CONTRACTOR'S QUARTERLY PROGRESS, STATUS AND MANAGEMENT REPORT

1 January - 31 March 2011

DOMAIN NAME SYSTEM SECURITY (DNSSEC) DEPLOYMENT COORDINATION

Contract No: FA 8750-10-C-0020

Data Item A001, CLIN 0002

Submitted by:

Shinkuro, Inc.
Bethesda, MD

Stephen D. Crocker
Principal Investigator

Jeffrey Dewhurst
Financial and Contract Administration

April 2011

Progress Against Planned Objectives (3.1.1)

- Continued monitoring the deployment of DNSSEC among TLDs, and compiling the data in a database with a map creating function for use by all Coordination Team members in presentations and workshops.
- Coordinated the upcoming DNSSEC day-long presentation and convention presence at the next FOSE Meeting. Contacted and obtained commitments from speakers, planned agenda, and negotiated contract with the event organizers.

Technical Accomplishments This Period (3.1.2)

- Continued acquiring and processing data from registries (.org, etc.) looking for DNSSEC uptake and usage patterns

Improvements to Prototypes This Period (3.1.3)

None this period

Deliverables This Period (3.1.5)

None this period

Publications This Period (3.1.7)

- Technical Information Report, DNSSEC Timing Model, Steve Crocker, Olafur Gudmundsson, Andrew Sullivan, Presented at SATIN Conference, April 4, 5, Teddington, UK
- Technical Information Report: Protocols for TLD & Registrar Adoption, Observing Validation in the Wild, Steve Crocker, Olafur Gudmundsson, Presented at SATIN Conference, April 4, 5, Teddington, UK
- Technical Information Report: Initial best practice proposals for registrar transfer, name server changes, and unanticipated effects of caching, Steve Crocker, Olafur Gudmundsson, Andrew Sullivan, Presented at SATIN Conference, April 4, 5, Teddington, UK

Meetings and Presentations This Period (3.1.8)

- DNSSEC Deployment Coordination Meetings were held at Shinkuro's offices on January 4th, February 1st, and March 8th.
- Steve Crocker spoke at a UNESCO meeting in Paris on Feb 3 - 4th
- Shinkuro assisted in the preparation and presentation of a DNSSEC workshop at the ICANN meeting in Silicon Valley, California March 13 - 18.
- Shinkuro assisted in the preparation and presentation of a DNSSEC workshop at the IETF meeting in Prague, Czech Republic, March 27 – April 1st.

Issues or Concerns (3.1.9)

None at this time.

Planned Activities (3.2.1), Information Covering the Next Three Months

- Coordination meetings are scheduled for April 11, May 10, and June 7 at Shinkuro.
- Steve Crocker is speaking at the SATIN meeting in Teddington, UK on April 4 - 5
- The next ICANN meeting will take place in Singapore on June 19 - June 24, and a DNSSEC workshop is scheduled. Steve Crocker of Shinkuro will attend.
- The FOSE Conference and Exposition, Washington DC Convention Center, July 19 - 21

Attachments follow:

1.) Technical Information Report, DNSSEC Timing Model, Steve Crocker, Olafur Gudmundsson, Andrew Sullivan, Presented at SATIN Conference, April 4, 5, Teddington, UK
2.) Technical Information Report: Protocols for TLD & Registrar Adoption, Observing Validation in the Wild, Steve Crocker, Olafur Gudmundsson, Presented at SATIN Conference, April 4, 5, Teddington, UK
3.) Technical Information Report: Initial best practice proposals for registrar transfer, name server changes, and unanticipated effects of caching, Steve Crocker, Olafur Gudmundsson, Andrew Sullivan, Presented at SATIN Conference, April 4, 5, Teddington, UK

DOMAIN NAME SYSTEM SECURITY (DNSSEC) DEPLOYMENT COORDINATION

Contract No: FA 8750-10-C-0020

Data Item A003, CLIN 0002

Technical Information Report
DNSSEC Timing Model

Submitted by:
Shinkuro, Inc. Bethesda, MD

Stephen D. Crocker
Principal Investigator

Jeffrey Dewhurst
Financial and Contract Administration

February 2011

Getting Around and Being Seen: Timing DNS Changes with DNSSEC

March 9, 2011

Ólafur Guðmundsson
Shinkuro, Inc.
5110 Edgemoor Lane
Bethesda, MD 20814
U.S.A
<ogud@shinkuro.com>

Stephen D. Crocker
Shinkuro, Inc.
5110 Edgemoor Lane
Bethesda, MD 20814
U.S.A.
<steve@shinkuro.com>

Andrew Sullivan
Shinkuro, Inc.
5110 Edgemoor Lane
Bethesda, MD 20814
U.S.A.
<ajs@shinkuro.com>

Abstract

DNS operation has always been sensitive to the timing of changes to zones, because of the distributed nature of the system. The sensitivity is more acute in the presence of the DNS Security Extensions. To make clearer the origins and nature of the sensitivity, we first distinguish among three different kinds of distribution, and outline how the different kinds of distribution lead to two different periods: the zone distribution period, and the visibility period. With these notions in hand, we step through two common tasks in the operation of a DNSSEC-enabled zone.

1 Introduction

This paper is about timing issues in Domain Name System (henceforth, "DNS") administration and operation, with particular attention to the complexities added when using the DNS Security Extensions (henceforth "DNSSEC"). Some time-related parameters, such as time to live (TTL) and signature validity period, appear explicitly in various DNS records. Other time parameters, such as signature re-signing period, are set internally within the operation of a zone. The operating environment determines other parameters, such as the propagation time for changes. For correct operation of a zone, these timing parameters must adhere to certain constraints. Moreover, these timing parameters govern how quickly changes in the configuration may be made and how frequently the same data has to be resent to secondary servers and caches. This paper outlines the basis for the different parameters in the distributed nature of the DNS. It exposes two fundamental timing considerations that can be used as the basis of sound zone administration. Finally, it steps through two common DNSSEC operation scenarios, in order to illustrate how the two timing considerations make the zone appear from various places on the Internet.

DNSSEC introduces new set of possible failure modes for DNS operation, related to expriry of signatures, failure in trust delegation, and other misconfigurations. There has been some work done in providing guidance for DNSSEC operation [7] but these guidelines have been more motivated by cryptographic considerations and less by DNS worst-case behavior. In the case of failures rooted in DNSSEC, only those sites that perform DNSSEC validation will observe the errors; the rest of the world would not detect anything wrong. These failures can be avoided by careful selection of paramters, proper procedures, testing, and the

use of good tools. In what follows, we outline the interconnections among the various timings in the hope of providing some guidance to administrators who are selecting the parameters, and to tool designers hoping to help with the deploment of DNSSEC.

We begin this paper in Section 2 with a brief review of the structure of DNS and the operation of a zone. We explore two basic periods that occur during zone data changes: the distribution of the change, and the visibility of the change. We continue in Section 3 with an inventory of traditional DNS timing parameters and their necessary relationships to each other. We proceed to discuss the important timing parameters relevant to DNSSEC. Finally, in Section 4, we cover timing considerations for some operational scenarios.

Although we review some of the key DNS terms and principles, the reader will find an elementary familiarity with the DNS to be necessary for understanding what follows.

2 Background

The DNS is a distributed, loosely-coherent database used (among other things) to map names of Internet hosts to their Internet Protocol numbers. DNS data is arranged in a tree structure with a common root. It is divided up into administrative segments called zones. From the common root, DNS is distributed both in terms of the operation of parts of the tree (by using zones), and in terms of how the data is delivered to clients requesting it.

Fundamentally, a DNS answer is provided by a server acting in response to a DNS resolver, requesting the value of a `NAME` and `TYPE`[1]. The `NAME` is expressed (in "presentation form") as strings (called labels) separated by a dot ("."). So, for instance, the resolver asks for a `TYPE A` record for `NAME an.example.com`. The server responds to the resolver with an answer containing the appropriate resource record (or "RR") answering the request – in this case, the IP address of the host `an.example.com`.

Since the DNS is a distributed database, the answers a requester might get from the system are determined by the state of more than one machine in the system. Since there are several independent systems involved, the answers a requester might get are partially determined by the state of each of those independent systems. This means that, for DNS operators, it is important to address the timing of changes to data in the DNS, taking into consideration the possible state of other nodes in the DNS (including clients). The considerations get more complicated with the advent of DNSSEC. These timings are governed by a number of parameters that may be adjusted by the zone operator.

2.1 DNS as a distributed database

The DNS is distributed in three ways, which we may call administrative distribution, distribution of publication, and distribution of response. The entire database is broken up into administrative sections called zones. Each zone has its own administrator, which is why the domains `usenix.org` and `shinkuro.com` can be operated completely independently from one another. Parent zones delegate portions of their namespace across a zone cut to a child zone. Administrative distribution makes the operation of the global database much easier than it would be were it to require complete centralization, but it comes at the cost of the distribution of authority. Part of what DNSSEC does is provide a mechanism to ensure that responders to queries on either side of the zone cut do in fact have the authority for the zone for which they claim to respond. There is an additional cost, which is that the servers on the parent side of the zone cut, and the servers on the child side of the zone cut, have to co-ordinate the actions that need to take place on both sides of the cut.

In order to make DNS more resilient to failure, more than one server may be authoritative for a zone. Such distribution of publication means that a single failed server does not make the zone it is serving unavailable. Distribution of publication comes at the cost that data available at one server is not thereby

[1] Strictly, it is a triplet of `NAME`, `CLASS`, and `TYPE`, but in practice the `CLASS` is always "`IN`".

automatically and instantaneously available at every other authoritative server for that zone. The time that it takes between when the data source for the zone has the data and when all authoritative servers have the data we call the zone distribution time; we cover this concept in detail in Section 3.1.

The DNS assumes that certain records are needed frequently, and that those records are likely to be needed repeatedly in the same areas of the Internet. To avoid having the same record served over and over again by a zone's authoritative servers, DNS provides a caching mechanism. When an end node looks up a DNS record, it usually does so with its stub resolver; the stub resolver contacts a full-service resolver. The full-service resolver looks up the record on behalf of the stub, returns the answer to the stub, and remembers the answer for a period of time (established by the authoritative server) called the time to live or "TTL". The full-service resolver can re-use the cached answer for any other identical questions it receives during the TTL. This distributes the ability to respond to queries among several different DNS servers on the Internet, only some of which hold the authoritative data for the zone. The time that it takes between the end of the zone distribution time and when every querying agent on the Internet will certainly receive that new data in answer to a query we call the visibility time.

2.2 Types of data availability defined

No matter what the distribution mechanism for DNS data is, the three kinds of distribution mean that there is a gap in time between when data arrives at the data source for a zone, and when that data may be assumed to be in use on the Internet. From the zone operator's point of view, the time gap is important because the operator must take action some time in advance of when he or she wants a state of affairs to hold in the global DNS.

