each originating from specific points (i.e., 'trust points') in the DNS tree. Each of those islands will be identified by the trust point name, and validated by at least one associated public key. For the purpose of this document, we'll call the association of that name and a particular key a 'trust anchor'. A particular trust point can have more than one key designated as a trust anchor.

For a DNSSEC-aware resolver to validate information in a DNSSEC protected branch of the hierarchy, it must have knowledge of a trust anchor applicable to that branch. It may also have more than one

trust anchor for any given trust point. Under current rules, a chain of trust for DNSSEC-protected data that chains its way back to ANY known trust anchor is considered 'secure'.

Because of the probable balkanization of the DNSSEC tree due to signing voids at key locations, a resolver may need to know literally thousands of trust anchors to perform its duties (e.g., consider an unsigned ".COM"). Requiring the owner of the resolver to manually manage these many relationships is problematic. It's even more problematic when considering the eventual requirement for key replacement/update for a given trust anchor. The mechanism described herein won't help with the initial configuration of the trust anchors in the resolvers, but should make trust point key replacement/rollover more viable.

As mentioned above, this document describes a mechanism whereby a resolver can update the trust anchors for a given trust point, mainly without human intervention at the resolver. There are some corner cases discussed (e.g., multiple key compromise) that may require manual intervention, but they should be few and far between. This document DOES NOT discuss the general problem of the initial configuration of trust anchors for the resolver.

1.1. Compliance Nomenclature

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, [RFC2119].

2. Theory of Operation

The general concept of this mechanism is that existing trust anchors can be used to authenticate new trust anchors at the same point in the DNS hierarchy. When a zone operator adds a new SEP key (i.e., a DNSKEY with the Secure Entry Point bit set) (see [RFC4034], Section 2.1.1) to a trust point DNSKEY RRSet, and when that RRSet is validated by an existing trust anchor, then the resolver can add the new key to its set of valid trust anchors for that trust point.

There are some issues with this approach that need to be mitigated. For example, a compromise of one of the existing keys could allow an attacker to add their own 'valid' data. This implies a need for a method to revoke an existing key regardless of whether or not that key is compromised. As another example, assuming a single key compromise, we need to prevent an attacker from adding a new key and revoking all the other old keys.

2.1. Revocation

Assume two trust anchor keys A and B. Assume that B has been compromised. Without a specific revocation bit, B could invalidate A simply by sending out a signed trust point key set that didn't contain A. To fix this, we add a mechanism that requires knowledge of the private key of a DNSKEY to revoke that DNSKEY.

A key is considered revoked when the resolver sees the key in a self-signed RRSet and the key has the REVOKE bit (see Section 7 below) set to '1'. Once the resolver sees the REVOKE bit, it MUST NOT use this key as a trust anchor or for any other purpose except to validate the RRSIG it signed over the DNSKEY RRSet specifically for the purpose of validating the revocation. Unlike the 'Add' operation below, revocation is immediate and permanent upon receipt of a valid revocation at the resolver.

A self-signed RRSet is a DNSKEY RRSet that contains the specific DNSKEY and for which there is a corresponding validated RRSIG record. It's not a special DNSKEY RRSet, just a way of describing the validation requirements for that RRSet.

N.B.: A DNSKEY with the REVOKE bit set has a different fingerprint than one without the bit set. This affects the matching of a DNSKEY to DS records in the parent [RFC3755], or the fingerprint stored at a resolver used to configure a trust point.

In the given example, the attacker could revoke B because it has knowledge of B's private key, but could not revoke A.

2.2. Add Hold-Down

Assume two trust point keys A and B. Assume that B has been compromised. An attacker could generate and add a new trust anchor key C (by adding C to the DNSKEY RRSet and signing it with B), and then invalidate the compromised key. This would result in both the attacker and owner being able to sign data in the zone and have it accepted as valid by resolvers.

To mitigate but not completely solve this problem, we add a hold-down time to the addition of the trust anchor. When the resolver sees a new SEP key in a validated trust point DNSKEY RRSet, the resolver starts an acceptance timer, and remembers all the keys that validated the RRSet. If the resolver ever sees the DNSKEY RRSet without the new key but validly signed, it stops the acceptance process for that key and resets the acceptance timer. If all of the keys that were

originally used to validate this key are revoked prior to the timer expiring, the resolver stops the acceptance process and resets the timer.