In general, we may say that data is available in the DNS just in case it is available via an explicit query from at least one name server. That data is merely available, however, is not enough. It may be available at some or all of the authoritative name servers for the zone. It may also be more or less available from any random place on the Internet.

Data in the DNS travels in sets of resource records, or RRSets. It is useful to imagine two RRSets for the same name (the RNAME), before and after a change. We can call these $RRSet'$ and $RRSet''$. The exception to this is in the distribution of data between authoritative name severs: in that case, entire versions of the zone appear in the slave name server at one time, instead of one RRSet at a time.

2.2.1 Data propagation

Data is either deleted from or inserted to the DNS. An update to a single RR in the RRSet is always treated as the removal of that record combined (atomically) with the insertion of a different RR for the same RNAME. Imagine an administrator who wants to replace $RRSet'$ with $RRSet''$. It is possible for the administrator to add $RRSet''$ via a system that is not actually available to most other systems on the Internet. In any case, $RRSet''$ gets into a system that is the source for other DNS servers. This is usually called the primary or master server. The primary server may act as a nameserver on the global Internet (as part of the NS set for the zone in question). It usually acts as the source for data for other nameservers; these other name servers are called secondary or slave servers. The master-slave relationship may be repeated such that a server may act as a slave to one server and as a master for another server. In practice, slaves are often configured to fetch their data from the master server or from one or more other slaves, in order to avoid failure in case the connection to the master is unavailable.

The period from when the operator first introduces $RRSet''$ until $RRSet''$ is available from every authoritative server (and $RRSet'$ is no longer available from them) we call the zone distribution period. During that interval, queries to different authoritative name servers may (and unless there is a "hidden master", will) yield a different RRSet for the same RNAME. At the end of the period, in principle every query to every

authoritative server for a zone will yield $RRSet''$. Another way to say this is that, at the end of the propagation period, every authoritative name server will have reached the same point in the zone's SOA serial number progression (explained in Section 3.1.1) . In rapidly-changing zones, of course, there is always a new propagation period underway, so it may not be possible to see that every authoritative name server received $RRSet''$; it is easy to see whether all the name servers have reached (or passed) a given point in the SOA serial number progression.

2.2.2 Data visibility

Because of caches, the answer to a query of the DNS is not only dependent on zone distribution. The time between the end of the zone distribution time and the time at which any random query from anywhere on the Internet will receive $RRSet''$ we call the visibility time. There are two slightly different visibility times.

When completely new data enters the zone, we can think of it as a case where $RRSet'$ is empty, and $RRSet''$ is not empty. In this case, the visibility time is the time of the negative TTL on the zone authoritative for that RNAME. We call this the insert visibility time. The negative TTL is discussed in detail in Section 3.1.1.

We can contrast insert visibility with delete visibility time. If $RRSet'$ is not empty, then what is necessary is that the copy of $RRSet'$ in caches on the Internet expire, and that $RRSet''$ be fetched from the authoritative servers. It does not matter whether $RRSet''$ is new data, or is an empty set: the key is that $RRSet'$ needs to be deleted everywhere before it will stop occluding the visibility of $RRSet''$. The delete visibility time is at least as long as the TTL on $RRSet'$. We cover this parameter in Section 3.1.1.

2.3 How DNSSEC validation works: the condensed version

DNSSEC ([3][5][4][9]) is a set of extensions to the DNS to provide a mechanism for assuring the authenticity DNS answers. The assurance within a zone is provided by signatures on each RRSet. The signatures can be cryptographically validated using public key cryptography. These signatures (the RRSIG records) provide the assurance that the signed RRSets in the answer have not been altered in their travels from the authoritative server. The assurance is built in a chain of trust, starting with a trust anchor – a key configured locally in the validator of the data.

It is of course impractical to configure the public key of every Internet zone in every validator. The administrative distribution of the DNS is used to make this problem tractable. If a validator has a trust anchor for a given zone, then zones under the trusted zone may also be trusted, so long as there is no break in the chain of trust. In the parent zone, a delegation signer (DS) record, signed by the parent, provides proof that the parent zone knows about the corresponding DNSKEY record found in the child zone. By matching the (signed) DS record in the parent to the (signed) DNSKEY record in the child, a validator can prove that the servers on both sides of the zone cut are legitimate, thereby preserving the ability to validate data from each zone without configuring the public key for every zone. This approach is analogous to following a chain of name server delegations, with the difference that the NS records on both sides of the zone cut are the same; whereas the DS and DNSKEY records are different types.

3 List of relevant parameters

A number of different parameters control the distribution and visibility times for different kinds of DNS data. Below we outline the terms relevant later in this paper, and some important foundation terms. We use symbolic names in the following sections in order to provide a convenient notation for calculation.

In all of what follows, we use unmarked names, such as K, to denote any instance of a given parameter. We use names marked with the "prime" mark ($'$) to denote the starting instance of that parameter or variable.

Symbol	Name	Where set
Z_{ser}	Zone SOA serial	SOA RR 1st numeric value
Z_e	Zone expiry	SOA RR 4th numeric value
Z_f	Zone refresh	SOA RR 2nd numeric value
Z_t	Zone retry	SOA RR 3rd numeric value

Table 1: Protocol-defined zone distribution parameters

So, for instance, K' denotes any key in the DNSKEY set before any action is taken by the zone administrator. We use names marked with a "double-prime" mark ($''$) to denote the final instance of that parameter or variable. To continue the example, K'' is any key in the DNSKEY set, that was not in K', after some action is taken by the administrator.

3.1 Parameters controlling whole zone distribution and availability

In this section, we discuss parameters that influence zone distribution. The primary concern is to establish the outer limit of the zone distribution time, Z_d. Z_d determines the earliest start time of visibility. In order to define precisely how we arrive at Z_d, we need to consider a number of parameters. Nevertheless, it is important to keep in mind that Z_d is a more or less fixed window of time for a given zone; it is likely to be very different for different zones.

3.1.1 Protocol-defined parameters

Some of the parameters that control zone distribution are determined by the DNS protocol itself. They are all parts of the SOA RR, and defined in [10]. These are outlined in Table 1.

Z_{ser} is a serial number that uniquely identifies one version of the zone. It is used to indicate that there is new data in the zone, or that some data has been removed. Under the protocol, $RRSet'$ and $RRSet''$ should never be in the zone with the same serial number. The serial number change is the mechanism by which different authority servers detect that they have different versions of the zone.

Z_e is the zone expiry time. It establishes an upper limit on the interval after which zone data may no longer be served as authoritative, starting from the time the secondary receives the data. A secondary will stop answering queries for a zone this long after retrieving it, unless it has been able to refresh in the interval.[2]

Z_f denotes the zone refresh time. The refresh time is the interval between when an authority server receives a copy of the zone, and when it should next check whether it has the latest copy of the zone. Obviously, if Z_f is greater than Z_e, then the server will stop serving the zone before it checks to see whether the local copy is up to date.

Z_t is the retry time, which is an interval an authority server should wait after a refresh failure before trying to refresh again.

[2]There are some complexities to the way the expiry time works when slaves get data from one another. They are discussed in [2].

Symbol	Name	Relevant factors
Z_{pl}	Zone propagation length	Number of nameservers in distribution chain
Z_{tnum}	Number of zone retries	Network reliability and server load
Z_{xfr}	Zone transfer time	Network and server speed, and zone size
$O(Z)$	Zone size	Number of RRSets in the zone

Table 2: Distribution parameters that are the consequence of operation

3.1.2 Parameters that arise from the operational environment

Some of the parameters affecting zone distribution time are just a consequence of the operation in question. They are outlined in Table 2.

Z_{pl} is the graph distance from the primary master nameserver to the furthest authoritative name server in the set. It is determined by the operator's name service design. Z_{tnum} is the number of retries a slave makes before it successfully transfers the zone. In most operations, this is usually zero; but under heavy load, or when network failures occur, this number becomes a factor in zone distribution. $Z_{xfr}$is the time it takes to copy the zone data from the master server to a slave. Z_{xfr} is partly determined by the size of the zone itself, $O(Z)$.[3]

3.1.3 Zone distribution time defined

The time for zone distribution can be determined using the foregoing parameters. It is simply the time that the protocol settings determine zone distribution will take, plus any important operational factors.

In principle, the lower bound of Z_d could be zero, although in practice it is almost always longer than that (because there is almost always a gap between when data is first injected to one server, and when it is available at all the others. It is especially bad practice to operate with only one name server.)

Formally, we can calculate Z_d as

$$Z_d = (Z_f \times Z_{pl}) + (Z_{xfr} \times Z_{pl}) + (Z_t \times Z_{tnum})$$

For practical purposes, $(Z_t \times Z_{tnum})$ is usually 0, because retries should be a rare occurrence. (If they are not a rare occurrence, that is an indication that something is probably wrong.) Similarly, in many common deployments, the value of Z_{pl} is 1, because the deployment has one master server, and all slaves receiving changes directly from the master. For those environments, we can simplify the calculation to

$$Z_d \approx Z_f + Z_{xfr}$$

In such environments, and with good monitoring of the value of Z_{ser} on the master and all the slaves, the second form will be more convenient to use for calculation.

3.1.4 What about IXFR and Notify?

One might at this juncture wonder why we have ignored the great time savers of the DNS, IXFR and Notify. Notify [14] provides a mechanism for master servers to tell slaves that there is a new copy of the zone ready

[3]It might be argued that the time to load the zone after transfer is another factor. We have ignored it because it is mostly an artifact of implementation, but in some environments it can be a significant cost, and should not be overlooked.

Symbol	Name	Where determined
T	TTL (time to live) on any RRSet	TTL field in each RR
T_{rrsig}	TTL on the RRSIG	TTL field from RRSIG in RRSet
T_{dnskey}	TTL on the DNSKEY RRSet	TTL field in DNSKEY RRSet
T_{ds}	TTL on the DS RRSet	TTL field from DS RRSet
T_{neg}	Negative TTL	SOA RR 5th numeric value
$\max(T)$	Maximum TTL in zone	Largest TTL value in the zone

Table 3: Parameters controlling visibility

to be picked up. In a rapidly-changing zone, this means that the slaves do not have to wait Z_f before getting the changes. IXFR [12] provides a way to send just the changes to the slaves, avoiding the need to transfer the entire (possibly large) zone to a slave that already has most of the data in it.

It is true that IXFR and Notify can help reduce the effective length of Z_d in practice. For the purposes of safe operation, however, we prefer to base our calculations on the worst case. Under load (or, depending on the implementation in question, at any time), IXFR is allowed to fall back to AXFR, so a complete zone transfer is possible even when using IXFR. Servers may drop Notify messages when under heavy query load; but no server is allowed to ignore Z_f or Z_e. Of course, with careful monitoring of Z_{ser} at all authoritative servers, it is possible to establish that Z_d in fact happens sooner than the equations in Section 3.1.3 would indicate.

3.2 Parameters controlling visibility of individual RRSets

Recall that whereas zone distribution works in terms of versions of the zone, data visibility really functions at the level of the RRSet because caches do not get whole copies of zones. The parameters in Table 3 are always determined by the data in the authoritative server, although sometimes only implicitly.

T is the time to live, or TTL, on any RRSet. The TTL establishes how long a cache may keep an RRSet, and continue to serve it in answer to other queries, without checking the authority server to make sure the data is up to date. There is one slightly odd thing about the TTL: it is a field on each resource record, rather than a field on the RRSet, even though it governs caching of the RRSet. Perhaps unfortunately, there is no data in an RRSet that is not carried in an individual RR in the set. For this reason, the TTLs on every RR in the RRSet are required to be the same by [6]. Failure to follow that rule causes serious operational effects in a traditional DNS operation; failure to follow the rule in a DNSSEC context almost certainly leads to validation failures.

The TTL on certain RRs or RRSets is important in itself, so we give them special names in the table. It is important to remember that T_{rrsig} is actually the TTL of the covered RRSet; we are not aware of any implementations where one may set the TTL on an RRSIG independently of the RRSet it covers. In principle, the TTLs on DNSKEY and DS records can be set independently of other TTLs in the zone; we discuss some practical limitations on that freedom in Section 4.

T_{neg}, the negative TTL, is defined in the SOA record for the whole zone. The details of how it works are in [1]. What is important for our purposes is that T_{neg} establishes a period of time in which negative answers, which tell the resolver that the requested RRSet does not exist, may be reused for queries for the

Symbol	Name	How determined
K	DNSKEY	Any DNSKEY record in the zone
ZSK	Zone-signing key	No SEP bit on DNSKEY record
KSK	Key-signing key	SEP bit on DNSKEY record
K_l	Key lifetime	Is the key available in the zone?
K_a	Key activity period	Is the key used to sign anything?
DS_K	DS record for K	The DS record that corresponds to the named key

Table 4: Key data

same RRSet.

When actions require operation on the whole zone, one still needs to consider the TTLs. In that case, the value that one uses for the TTL is $\max(T)$ for the entire zone.

3.3 Additional lifetime parameters for DNSSEC operation

DNSSEC introduces additional parameters to the lifetime of certain RRs in the zone. Also, there are some policy parameters on DNSSEC data that are nowhere reflected in zone data, but that nevertheless constrain the lifetime of the data.

There are two critical types of DNSSEC data to consider. The first of these is key data, listed in Table 4. We use the name K to refer to any arbitrary key record in the zone. Key records come in sets of DNSKEY RRSets. Not all the keys perform the same function. For the purposes of use, [8] (and, following it, [3]) defines a secure entry point (or "SEP") bit on DNSKEY records. The SEP bit indicates that the key should be used as a secure entry point; this key is what is often called the key-signing key or KSK. It is used to generate signatures over a different key, the zone-signing key or ZSK. Keys performing these actions are named KSK and ZSK, respectively, in Table 4. We use these names only when it is important to differentiate between a particular ZSK and a particular KSK. In a secured delegation, the KSK corresponds to the DS record that appears in the parent zone. Even though the different keys can be understood to be performing different functions, they are not actually distinguished by the protocol. The SEP bit is merely a hint, and all K in a zone are returned as part of the DNSKEY RRSet. Validators may try any of the keys when attempting validation.

Keys have a lifetime, K_l, which can be determined by whether K is available in the zone[4]. If the key is actually used to generate a signature, then it is in its active period K_a. People used to other security protocols, where keys have a lifetime associated with them in the protocol, are often surprised that DNSKEYs do not have an explicit lifetime or activity period. The lifetime and activity period are entirely a consequence of the operation of the zone.

The other type of DNSSEC data to consider is RRSIG data, covered in Table 5. RRSIGs are the signatures that cover each RRSet in a signed zone. It is these signatures that are checked during DNSSEC validation. RRSIGs are generated with at least one of the ZSKs for the zone.

[4]For a different account of key lifetime and activity, see [11]

Symbol	Name	How determined
$S_K(RRSet)$	The RRSIG S with K on $RRSet$	The RRSIG that covers the RRSet in question, using the key noted
S_e	RRSIG expiry	RRSIG record field 5
S_i	RRSIG inception	RRSIG record field 6
S_l	RRSIG lifetime	Period between R_i and R_e
S_f	RRSIG refresh	Point in time at which operator generates a new RRSIG
$R[K]$	Signed RRSet from K	The signed RRSet (including the RRSIG) generated with K
$P_{R[K]}$	RRSet signing time	Period of time it takes to sign one RRSet
$P_{Z[K]}$	Zone signing time	Period of time it takes to completely sign a zone
$P_{rm(S_K)}$	Signature removal period	Period of time it takes to remove a single RRSIG

Table 5: RRSIG data

An important difference between most of the traditional in-zone time parameters (such as TTL) and the DNSSEC parameters is that some of the latter are expressed as absolute time values, instead of as intervals. Since the absolute time can end when a record is cached (because of its TTL), one must never assume that the absolute time on the record is the only relevant time to the life of that record. It is also critical to ensure that any DNS systems, including servers and validators, always have the correct time.

For signatures over an arbitrary RRSet, we use the notation $S\,(RRSet)$. We use a similar notation for RRSIG parameters, so S_f is the RRSIG refresh time (and $S_f(RRSet)$ is the refresh time for the RRSIG over $RRSet$). This can be slightly confusing, because a change in the RRSIGs on an RRSet in fact means that the RRSet moves from $RRset'$ to $RRSet''$. Even though an RRSIG covers an RRSet, the RRSIG is in fact normally delivered with the answer to a query for that RRSet. From an observer's point of view the RRSIG usually appears to be an integral part of the RRSet. Therefore, we denote the signed RRSet that includes $S(RRSet)$ (and what one would normally receive in response to a DNSSEC-enabled query for a given RNAME) as $R[K]$, where K is the key used to generate the RRSIG. The time it takes to generate $R[K]$ is $P_{R[K]}$. By extension, the time to sign the entire zone is $P_{Z[K]}$. Note that in some implementations, $P_{Z[K]}$ may not be equivalent to the sum of $P_{R[K]}$ for all R.

4 Operational scenario timing calculations

As outlined above, we call the time at which an operator may assume that a given $RRSet$ is available in response to queries from anywhere on the Internet the visibility time. "Visible" means "visible from anywhere", rather than "visible from somewhere". Until it can be guaranteed that any node on the Internet would see the data, it is not yet really visible.

We can calculate the visibility time for different operational cases. These cases are limits to the zone operator's freedom of action (assuming the zone operator never wants a query to get erroneous results). We begin by discussing the scenarios for replacing $RRSet'$ with $RRSet''$ without DNSSEC involved, because

these cases are somewhat simpler. We then proceed to the timings involved in each action when DNSSEC is involved.

4.1 Adding a completely new $RRSet$

Adding a new $RRSet$ is the equivalent of changing $RRSet'$ to $RRSet''$ where $RRSet'$ is empty. The possible lifetime of the existing caches is the negative TTL of the target zone. Therefore, the time until visibility is $Z_d + T_{neg}$.

4.2 Changing $RRSet'$ to $RRSet''$

Whereas a completely new record is occluded in caches for the duration of the negative TTL, a change to an $RRSet$ will be occluded for as long as the TTL on the old $RRSet$. More formally, to replace $RRSet'$ with $RRSet''$, the time until visibility is $Z_d + T_{RRSet'}$.

4.3 Changing SOA parameters

The SOA RR contains several of the parameters governing Z_d and zone data visibility; but because it is itself part of the zone data, it is subject to the same timing considerations as other records. It is important to remember, therefore, that distribution of changes to the SOA is also subject to the same visibility rules: $Z_d + T_{RRSet'}$. In this case, $RRSet'$ is the original SOA RR, including its associated TTL.

4.4 Related RRSets

Some RRSets refer to other names. For example, an MX RRSet points to a name, and that name then has A or AAAA RRSet(s) to provide the actual IP address of the target host. These referring RRSets are sensitive to the visibility time of the related RRSet. To continue with the example, changing the target IPv4 address of a host will take $Z_d + T_{RRSet'}$ for the A record's RRSet; that means that an associated MX record will continue to direct mail traffic to the old IPv4 address until the visibility time for $RRSet''$ of the A record.

In a similar vein, it is important to remember that the TTL on related RRSets turns out to be the lowest of all the related RRSets. For instance, suppose the A record for `ns1.example.org` is used as glue[5] for NS records for `example.org`, and the NS records have a TTL of 86400, but the A record has a TTL of 600. Whenever a query has to be answered for `example.org`, the name server `ns1.example.org` may be consulted. In this case, even if the answer for `example.org` is still cached, there will need to be a query to the authoritative name servers whenever the record for `ns1.example.org` has expired. In practice, this will mean that the available cache time is only 600, rather than 86400, seconds.

4.5 General remarks on DNSSEC RRs and zones

DNSSEC makes the DNS more timing-sensitive in two ways. First, because of the absolute beginning and end times of the signature lifetimes, failures can be directly induced whenever a necessary action is not completed in time. This is dissimilar to traditional DNS operation, where all of the timings are intervals (and therefore are only sensitive to the time since the last action). Second, whereas gaps in data in traditional DNS operation are often in practice covered up by data otherwise available from the cache, DNSSEC requires that data come with a proof of its origin. If the data cannot be validated, the answer will be treated as bogus. This means that, for reliable operation, fewer errors can be tolerated than in (current) non-DNSSEC practice.

[5] Among other things, a "glue record" is used to provide the IP addresses for name servers, when those name servers are subordinate to the zone being served.

4.6 RRSIG lifetimes on RRSets

Every RRSIG has an expiry time on it. Therefore, S_f must occur in time for the signature to be visible everywhere before S_e. So, $S_f \leq S_e - Z_d - T_{rrsig}$. This is true for every RRSIG. Note that the signature expiration, even though it is an absolute time, is not the only factor in this calculation.

4.7 Replacing DNSKEY records

Because the DNSKEY records contain the keys that are the basis for generating RRSIGs, actions on them are extremely sensitive to timing. Unless the key that produced the signature is available to the validator at the time of validation, validation will fail and the RRSet will be treated as bogus. This means that every action needs to be timed so that all the necessary keys are visible at any moment.