Once the timer expires, the new key will be added as a trust anchor the next time the validated RRSet with the new key is seen at the resolver. The resolver MUST NOT treat the new key as a trust anchor until the hold-down time expires AND it has retrieved and validated a DNSKEY RRSet after the hold-down time that contains the new key.

N.B.: Once the resolver has accepted a key as a trust anchor, the key MUST be considered a valid trust anchor by that resolver until explicitly revoked as described above.

In the given example, the zone owner can recover from a compromise by revoking B and adding a new key D and signing the DNSKEY RRSet with both A and B.

The reason this does not completely solve the problem has to do with the distributed nature of DNS. The resolver only knows what it sees. A determined attacker who holds one compromised key could keep a single resolver from realizing that the key had been compromised by intercepting 'real' data from the originating zone and substituting their own (e.g., using the example, signed only by B). This is no worse than the current situation assuming a compromised key.

2.3. Active Refresh

A resolver that has been configured for an automatic update of keys from a particular trust point MUST query that trust point (e.g., do a lookup for the DNSKEY RRSet and related RRSIG records) no less often than the lesser of 15 days, half the original TTL for the DNSKEY RRSet, or half the RRSIG expiration interval and no more often than once per hour. The expiration interval is the amount of time from when the RRSIG was last retrieved until the expiration time in the RRSIG. That is, queryInterval = MAX(1 hr, MIN (15 days, 1/2*OrigTTL, 1/2*RRSigExpirationInterval))

If the query fails, the resolver MUST repeat the query until satisfied no more often than once an hour and no less often than the lesser of 1 day, 10% of the original TTL, or 10% of the original expiration interval. That is, retryTime = MAX (1 hour, MIN (1 day, .1 * origTTL, .1 * expireInterval)).

2.4. Resolver Parameters

2.4.1. Add Hold-Down Time

The add hold-down time is 30 days or the expiration time of the original TTL of the first trust point DNSKEY RRSet that contained the new key, whichever is greater. This ensures that at least two validated DNSKEY RRSets that contain the new key MUST be seen by the resolver prior to the key's acceptance.

2.4.2. Remove Hold-Down Time

The remove hold-down time is 30 days. This parameter is solely a key management database bookeeping parameter. Failure to remove information about the state of defunct keys from the database will not adversely impact the security of this protocol, but may end up with a database cluttered with obsolete key information.

2.4.3. Minimum Trust Anchors per Trust Point

A compliant resolver MUST be able to manage at least five SEP keys per trust point.

3. Changes to DNSKEY RDATA Wire Format

Bit 8 of the DNSKEY Flags field is designated as the 'REVOKE' flag. If this bit is set to '1', AND the resolver sees an RRSIG(DNSKEY) signed by the associated key, then the resolver MUST consider this key permanently invalid for all purposes except for validating the revocation.

4. State Table

The most important thing to understand is the resolver's view of any key at a trust point. The following state table describes this view at various points in the key's lifetime. The table is a normative part of this specification. The initial state of the key is 'Start'. The resolver's view of the state of the key changes as various events occur.

This is the state of a trust-point key as seen from the resolver. The column on the left indicates the current state. The header at the top shows the next state. The intersection of the two shows the event that will cause the state to transition from the current state to the next.

	NEXT STATE					
FROM	Start	AddPend	Valid	Missing	Revoked	Removed
Start		NewKey				
AddPend	KeyRem		AddTime			
Valid				KeyRem	Revbit	
Missing			KeyPres		Revbit	
Revoked						RemTime
Removed						

State Table

4.1. Events

NewKey The resolver sees a valid DNSKEY RRSet with a new SEP key. That key will become a new trust anchor for the named trust point after it’s been present in the RRSet for at least ’add time’.

KeyPres The key has returned to the valid DNSKEY RRSet.

KeyRem The resolver sees a valid DNSKEY RRSet that does not contain this key.

AddTime The key has been in every valid DNSKEY RRSet seen for at least the ’add time’.