There are at base two different strategies for replacing a given key, K. One is to work with only one RRSIG to cover each RRSet in the zone; each RRSIG is generated using either K' or K''. An alternative is to use more than one RRSIG to cover each RRSet (at least during the key replacement period). In this case, $RRSIG'$ is generated with K', and $RRSIG''$ is generated with K''; when they key replacement period is over, all $RRSIG'$ and K' records are removed from the zone.

In what follows, we provide different series of steps for replacing ZSK and KSK DNSKEY records. Remember, however, that from the protocol point of view, these are not really different kinds of keys, so they will all be returned as DNSKEY records for the zone's RNAME. The different procedures simply reflect different operational considerations because it is usually the KSK that forms the basis for the DS record in the parent. We present the actions in a table, with exactly one action happening on each line (some "actions" are things that happen in the protocol; some are just time passing).

4.7.1 Replace ZSK: two keys, one RRSIG per RRSet

In this example we replace K' with K'' in a "lazy" fashion: the different RRSets may be signed using different keys under the same Z_{ser}. Both K' and K'' are signed by the KSK for the zone. The steps are outlined in Table 6.

The table indicates the lifetime and activity period of the two keys in question, at three points: in the zone itself, at some point on the Internet that happens to see the latest data in the zone ("Maybe visible"), and at any point on the Internet ("Certainly visible"). The lifetime is indicated on the upper line in each case. On the lower line, we show the possible signed RRSets available at that point on the Internet. Remember that key lifetime is determined by whether the key appears in the DNSKEY record for the zone, and that the use of a key to produce a signed RRSet is what determines its activity period. Each line includes the state at the *end* of the step.

The table illustrates how the lifetime and activity level of keys is apparently different at the authority server and from various points on the Internet. For instance, Step 2 introduces K'' to the zone data, but it only becomes visible after $Z_d + T_{dnskey}$.

At step 4, K' is still active in the zone because it is the basis for some signatures. For any RRSet in the zone, either K' or K'' is used to generate the RRSIG, but never both. This is indicated by the exclusive or symbol ($\oplus$) in the table. At a given point on the Internet, between steps 4 and 9, it is likely that K' and K'' will both appear to be in use; but they will never both appear to be in use to generate RRSIGs for the same RRSet.

Steps 4 and 10 use $max(T)$ instead of $T_{rrsig'}$ because it is possible for the TTLs in the zone to vary, and it is difficult to know the TTL on the RRSet that was just signed. If the TTL were available, then at these steps a more optimistic value might be available.

Assuming all keys are working properly, this series of steps will ensure that no validator ever sees an RRSIG that it cannot validate. As we noted in Section 4.6, it is critical that step 10 completes for each

			In zone	Maybe visible	Certainly visible
Step	Action	Time	K_l / $R[K]$	K_l / $R[K]$	K_l / $R[K]$
1	Start	0	K' / $R[K']$	K' / $R[K']$	K' / $R[K']$
2	Add K''	0	$\{K', K''\}$ / $R[K']$	$\{K', K''\}$ / $R[K']$	K' / $R[K']$
3	Wait	$Z_d + T_{dnskey}$	$\{K', K''\}$ / $R[K']$	$\{K', K''\}$ / $R[K']$	$\{K', K''\}$ / $R[K']$
4	Generate first RRSIG with K''	0	$\{K', K''\}$ / $R[K'] \oplus R[K'']$	$\{K', K''\}$ / $R[K']$	$\{K', K''\}$ / $R[K']$
5	Wait	Z_d	$\{K', K''\}$ / $R[K'] \oplus R[K'']$	$\{K', K''\}$ / $R[K'] \oplus R[K'']$	$\{K', K''\}$ / $R[K']$
6	Wait	$max(T)$	$\{K', K''\}$ / $R[K'] \oplus R[K'']$	$\{K', K''\}$ / $R[K'] \oplus R[K'']$	$\{K', K''\}$ / $R[K'] \oplus R[K'']$
7	Loop	$P_{SIG_{K''}(Z)}$	$\{K', K''\}$ / $R[K'] \oplus R[K'']$	$\{K', K''\}$ / $R[K'] \oplus R[K'']$	$\{K', K''\}$ / $R[K'] \oplus R[K'']$
8	Generate Last RRSIG with K''	0	$\{K', K''\}$ / $R[K'']$	$\{K', K''\}$ / $R[K'] \oplus R[K'']$	$\{K', K''\}$ / $R[K'] \oplus R[K'']$
9	Wait	Z_d	$\{K', K''\}$ / $R[K'']$	$\{K', K''\}$ / $R[K'']$	$\{K', K''\}$ / $R[K'] \oplus R[K'']$
10	Wait	$max(T)$	$\{K', K''\}$ / $R[K'']$	$\{K', K''\}$ / $R[K'']$	$\{K', K''\}$ / $R[K'']$
11	Remove K'	0	K'' / $R[K'']$	$\{K', K''\}$ / $R[K'']$	$\{K', K''\}$ / $R[K'']$
12	Wait	Z_d	K'' / $R[K'']$	$\{K', K''\} \oplus K''$ / $R[K'']$	$\{K', K''\}$ / $R[K'']$
13	Wait	T_{dnskey}	K'' / $R[K'']$	K'' / $R[K'']$	K'' / $R[K'']$
Total time		$(2 \times (Z_d + T_{dnskey})) + (2 \times (Z_d + max(T))) + P_{Z[K'']}$			

Table 6: Replacing a ZSK using two keys and one RRSIG per RRSet

$RRSet$ before $S'_e(RRSet) - Z_d - T_{rrsig}$. After that point, the old RRSIG is expired, and that RRSet may fail validation.

The total time noted is really a minimum time for the entire set of actions to be completed; if an operator takes longer than the noted time for each step, then of course the real time to completion will be longer. Step 7 is a loop, so it is sensitive to $O(Z)$. It is also worth noting that the steps after step 10 are all clean-up steps: once the new key is used for all signatures, the old key is effectively retired. The clean-up is important, however, because the old key will continue to be visible until the final step.

4.7.2 Replace ZSK: two keys, two RRSIGs on each RRSet

In this variant, the RRSIGs generated with K'' co-exist with the RRSIGs generated with K' until all records are signed with K''. At that point, the old key and signatures are removed. It works just the same way as outlined in Table 6, except that additional time is needed for the period in which to remove $S_{K'}(RRSet)$ for every RRSet in the zone. In addition, instead of *either* $R[K']$ or $R[K'']$ being available while the new key is introduced, *both* are available. The generation of the complete table is left as an exercise for the reader.

A significant disadvantage of this approach is that every record in the zone gets two signatures at some point, and portions of the zone have two signatures for a long period of time. Additional signatures increase the size of the responses to DNS queries; and increase the amount of storage space needed to hold the zone, and the disk bandwidth needed to answer a given query. It might appear that this approach can extend the effective key lifetime K'_l considerably, because if a signature expiry R'_e will occur before the new signature is generated, the old signature-maintenance regime using K' may ensure that no signatures expire. In fact, the expiry R'_e occurs at exactly the same time whether the signature is replaced using K' or K''. Therefore, it does not seem that a two-signature approach is any safer, and it presents some disadvantages.

4.7.3 Replace ZSK: complete replacement of all RRSIGs

In some modes of operation, a complete re-signing of the zone is undertaken every time the ZSK is replaced. In such an operation, the step-by-step analysis outlined in Table 6 does not have a perfect analogue. Instead, times need to be calculated using values for the whole zone and with $max(T)$ in every case. A complete working out of the steps is left as an exersize to the reader.

4.7.4 Replacing the KSK

The KSK usually only signs the DNSKEY RRset and no other RRsets. Therefore, no RRSIG records covering "ordinary" RRSets will be generated with the KSK. The RRSIGs covering the DNSKEY records are generated using the KSK.

There are two modes of operation for signed zones: one, where a delegator[6] provides a secure delegation via DS records, and another without a secure delegation at all. In the latter case, the mechanism defined in [13] is used to automate the update of trust anchors. Such a mechanism will always be needed for the root zone, but zones without any trusted delegation can also use that approach no matter where they are in the DNS tree. Because [13] deals explicitly with timing, and because it takes much of the operation out of the zone's control, we do not cover that case here. So, let us turn to the question of replacing the DS record in the parent.

There are three different possibilites on how to roll keys. Not every possibility will be available in every case, because some policies may be restricted by the policy of the delegator. The three possibilities are

1. two KSK and two DS

[6] Usually the parent, but in some circumstances some other trust anchor repository.

2. one KSK and two DS

3. two KSK and one DS

The first of these is conceptually simplest, and therefore possibly safest, so we work through it in what follows.

4.7.5 Replace KSK: two keys, two DS records in the parent

The safest way to replace a KSK is to ensure that the new KSK is added to the parent zone as soon as possible after it becomes available. We outline the series of steps to do so in Table 7. The table contains an additional column, outlining what signed RRSets could be valid, given the DNSKEY RRSet in the child and the DS RRSet from the parent. It is worth noting that, at step 5, $R[K'']$ is certain to be a valid RRSet, even though there is no such RRSet in existence yet.

Step 6 does not remove $R[K']$, which is why signatures from both K' and K'' appear in the table for a period. It is actually possible to replace K' completely at step 6: since both $R[K']$ and $R[K'']$ are certainly valid as of step 5, in fact K' is entirely redundant after that. The approach we outline is therefore not the most efficient, but it parallels the operation defined in [13] and may therefore be familiar to operators.

After step 9, it is possible to perform some operations in parallel: once $DS_{K'}$ is removed from the parent, nobody trusts it any more anyway, so it cannot be used as the basis for validation.

5 Conclusion

The timing parameters in DNS can mostly be set independently, but the foregoing shows that several of them are tightly interconnected. Moreover, it is apparent that correct operation of a zone, particularly when using DNSSEC, requires careful attention to the interplay of the individual parameters.

For zone operators, this means that choosing a value for any of the parameters should only be done when taking into consideration the wider set of parameters. It is useful to work out the effects of the settings of all the parameters, and work through the specific results for cases such as we have presented here. Tool builders can help zone administrators by providing tools that model the effects of all parameter settings at once, and by detecting potentially dangerous cases (such as TTLs on RRSIGs that will allow expired signatures to remain cached). In addition, tools that detect when a given state has been reached in the key rollover process (so that the next one can start) would be invaluable.

Because of the relationships of these parameters appears to be calculable, future work may uncover a general formula that would permit the calculation of the correct settings of all the parameters on the basis of knowing just one or two of them. Such a formula would be a boon to zone operators and tool builders, who are currently confronted with a number of apparently independent variables.

Step	Action	Time	In zone K_l $R[K]$	Maybe visible K_l $R[K]$	Certainly visible K_l $R[K]$	Certainly valid $R[K]$
1	Start	0	K' $R[K']$	K' $R[K']$	K' $R[K']$	$R[K']$
2	Add K''	0	$\{K', K''\}$ $R[K']$	$\{K', K''\}$ $R[K']$	K' $R[K']$	$R[K']$
3	Wait	$Z_d + T_{dnskey}$	$\{K', K''\}$ $R[K']$	$\{K', K''\}$ $R[K']$	$\{K', K''\}$ $R[K']$	$R[K']$
4	Add DS" (to parent)	0	$\{K', K''\}$ $R[K']$	$\{K', K''\}$ $R[K']$	$\{K', K''\}$ $R[K']$	$R[K']$
5	Wait	$Z_d + T_{ds}$ *Parent*	$\{K', K''\}$ $R[K']$	$\{K', K''\}$ $R[K']$	$\{K', K''\}$ $R[K']$	$R[K'] \wedge R[K'']$
6	Sign DNSKEY with K''	0	$\{K', K''\}$ $R[K'] \wedge$ $R[K'']$	$\{K', K''\}$ $R[K']$	$\{K', K''\}$ $R[K']$	$R[K'] \wedge R[K'']$
7	Wait	Z_d	$\{K', K''\}$ $R[K'] \wedge$ $R[K'']$	$\{K', K''\}$ $R[K'] \wedge$ $R[K'']$	$\{K', K''\}$ $R[K']$	$R[K'] \wedge R[K'']$
8	Wait	$T_{rrsig'}$	$\{K', K''\}$ $R[K'] \wedge$ $R[K'']$	$\{K', K''\}$ $R[K'] \wedge$ $R[K'']$	$\{K', K''\}$ $R[K'] \wedge$ $R[K'']$	$R[K'] \wedge R[K'']$
9	Remove DS' (from parent)	0	$\{K', K''\}$ $R[K'] \wedge$ $R[K'']$	$\{K', K''\}$ $R[K'] \wedge$ $R[K'']$	$\{K', K''\}$ $R[K'] \wedge$ $R[K'']$	$R[K'] \wedge R[K'']$
10	Wait	$Z_d + T_{ds}$ *Parent*	$\{K', K''\}$ $R[K'] \wedge$ $R[K'']$	$\{K', K''\}$ $R[K'] \wedge$ $R[K'']$	$\{K', K''\}$ $R[K'] \wedge$ $R[K'']$	$R[K'']$
11	Remove $R[K']$	0	$\{K', K''\}$ $R[K'']$	$\{K', K''\}$ $R[K'] \wedge$ $R[K'']$	$\{K', K''\}$ $R[K'] \wedge$ $R[K'']$	$R[K'']$
12	Wait	Z_d	$\{K', K''\}$ $R[K'']$	$\{K', K''\}$ $R[K'']$	$\{K', K''\}$ $R[K'] \wedge$ $R[K'']$	$R[K'']$
13	Wait	$T_{rrsig'}$	$\{K', K''\}$ $R[K'']$	$\{K', K''\}$ $R[K'']$	$\{K', K''\}$ $R[K'']$	$R[K'']$
14	Remove K'	0	K'' $R[K'']$	$\{K', K''\}$ $R[K'']$	$\{K', K''\}$ $R[K'']$	$R[K'']$
15	Wait	Z_d	K'' $R[K'']$	$\{K', K''\} \oplus$ K'' $R[K'']$	$\{K', K''\}$ $R[K'']$	$R[K'']$
16	Wait	T_{dnskey}	K'' $R[K'']$	K'' $R[K'']$	K'' $R[K'']$	$R[K'']$
Total time		$(2 \times (Z_d + T_{dnskey})) + (2 \times (Z_d + T_{rrsig'})) + (2 \times (Z_d + T_{ds})_{parent})$				

Table 7: Replacing a KSK:two keys, one RRSIG per RRSet, and one DS record

References

[1] M. Andrews. Negative Caching of DNS Queries (DNS NCACHE). RFC 2308 (Proposed Standard), March 1998. Updated by RFCs 4035, 4033, 4034.

[2] M. Andrews. Edns expire option. Work in Progress, August 2008. draft-andrews-dnsext-expire-00.

[3] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. DNS Security Introduction and Requirements. RFC 4033 (Proposed Standard), March 2005. Updated by RFC 6014.

[4] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Protocol Modifications for the DNS Security Extensions. RFC 4035 (Proposed Standard), March 2005. Updated by RFCs 4470, 6014.

[5] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose. Resource Records for the DNS Security Extensions. RFC 4034 (Proposed Standard), March 2005. Updated by RFCs 4470, 6014.

[6] R. Elz and R. Bush. Clarifications to the DNS Specification. RFC 2181 (Proposed Standard), July 1997. Updated by RFCs 4035, 2535, 4343, 4033, 4034, 5452.

[7] O. Kolkman and R. Gieben. DNSSEC Operational Practices. RFC 4641 (Informational), September 2006.

[8] O. Kolkman, J. Schlyter, and E. Lewis. Domain Name System KEY (DNSKEY) Resource Record (RR) Secure Entry Point (SEP) Flag. RFC 3757 (Proposed Standard), April 2004. Obsoleted by RFCs 4033, 4034, 4035.

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

[10] P.V. Mockapetris. Domain names - implementation and specification. RFC 1035 (Standard), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966.

[11] S. Morris, J. Ihren, and J. Dickinson. DNSSEC Key Management. Work in Progress, January 2009. draft-morris-dnsop-dnssec-key-management-00.txt.

[12] M. Ohta. Incremental Zone Transfer in DNS. RFC 1995 (Proposed Standard), August 1996.

[13] M. StJohns. Automated Updates of DNS Security (DNSSEC) Trust Anchors. RFC 5011 (Proposed Standard), September 2007.

[14] P. Vixie. A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY). RFC 1996 (Proposed Standard), August 1996.

DOMAIN NAME SYSTEM SECURITY (DNSSEC)
DEPLOYMENT COORDINATION

Contract No: FA 8750-10-C-0020

Data Item A005, CLIN 0002

Technical Information Report:
Protocols for TLD & Registrar Adoption,
Observing Validation in the Wild

Submitted by:

Shinkuro, Inc.
Bethesda, MD

Stephen D. Crocker
Principal Investigator

Jeffrey Dewhurst
Financial and Contract Administration

March 2011

Observing DNSSEC validation in the wild

***Abstract*—DNSSEC protocol deployment has taken place in phases, first protocol development followed by signing of top level zones and early adopter leaf zones. The next phase is to encourage wide scale validation, as that will improve the over DNS system and enable new applications. In order to tell people about DNSSEC usage it is important to be able to measure how many zones are signed and how widespread validation is. This paper will describe how to measure validation by looking at DNS queries. In this paper we present results from two sample periods monitoring a sub-set of the authoritative name serves for ORG.**

***Index Terms*—DNSSEC, DNS operations, Internet measurements.**

I. Introduction5

In this paper we will talk about how to measure the population of resolvers performing DNSSEC[cite:RFC4034] validation. Knowing the prevalence of validation among resolver populations has number of possible uses, measure global trends, identify communities that are generally using DNSSEC validation, auditing etc. .

A. Different types of resolvers from DNSSEC point of view

The standard classification of resolver into recursive and stub resolvers, does not apply to DNSSEC validation. What we are interested in what kind of DNSSEC behavior the resolvers show:

A) Validating Resolvers
B) DNSSEC capable but not validating
C) DNSSEC ignorant

In our discussion we in general make the following simplification Resolver == IP address. In many cases recursive resolvers are located behind a NAT device and we see evidence that there are frequently multiple resolvers behind a single IP address. When there are multiple resolvers we classify the address by highest observed behavior (A-C). We classify the resolvers by looking at the content of a DNS query and DNSSEC specific query patterns.

DNSSEC ignorant resolvers (C) never ask a question with EDNS0 and DO bit set, thus if all questions seen from an address have this behavior we classify the resolver as DNSSEC ignorant. This is not strictly correct as there are DNS proxies that strip EDNS0, and when multiple resolvers are behind a NAT then we can not tell the difference between each one.

B. Goals of this work: Be able to monitor DNSSEC validation growth

We want to develop simple rules and techniques that can be build into DNS monitoring systems and can issue cumulative results all the time not just once in a while like we did, or if that is not possible define a subset the data that DNS operators collect, that will allow validation measurements. Large DNS operators have to be able to handle enormous volume of DNS traffic, both answer the queries and frequently capture it to perform post-morem on incidents.

Your name is with xyz Department...

II. validation is Previous Work

In the past people have been monitoring how many zones are signed and other evidence of DNSSEC deployment. Cite: SecSpider[secspider] Lutz[lutz-dlv]. People have also looked a DNS cache performance[morris], how different DNS resolver operators perform[ager] etc. In addition lots of work was performed in the last few years to measure the deployment of ENDS0 and buffer sizes advertised[larson], and the impact of signing the root zone.

None of this is directly related to our work but we build on it.

III. Observing Validating resolvers

DNSSEC[RFC4035] extends the old DNS protocol by signing RRsets this provides source authentication. In order to perform these actions a chain of trust must be build from a trust anchor or the root of the DNS tree. The trust chain consists of 2 new RR types:

- DNSKEY that contains a public part of a key this record type is stored at the apex (top of) the zone the key will sign.
- DS is a cryptographic hash of the binding of a domain name and the public key at used to sign the DNSKEY set at the domain. The DS record is stored at the parent and is handed out as part of the referral returned by the parent when it receives a question for a domain name below a zone cut in one of the zones it is authoritative for.

A. Query Patterns to look for

Validating resolvers have slightly different query pattern from non validating ones, they will explicitly ask for records in the DNSSEC trust chain (DNSKEY and DS) that are almost never asked for by non validating resolvers. Below we will explain how each one works.

1) Monitoring DO bit : Resolvers that understand DNSSEC record types most of the time set the DO (DNSSEC OK)[] bit on queries. The presence of this bit can used as an indicator that the querier has potential to validate DNSSEC answers. Thus this resolver can be in category A or B.

2) Monitoring DNSKEY queries: : Every time as validating resolver attempts for the first time to validate information from a zone (secure.example.) it will get from the parent authoritative server (example.) a referral that contains a DS RRset. It will then ask the authoritative server for secure.example for

the Resource record it is looking for. Followed by a query for the DNSKEY record in order for it to be able to validate the resource record in prior query.

After the DNSKEY record has timed out of the validating resolvers cache, it needs to re-fetch the DNSKEY record in order to validate any future answers from the zone.