RemTime A revoked key has been missing from the trust-point DNSKEY RRSet for sufficient time to be removed from the trust set.

RevBit The key has appeared in the trust anchor DNSKEY RRSet with its "REVOKED" bit set, and there is an RRSig over the DNSKEY RRSet signed by this key.

4.2. States

Start The key doesn’t yet exist as a trust anchor at the resolver. It may or may not exist at the zone server, but either hasn’t yet been seen at the resolver or was seen but was absent from the last DNSKEY RRSet (e.g., KeyRem event).

AddPend The key has been seen at the resolver, has its 'SEP' bit set, and has been included in a validated DNSKEY RRSet. There is a hold-down time for the key before it can be used as a trust anchor.

Valid The key has been seen at the resolver and has been included in all validated DNSKEY RRSets from the time it was first seen through the hold-down time. It is now valid for verifying RRSets that arrive after the hold-down time. Clarification: The DNSKEY RRSet does not need to be continuously present at the resolver (e.g., its TTL might expire). If the RRSet is seen and is validated (i.e., verifies against an existing trust anchor), this key MUST be in the RRSet, otherwise a 'KeyRem' event is triggered.

Missing This is an abnormal state. The key remains a valid trust-point key, but was not seen at the resolver in the last validated DNSKEY RRSet. This is an abnormal state because the zone operator should be using the REVOKE bit prior to removal.

Revoked This is the state a key moves to once the resolver sees an RRSIG(DNSKEY) signed by this key where that DNSKEY RRSet contains this key with its REVOKE bit set to '1'. Once in this state, this key MUST permanently be considered invalid as a trust anchor.

Removed After a fairly long hold-down time, information about this key may be purged from the resolver. A key in the removed state MUST NOT be considered a valid trust anchor. (Note: this state is more or less equivalent to the "Start" state, except that it's bad practice to re-introduce previously used keys -- think of this as the holding state for all the old keys for which the resolver no longer needs to track state.)

5. Trust Point Deletion

A trust point that has all of its trust anchors revoked is considered deleted and is treated as if the trust point was never configured. If there are no superior configured trust points, data at and below the deleted trust point are considered insecure by the resolver. If there ARE superior configured trust points, data at and below the deleted trust point are evaluated with respect to the superior trust point(s).

Alternately, a trust point that is subordinate to another configured trust point MAY be deleted by a resolver after 180 days, where such a

subordinate trust point validly chains to a superior trust point. The decision to delete the subordinate trust anchor is a local configuration decision. Once the subordinate trust point is deleted, validation of the subordinate zone is dependent on validating the chain of trust to the superior trust point.

6. Scenarios - Informative

The suggested model for operation is to have one active key and one stand-by key at each trust point. The active key will be used to sign the DNSKEY RRSet. The stand-by key will not normally sign this RRSet, but the resolver will accept it as a trust anchor if/when it sees the signature on the trust point DNSKEY RRSet.

Since the stand-by key is not in active signing use, the associated private key may (and should) be provided with additional protections not normally available to a key that must be used frequently (e.g., locked in a safe, split among many parties, etc). Notionally, the stand-by key should be less subject to compromise than an active key, but that will be dependent on operational concerns not addressed here.

6.1. Adding a Trust Anchor

Assume an existing trust anchor key 'A'.

1. Generate a new key pair.

2. Create a DNSKEY record from the key pair and set the SEP and Zone Key bits.

3. Add the DNSKEY to the RRSet.

4. Sign the DNSKEY RRSet ONLY with the existing trust anchor key - 'A'.

5. Wait for various resolvers' timers to go off and for them to retrieve the new DNSKEY RRSet and signatures.

6. The new trust anchor will be populated at the resolvers on the schedule described by the state table and update algorithm -- see Sections 2 and 4 above.

6.2. Deleting a Trust Anchor

Assume existing trust anchors 'A' and 'B' and that you want to revoke and delete 'A'.

1. Set the revocation bit on key 'A'.

2. Sign the DNSKEY RRSet with both 'A' and 'B'. 'A' is now revoked. The operator should include the revoked 'A' in the RRSet for at least the remove hold-down time, but then may remove it from the DNSKEY RRSet.