3) Monitoring DS queries : A parent zone can monitor validation on performed against a child zone by looking at DS queries. Each time the DNSKEY from the child times out and if the DS record has timed out then the validating resolver needs to fetch both DS record from parent (example) and DNSKEY RRset from the child (secure.example). When a DNSKEY RRset expires from cache in a validating resolver the next time this set is needed to validate new RRset the DNSKEY set is needs to fetched and validated. If the corresponding DS RRset has also expired the DS set needs to be fetched and validated.

B. Rules used for determining resolver category: Due to scattering of DNS queries.

A.1 DS query –> Validating resolver, actually each DS query increases the probability it is validating but for now we are assuming 1 query means validation.

A.2 DNSKEY followed by DNSKEY after TTL –> Validating resolver

A.3 query for X.zone followed by DNSKEY right after –> Validating resolver

A.4 DNSKEY seen —> Possibly validating Due to scattering of DNS queries.

We realize that we might be classifying more than just recursive resolvers, there are monitoring tools, test queries and trust anchor maintainers that may issue similar query patterns. We have decided that these tools usage is closely related to DNSSEC deployment and thus we count them in.

While these rules may seem straight forward to apply there are complications depending on many factors including. If a complete set of traces over a long time from all authoritative name-servers can be analyzed it is possible to build a good picture as to how many of the resolvers in the domains "working set" are validating.

C. Traffic selection

What we wanted to do is to determine the world wide DNSSEC validation and we wanted a domain to analyze that is used all over the world and not just by the technical community. This lead us to approach the the operator of the .ORG domain, after discussion with PIR and Afilias we were granted access to some traces from the name-servers that Afilias operates for .ORG. This is only a partial set of name-servers. The volume of data is a big issue for all concerned, thus we started out analyzing short (as in time) samples. In this paper we talk about our initial results from analyzing this traffic and how in general to measure DNSSEC validation in the wild.

1) What DNSKEY query to look for ? : Our original thought was to only monitor the explicit queries for ORG's DNSKEY, then we thought about expanding our measurements to all DNSKEY queries for domains inside ORG. The reason is this allows us to detect more validators and overcome the scatter of DNS queries. There are 2 domains inside ORG that seem to have significant DNSKEY traffic for them ISC.ORG and IETF.ORG. Afilias operates most (5 out of 6) name-servers for IETF.ORG thus we see most of the traffic for IETF.ORG. This indicates that DNSSEC validation is taking place for IETF.ORG. but the traffic volume was not significant enough to draw conclusions. ISC.ORG has lots of DNSKEY traffic this seems to be driven in large part by traffic related to DLV.ISC.ORG, which was a tool for early DNSSEC adopters to validate via an alternative trust chain. We thought about including DLV.ISC.ORG look-ups in our statistics but decided against that even though that would inflate the numbers. The reason being we waned to measure the traffic of validators that validate to the root. Many of the resolvers that are looking for DLV.ISC.ORG do not seem to have been configured with the root key. Investigating the DLV.ISC.ORG traffic in more detail, is going to be an interesting topic, at first glance it looks like there are more validators resolving to the ISC DLV[cite] Trust Anchor than to the root at the time of measurements, but some are using both the root key and the DLV key.

D. Monitoring the ORG domain traffic

In middle of 2010 we got permission from PIR and Afilias to access query traces of the ORG domain from the Afilias's name server clusters, to help develop methodologies and tools to measure DNSSEC validation in the wild. ORG was rare among TLD's that its DNSKEY RR TTL is real short only 15 minutes, most other TLD's have much longer TTL on DNSKEY records. See table 1. for some of the DNSKEY TTL and DS TTL values used in signed TLD's.

Table I
SAMPLE OF SIGNED TLD TTL VALUES

domain	DNSKEY TTL	DS TTL	NSEC TTL
. (root)	2 days	1 day	1 day
org	15 min	1 day	1 day
br	6 hours	1 day	15 min
net	1 day	1 day	1 day
se	1 hour	1 hour	2 hours
us	1 day	2 hours	1 day
fr	2 days	not found	90 min
cz	1 hour	5 hours	15 min
jp	1 day	1 day	15 min

At this point there is no agreement among operators or protocol experts what are good values. A comprehensive timing analysis is needed to identify the better values, but that is frequently also affected by different goals. This topic is outside the scope of this paper.

We have collected traces for short periods (30 - 50 minutes), periodically to analyze and improve our tools and techniques.

1) .ORG Name-server distribution impact on query patterns : We have access to traces for the name-servers that Afilias (ORG Registry) operates. The NS set for .ORG has 6 name-serves all the addresses are any-casted, 4 of the addresses

are provided by Afilias and 2 are provided by an out side contractor (PCH). Afilias operates 5 sites around the world and PCH provides ORG resolution in about 15 locations.

The first question is how much of the .ORG traffic ends up on servers we have traces from. A naive answer is 4/6'th or 2/3'rds, but that is not how DNS resolves work. Most recursive DNS resolvers try to go to the "closest" authoritative server for a given domain, thus we have two different behaviors for resolvers. In the beginning scattering of queries while trying to figure out which one is the closest one. Once that has been established most of the queries go to the closest one, with a sporadic one checking if distance has changed. Different resolvers have different criteria for selecting "closest" authoritative server, Unbound for example says all servers that are within [min_rt .. min_rtt + 400ms] are equal, and with anycasting almost all the ORG name-servers will fall into this band. Bind on the other hand has much smaller band (about 20ms) thus it will discriminate more among authoritative servers. [1]

For this reason some busy resolvers will concentrate queries to a single address. But resolvers that do not send lots of queries will frequently scatter the queries, except the ones that do not care about performance and only send queries to the first name-server in the NS set the resolver got. The implications of this behavior are two fold:

- resolvers that resolve lots of ORG queries will either be in our sample or not at all.
- resolvers that only do occasional ORG query will show up in our samples but we may not see enough of its queries to determine if it is performing DNSSEC validation.
- resolvers that do not do RTT optimization will lock in on a "random" server.

To understand better the actual situation is Afilias's servers answer about 50% of the queries for .ORG. Based on this we need a model that tells us how to apply what we see in the sample we have as to how the whole resolver population behaves. If there are geographical communities that are strong DNSSEC validators we may either see that large part of the community or it will look like a low volume resolver.

2) Sample period and what to look for : One of the main reasons ORG is a good domain to use to measure DNSSEC validation in the wild is the short TTL on its DNSKEY as that forces validating resolver to re-query for the DNSKEY every 15+ minutes. Of course the validators only fetches DNSKEY when it needs it thus the 15+. The DS TTL of 1 day on the other hand means that DS queries are not as common as in TLD's with short DS TTL. Many of the DS queries are for domains that have a different TTL on their NS and DNSKEY RRsets.

For example in .BR domains the DNSKEY TTL is 24 hours but the TTL on the NS record in the hit is also 24 hours but the TTL on the DS records is 6 hours. Thus for domains that have TTL on their D5NSKEY less than 24 hours there is a potential for a DS query.

[1]Unbound does this scattering of queries. This indicates that DNSSEC validation Is for cache spoofing prevention, Bind is more tuned for performance.

The sample period for ORG has to be long enough that the probability of DNSKEY query happen multiple times for a busy validating resolver. For a non busy validator the time of day may have a significant impact on the results. One of our assumptions was Validator == IP address but for longer period that may not hold as well and we get either multiple resolvers at a single address or a mobile resolver will show up on multiple addresses.

We selected originally 30 minutes as a sample size but after some experience increased that to 50 minutes to increase the possibility of seeing DNSKEY questions. Traces come in 10 minute increments.

IV. Results

Due to the scatter of queries and the fact we only see a subset of authoritative serves we were unable to apply rule A.3. We have analyzed two sets of traces one from early November 2010 and the second one from early January 2011. Both traces include all of the Afilias sites, and are from the same time of day 2000-2050 UTC.

The time was chosen randomly and we expect that if there are significant differences in geographical distribution of DNSSEC validators the time of day will have some impact on the results. What we are mainly interested in at this time is to see if there were changes in DNSSEC validation.

The traces are full packet captures thus a quite large, requiring long transfer time and processing time. We compacted the results to focus on the models above there are significant other topics to research in the traces like TCP usage and trying understand the client behaviors.

A. November 2010 Results

The table 2 shows how many different resolvers asked for DS records from each site in each 10 minute interval, "Seen" is the number of validators this state has seen. "Population" is the cumulative number of validators seen from the beginning by all sites.

Table II
DS queries seen in each period in Nov 2010

	A	B	C	D	E	Seen	Population
0-10	1971	921	218	881	348	3066	3066
10-20	478	880	1146	823	342	2609	4028
20-30	1753	850	1124	832	297	3191	4765
30-40	1405	880	835	825	338	2816	5202
40-50	289	828	929	808	361	2315	5439
Total	2869	1894	1924	1682	651		

Table 3 shows how many resolvers passed the two DNSKEY look-up test (A-2). The large difference between Questions and Sources is to large extent caused by validators that advertise larger EDNS0 value than fits on their "site" link and the fact that the .ORG DNSKEY set is over

As you can see in table 3 only about 29% of the resolvers that issue DNSKEY query pass our test but only 863 are certified by a single site, thus at least 312 validators are pass because we are seeing questions at multiple sites. This is partially caused by the scattering of DNSKEY queries even

Table III
DNSKEY QUERIES SEEN AND RESULTS OF DNSKEY CRITERIA

	Questions	Sources	Passed	Pass %
A	3804	1337	118	8.8
B	7148	1169	238	20,4
C	3201	1097	179	16.3
D	3565	929	210	22.6
E	912	481	118	24.5
Total	18630	4045	1175	29.0

though we had 50 minute trace the probability we see the second DNSKEY query is still constrained by the scattering of queries and the need of the validator to fetch the ORG's DNSKEY.

In addition 993 validator are confirmed by both rules, and we have a total of 5621 validators confirmed, i.e. only 182 are confirmed by DNSKEY look-ups. Further of the 2870 possible validators 2764 are on the DS list leaving only 106 possible ones.

B. January 2011 results

In table 4 there are the results for DS queries and in table 4 the DNSKEY results. We see that there are fewer DS sources than in the prior sample

Table IV
JANUARY 2011 DS QUERIES AND SOURCES

	A	B	C	D	E	Seen	Population
0 - 10	1469	681	641	614	350	2607	2607
10 - 20	796	533	725	457	340	1951	3209
20 - 30	707	711	375	604	343	1879	3453
30 - 40	1384	406	320	473	307	2140	3817
40 - 50	782	345	259	593	320	1630	3970
total	2171	1171	1034	1149	686	3970	

In table 5 we have the DNSKEY look-up test results. Adding up the Passed column for all sites in Table 4 adds up to 615 thus there are at least 143 that passed by only correlating multiple sites. The strong tendency of validators to be successfully pass this test at a single site indicates that there are many more that we can not see.