6.3. Key Roll-Over

Assume existing keys A and B. 'A' is actively in use (i.e. has been signing the DNSKEY RRSet). 'B' was the stand-by key. (i.e. has been in the DNSKEY RRSet and is a valid trust anchor, but wasn't being used to sign the RRSet).

1. Generate a new key pair 'C'.
2. Add 'C' to the DNSKEY RRSet.
3. Set the revocation bit on key 'A'.
4. Sign the RRSet with 'A' and 'B'.

'A' is now revoked, 'B' is now the active key, and 'C' will be the stand-by key once the hold-down expires. The operator should include the revoked 'A' in the RRSet for at least the remove hold-down time, but may then remove it from the DNSKEY RRSet.

6.4. Active Key Compromised

This is the same as the mechanism for Key Roll-Over (Section 6.3) above, assuming 'A' is the active key.

6.5. Stand-by Key Compromised

Using the same assumptions and naming conventions as Key Roll-Over (Section 6.3) above:

1. Generate a new key pair 'C'.
2. Add 'C' to the DNSKEY RRSet.
3. Set the revocation bit on key 'B'.
4. Sign the RRSet with 'A' and 'B'.

'B' is now revoked, 'A' remains the active key, and 'C' will be the stand-by key once the hold-down expires. 'B' should continue to be included in the RRSet for the remove hold-down time.

6.6. Trust Point Deletion

To delete a trust point that is subordinate to another configured trust point (e.g., example.com to .com) requires some juggling of the data. The specific process is:

1. Generate a new DNSKEY and DS record and provide the DS record to the parent along with DS records for the old keys.

2. Once the parent has published the DSs, add the new DNSKEY to the RRSet and revoke ALL of the old keys at the same time, while signing the DNSKEY RRSet with all of the old and new keys.

3. After 30 days, stop publishing the old, revoked keys and remove any corresponding DS records in the parent.

Revoking the old trust-point keys at the same time as adding new keys that chain to a superior trust prevents the resolver from adding the new keys as trust anchors. Adding DS records for the old keys avoids a race condition where either the subordinate zone becomes unsecure (because the trust point was deleted) or becomes bogus (because it didn't chain to the superior zone).

7. IANA Considerations

The IANA has assigned a bit in the DNSKEY flags field (see Section 7 of [RFC4034]) for the REVOKE bit (8).

8. Security Considerations

In addition to the following sections, see also Theory of Operation above (Section 2) and especially Section 2.2 for related discussions.

Security considerations for trust anchor rollover not specific to this protocol are discussed in [RFC4986].

8.1. Key Ownership vs. Acceptance Policy

The reader should note that, while the zone owner is responsible for creating and distributing keys, it's wholly the decision of the resolver owner as to whether to accept such keys for the authentication of the zone information. This implies the decision to update trust-anchor keys based on trusting a current trust-anchor key is also the resolver owner's decision.

The resolver owner (and resolver implementers) MAY choose to permit or prevent key status updates based on this mechanism for specific trust points. If they choose to prevent the automated updates, they will need to establish a mechanism for manual or other out-of-band updates, which are outside the scope of this document.

8.2. Multiple Key Compromise

This scheme permits recovery as long as at least one valid trust-anchor key remains uncompromised, e.g., if there are three keys, you can recover if two of them are compromised. The zone owner should determine their own level of comfort with respect to the number of active, valid trust anchors in a zone and should be prepared to implement recovery procedures once they detect a compromise. A manual or other out-of-band update of all resolvers will be required if all trust-anchor keys at a trust point are compromised.

8.3. Dynamic Updates

Allowing a resolver to update its trust anchor set based on in-band key information is potentially less secure than a manual process. However, given the nature of the DNS, the number of resolvers that would require update if a trust anchor key were compromised, and the lack of a standard management framework for DNS, this approach is no worse than the existing situation.

9. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

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

[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.

10. Informative References

[RFC4986] Eland, H., Mundy, R., Crocker, S., and S. Krishnaswamy, "Requirements Related to DNS Security (DNSSEC) Trust Anchor Rollover", RFC 4986, August 2007.

Author's Address

Michael StJohns
Independent

EMail: mstjohns@comcast.net

Full Copyright Statement

Intellectual Property