Table V
JANUARY 2011 DNSKEY

	Questions	Sources	Passed A.2 test	Pass %
A	3889	1519	248	16.3
B	3364	859	99	11.5
C	1316	496	36	7.3
D	1795	749	134	17.9
E	964	511	98	19.2
Total	11196	3348	758	22.6

Of the 2590 possible validators from this results 1805 are on the DS list leaving 785 as possible validators. This gives us 4728 confirmed validators.

C. Comparing the two sets results

At first glance at the tables 1-4 it is possible to draw the conclusion that DNSSEC validation is down, but before jumping to that conclusion we need examine if there are other factors in play. Traffic in the November trace contained about 59M questions, while the January trace only had 37M. January trace is collected on a Sunday but the November one on a Tuesday. In table 6 we present more statistics about the two collection periods.

Table VI
STATISTICS FOR BOTH PERIODS

	Nov 2010	Jan 2011	
Questions	58891887	37342902	62% of prior sample
Resolvers	676599	573773	85% of prior sample
No DNSSEC	214036	185954	
DNSSEC Capable	462563	387779	
% DNSSEC capable	68.4	67.7	
Confirmed Validators	5621	4728	84% of prior sample
Suspected Validators	106	785	

In both periods the percentage of confirmed validators is about 1.2% as total number of resolvers, but that does not tell the whole story the lower query volume probably means we have less of a chance to see DS or DNSKEY queries from busy resolvers. X

Back to DLV.ISC.org we see over 7300 different sources doing look-ups and less than half are on the list we built using our criteria.

D. How much validating goes on ?

A more fundamental question is how many answers from ORG are potentially validated as a fraction of all answers?

- In the 2010 November sample the confirmed and possible validating resolvers accounted 8% for of the observed DNS queries to ORG.
- In the 2011 January sample the confirmed and possible validating resolvers accounted for over 10% of the observed DNS queries to ORG.

Some of this traffic is caused by validators that advertise too large of a EDNS0 buffer and can not receive the DNSKEY RRset from ORG that is over 1300 bytes. This is still an impressive percentage this early in DNSSEC deployment.

V. XCONCLUSIONS

In this paper we analyzed traces from .ORG and tried to estimate the size of the DNSSEC validating resolver population. We had 4 proposed methods for identifying the validators. Due to the behavior of DNS resolvers one of the one of the techniques was ruled out, as it will not work for validators that are "discovering" a domain or for busy resolvers that are "locked" in on "close" authoritative server. We applied the other 3 techniques to the samples and were able to estimate the size of validator population. Based on our results it seems that by simply looking at DS queries it is possible to determine if a resolver is a validator.

DOMAIN NAME SYSTEM SECURITY (DNSSEC)
DEPLOYMENT COORDINATION

Contract No: FA 8750-10-C-0020

Data Item A004, CLIN 0002

Technical Information Report
Initial best practice proposals for registrar transfer, name server changes, and unanticipated effects of caching

Submitted by:
Shinkuro, Inc. Bethesda, MD

Stephen D. Crocker
Principal Investigator

Jeffrey Dewhurst
Financial and Contract Administration

February 2011

Motivations and Terminology for DNSSEC Operations Handover

Stephen D. Crocker
steve@shinkuro.com and

Ólafur Guðmundsson
ogud@shinkuro.com and

Andrew Sullivan
ajs@shinkuro.com

Shinkuro, Inc.
4922 Fairmont Avenue,
Suite 250
Bethesda MD 20814 U.S.A.

***Abstract*—On the contemporary Internet, the DNS services for many domain names are operated by parties other than the registrant of the domain name. If a registrant wishes to move the domain name's DNS operations from one party to another, several different actors need to co-ordinate their actions. While this has always been true, the addition of DNSSEC signatures and validation to the DNS means that these operations become more delicate: while DNS without DNSSEC is mostly tolerant of data inconsistencies, DNSSEC is designed to detect such inconsistencies and to stop resolution after detection. In order to create a complete description of a smooth operational transfer without the domain having to go through an insecure phase, it is necessary to develop a taxonomy of all the actors, to understand when and how their roles might be combined and what that might mean, and to understand the effects of any actor's failing to act. We provide this description, and go on to suggest the next steps for a complete procedure for DNS operation transfer.**

***Index Terms*—DNSSEC, DNS, domain name operation transfer**

I. Introduction

THE Domain Name System (DNS) is designed to be loosely coherent. Loose coherence is necessary in a distributed, cached system because it is possible to get different answers to the same question from different parts of the network at the same time. One of the consequences of this loose coherence has been a high tolerance to delayed action and outright misconfiguration on the parts of both DNS operators and the system itself. The introduction of DNSSEC means that some of that tolerance will be lost.

Many DNS domain registrants do not actually operate their own DNS zones, instead preferring to hire specialist DNS operators. The competitive market in DNS operation means that it is sometimes necessary to move a zone from one operator to another. The loose coherence of the DNS has prevented those moves from being too difficult. But the advent of DNSSEC promises to complicate matters, because the failure to respect the timing of activities (and sometimes, an unwillingness by vendors who are losing a customer to co-operate) means that a misstep can put a zone into a bogus state, and cause resolution failures for that domain name.

In what follows, we first explain why there is a problem to be solved. We then provide a taxonomy of all the roles involved in the current domain name, DNS, and DNSSEC marketplaces, and then provide a sketch of a future complete procedure for transferring DNS operation of a zone without turning of DNSSEC signatures for that zone.

II. The Competitive Environment

While the DNS offers the opportunity to break the name space up into different pieces, each of which may be administered mostly independently, it does not operate as a closed system. To begin with, there are the questions of how domain names and, underneath them, other DNS records, appear in the DNS. In addition, there is the matter of who actually performs the operation of the DNS for the zone in question.

A. The domain name registration market

In parts of the DNS name space, the registration of domain names itself has become a commercial activity. There are two basic models of this activity: a two-party registration model, and a three-party registration model.

In the two-party registration model, someone wishing to register a domain name inside a name space approaches the registration authority, and undertakes an agreement with that authority. As a result, the authority enters the registration into the set of names inside the name space. There may be delegation consequences of this; see II-B.

In the three-party registration model, someone wishing to register a domain name inside a name space approaches an authorized agent for that name space. The authorized agent contacts the registration authority, and inquires as to whether the registration is permitted. If so, the agent registers the domain name on behalf of the initial requester, and then enters the registration information about the individual in its own records.

The registration authority described above has historically been called the "registry", and the original requesting party the "registrant". While strictly speaking the term "registry" might properly be used for any level of registration authority in the DNS, the concentration of commercial activity at or near the root has caused the term "registry" in some circles to mean more or less "TLD- and near-TLD-registries".

The authorized agent in the three-party model is what is usually called the "registrar". For historical reasons, this word

is also closely associated with the administration of top-level and near-top-level domains. There is nothing about the model that requires such an interpretation, however. For instance, in a university, it would not be at all surprising for there to be a central registry of names in the university, all operated as one zone; but for each department to control some part of the namespace, and therefore to be working effectively as a registrar.

The three-party registration model is extremely common among top-level domains and is so widely considered "normal" that one frequently hears the two-party model described as, "The registry is also the registrar." It is also possible to have a hybrid model, in which the registry will perform registration functions directly on behalf of registrants, but will also allow other registrars to compete in performing the registration function.

The three-party model varies from registry to registry in its implementation. For instance, some registries require that a lot of ancillary data about the domain name registered also be deposited with the registry (the so-called "thick registry" model). Other registries require nothing more than the registration, the registrar's (or, in the terminology of one protocol, "sponsor's") identifier, and the data necessary to publish the domain name's data in the DNS. More importantly for our purposes, some registries have very strict rules that all the data in the registry must always and only come from the registrars, whereas others permit various classes of information to be maintained directly in the registry by individuals associated with the domain name (such as the registrant, the administrative contact, or the technical contact).

The models outlined above are not quite as clean as suggested, because in fact under neither model does the registry have to be simple. Instead, the registry might be made up of the registry operator (which generally sets policies, writes contracts, and so on), and the technical operator (which generally performs the actual technical operation of registration and, perhaps, other activities). These various business models might be interesting in themselves, but they have in principle no effect on the basic registry function, and do not affect any conclusion about how to perform operations transfers except as matters of implementation detail. Therefore, we shall ignore this distinction.

B. The delegation market

One may distinguish between the registration and delegation functions in a registry. Simply registering a domain in a registry does not automatically entail that the registration is delegated for operation on the Internet. To see that this is so, consider the case of so-called "reserved" names in registries, which are not permitted to be registered, but which are not allowed to resolve on the Internet either. Similarly, there are names registered that are not delegated, because they do not meet the requirements for delegation (such as having some minimum number of name server records). These cases demonstrate that registration and receiving a delegation are not the same thing.[1]

Many (perhaps most) registrants registering in a registry expect a delegation of name space. What one receives, in this case, is the ability to designate certain servers as the authoritative DNS servers for that domain; and (accordingly) to operate any services one likes inside the delegated name space (subject, of course, to contractual limitations).

Under most circumstances, the delegation market is completely bound up with the registration market: acquiring a registration entails the right to a delegation too. The distinction is important for our purposes, however, because the parties involved in delegation and the parties involved in registration need not be the same.

It is a prerequisite to participate in the delegation market that one holds a registration. Both registrants and registrars participate in the delegation market. In the two-party registration model, the registrant simply requests delegation from the registry, which involves submitting some number of name server records for the domain to the registry. The registry adds these records to the registry's zone.

In the three-party registration model, the registrar adds the name server records for the domain. These may be records that the registrant asks to have submitted, or they may be records the registrar submits without prompting; it is not unusual to see domains that have been registered but for which the registrant has not added name server records to resolve (usually to a host that serves HTTP requests and nothing more).

C. The DNS operation market

There is no requirement that any or all of the name servers added by a registrant for a domain actually be under the direct control of the registrant, and many registrants opt to have their DNS services provided by someone else. In order for this to work, the registrant causes to be registered some DNS records with the DNS provider in a zone with the registrant's domain name at the apex, and then adds NS records containing the DNS provider's name servers to the registry. The DNS provider may or may not be the registrar of the three-party registration model. A registrant might use more than one DNS provider for a domain.

If a registrant moves the operation of some part of the DNS for a zone from one DNS operator to another, we can speak of these as the losing operator and the gaining operator respectively.

D. The non-DNS operation market

It might seem that the services offered at a name (such as web, email, and so on) are irrelevant to considerations of the DNS, but they are worth keeping in mind for three reasons:

1) Nobody would put anything in the DNS if it were not for the other services offered at a host.

[1]Moreover, as we noted, not all registries are at the top level, and not everyone who registers inside a lower level registry wants or expects a delegation. For instance, it is useful to think of the service blogspot.com as (partly) a registry. But nobody expects a delegation, in the DNS sense, of names inside blogspot.com.

2) Resolution failures in a domain will affect the services offered there.
3) The services themselves may be contracted out to a party other than the registrant.

If the host offering the service is not changing as part of the DNS migration, then there is no reason that a DNS operator change needs to affect non-DNS operators. If, however, the non-DNS operations are part of a package of services that all need to move at the same time, then the change of DNS operators becomes still more delicate.

E. Summary: Dramatis Personae

Given the above, there are the following actors potentially involved in any transfer of DNS operation from one operator to another:

- The domain name registrant
- The domain name registry
- The domain name registrar
- Other domain name contacts
  - Administrative contact
  - Technical contact
- The losing DNS operator
- The gaining DNS operator
- The losing operator of "bundled" non-DNS services
- The gaining operator of "bundled" non-DNS services.

The other domain name contacts are only relevant insofar as they are able to effect changes to the delegation data (either directly in the registry, or indirectly by operating through a registrar, but in either case without the participation of the registrant).

III. Transferring Zone Operation Without DNSSEC

A. Isn't this trivial?

While most changes in the DNS are confined to a single zone, and can therefore be undertaken by that zone operator without paying a lot of attention to other zones[2], changes at the zone cut (the point where a delegation happens from a parent to a child zone) must be co-ordinated between the parent and the child. Historically, correct maintenance of the data across the zone cut has been troublesome. There are three reasons for this. First, because the database is distributed and loosely coherent by design, it is necessary to tolerate some differences across zone cuts. While RFC 1034 [3] requires that the "administrators of both zones should insure that the NS and glue RRs which mark both sides of the cut are consistent and remain so," but it is silent on how to do this. Since data will need to be inconsistent across the zone cut at some times in order to introduce changes[3], tolerance of inconsistency between the apex NS records of a zone and the delegation records at the parent is required for the DNS to operate.

Second, because that tolerance needs to be built in, it is easy for administrators of the DNS not to notice such configuration errors. Administrators' expectations in this regard have, moreover, been helped historically by the efforts of DNS server vendors to make every effort in support of resolution attempts. In particular, BIND 4-era systems were rather promiscuous in accepting glue records wherever those records came from. This meant that resolution continued to work even in the face of serious misconfiguration. While some of these practices have been tightened considerably over time due to their unfortunate security effects, the basic fact is that some inconsistency across zone cuts is a normal part of the functioning global DNS.

Finally, because DNS is designed for loose consistency, the overall system can be made at once more robust and less subject to performance reductions by caching recently-seen data in various places around the Internet. Caches improve robustness because even in the event of a transient name server failure, data might be available to answer a given query. It improves performance, obviously, by allowing answers to be provided immediately by a cache instead of needing to ask for the record from the authoritative server for the name.

So, the very things that make the DNS resilient are the things that contribute to operational problems with the DNS. It is therefore worth outlining exactly how a transfer should happen.

B. The importance of timing

One of the reasons operational transfers are tricky is because of the various timing issues that affect DNS operation. In DNS operation without DNSSEC, there are two kinds of timing issue: those having to do with the operation of the authoritative DNS servers, and those having to do with the effects of caches (and therefore, the visibility of changed data). The parameters controlling these timings are discussed in detail in [4].

Data in the DNS mostly travels in sets of resource records, or "RRsets"[4]. There are two exceptions to this rule.

The first is in the transfer of zone data from one authoritative server to another using AXFR[2] and IXFR[5], where entire versions of a zone (as distinguishable by the SOA serial number) are transferred in a single operation. While these protocols do indeed work by moving individual RRsets between the source server (the master) and the client requesting the transfer (the slave), all of the RRsets become visible in the slave at the same time. The protocols do not allow for partial transfer of zone data.

The second is where DNSSEC is involved. Under DNSSEC, the RRSIG for the RRset travels with the RRset. So in one sense, it behaves like any other member of the RRset. Caches hold the value of the RRset for the time defined by the TTL field on the RRset[5]. But a cache might have originally asked for the RRset with DO=0, which would mean that the RRSIG

[2]The exception, of course, is a domain that is used to provide name service to other domains.

[3]To see why this is so, a simple example will illustrate. Suppose the operator of example.com wants to add a new name server (NS) record to the apex of the zone. The NS set also needs to appear in the parent at the other side of the zone cut in order to effect the delegation. The new name server should be inserted into the apex of the example.com zone first, because if the delegation point offers a nameserver that is not actually authoritative for the zone, resolution will not work as expected. But as a result, there is a short time of inconsistency necessary for this operation.

[4]This is sometimes spelled "RRSet", as in [1], which also clarified the notion of the RRset.

[5]Strictly, the TTL is defined by each resource record. [1] requires that all TTLs in an RRset be the same, and also requires any client to treat the TTL on an RRset as the lowest value of those TTLs.

would not be in the cache with the rest of the RRset. In this sense, the RRSIG is separable from the rest of the RRset. This has implications when we turn our attention to performing operations transfers in the presence of DNSSEC.

C. Who must act when transfers are performed: no DNSSEC

To get a complete description of how to transfer DNS operation, we must first work out exactly who must perform each action. For the purposes of this description, let us imagine Adam is a registrant of example.tld. Adam has registered example.tld in a three-party registration model. He used RegistrarCo to be his registrar, who registered the name with the .tld registry.

When we begin, Adam has a contract with Bernice to run the DNS for example.tld. Bernice accepts changes to the DNS zone data from Adam and publishes them. Let us suppose that Adam is not sophisticated with the DNS, so every change to the DNS has to be performed by his operator.

Adam decides to hire Charlie to run the DNS instead. So, Adam undertakes an agreement with Charlie. Adam also informs Bernice that Charlie is authorized to get copies of the example.tld zone. Now Charlie gets a copy of the zone, and starts keeping up to date with the zone data. Charlie's servers answer authoritatively for the zone, but are not in the apex NS set.

Now one of two things happens. Either Charlie asks Bernice to add his name servers to the apex NS set for the zone, or else Charlie logs in to Bernice's system and adds the NS records himself. In either case, Charlie's name servers become part of the apex NS set. No later than this point, change control over DNS data for the zone must no longer reside with Bernice.

Next, Charlie either contacts RegistrarCo on behalf of Adam (having been somehow authorized to do so), or else Adam contacts RegistrarCo directly. Whoever does this adds Charlie's nameservers to the nameserver set for example.tld. RegistrarCo contacts the TLD registry, and adds Charlie's nameservers to the NS records for example.tld. At this point, Charlie's nameservers are active on the Internet for example.tld, but so are Bernice's.

Next, Charlie contacts RegistrarCo on behalf of Adam, or Adam contacts RegistrarCo himself, and removes Bernice's nameservers from the set for example.tld. RegistrarCo then updates the TLD registry.

Under the above series of steps, Bernice can actually stop responding to DNS requests around the time Charlie's nameservers make it into the TLD registry, because there is for the most part enough data in the system to cause queries to be directed towards Charlie if Bernice does not respond or if she responds with a non-authoritative answer. (In fact, for some clients such action may result in a transient outage, but the outage may well be short enough that nobody notices.) Indeed, a problem sometimes observed is that former DNS providers do not stop responding in a timely way, continuing to answer queries as though they are authoritative some time after the customer has left them behind. This can be a problem because of the way some resolvers behave.

Note that none of the above has attended at all to the possibility that the service that depends on the DNS has to move too – as, for instance, when web service and DNS service are offered as part of a single "package deal".

D. Child-centric and parent-centric resolvers

Adding further complication to the process of transferring DNS operation from one operator to another is the variation in behavior among of recursive resolvers widely deployed in the Internet. The variation relates to different strategies for renewing a record after its TTL has expired. Caching nameservers have two strategies for getting data from the DNS. Recall that the NS RRset for a domain is present in two places: on the parent side of the zone cut, at the delegation point; and on the child side, at the apex of the zone in the authoritative server. When resolving names inside a zone, some resolvers prefer to use the first NS RRset they get for that zone, and always ask a nameserver in that RRset when resolving requests in the zone (for as long as the NS RRset remains in cache). Since the first nameserver that will be encountered is usually from the delegating nameserver – the parent – we call these resolvers "parent-centric," to distinguish them from the other cases. (We are unaware of any resolvers that actually follow this approach consistently.) Other resolvers, upon seeing the NS RRset that came from the authoritative servers for the domain – the child – will replace the NS set from the parent in their cache, and prefer the child-provided NS RRset. We call these resolvers "child-centric"; Unbound is an example of this strategy. Note an important effect here: in the event the NS RRset is different at the child and parent, the different kinds of resolvers will decide to query different sets of nameservers.

A subclass of child-centric resolvers are ones that opportunistically extend the TTL of an RRset whenever they see the NS RRset for the zone in authoritative responses to other queries. In the case of high volume recursive resolvers, e.g. resolvers at ISPs serving a lot of customers, this "TTL stretching" might continue indefinitely, thereby preventing a TTL expiration indefinitely. We call such resolvers "sticky child-centric" resolvers. BIND 4 exhibits this behaviour. This strategy works well in the usual case where the NS RRset is indeed the same and the TTL just needs to be refreshed, but it does not work well if the name server RRset has changed. In fact, if the parent is now pointing to another set of name servers but the old name servers are continuing to answer, the recursive resolver will never learn the name of the new name server. However, if the old name server stops answering queries, the child-centric resolver will eventually go back to the parent (albeit after a bit of pause before it resolves the query).

E. Why DNSSEC matters for the transfer

RRSIG records are generated with a particular set of keys, and those keys need to be available for validation. Moreover, in order to validate the delegation from the parent, a DS record in the parent needs to match one of the DNSKEY records in the child. These two facts make DNS operator transfers in the presence of DNSSEC more delicate, because transfers that used to work in the face of a sloppy procedure will suddenly fail to work: either signatures will be provided that cannot

be checked with any of the available keys, or else the apex DNSKEY RRset will not contain a key that corresponds to the DS record from the parent. In either case, the domain will validate as bogus, and validating resolvers will stop processing the name.

In the pre-DNSSEC case, when disagreeing data from different sources are received, resolvers can try to make the best of the situation, and return whatever result seems most likely to allow resolution to happen. But because this operational expedient is indistinguishable from an attack on the DNS messages, DNSSEC will not permit it. This suggests that coordination between the gaining and losing operator of the DNS for the target zone need to co-operate. But of course, the losing operator has little incentive to undertake the expense of contributing to the solution when the only reward is a lost customer.

F. Why not go insecure?

One answer to the complications above is to say that the registrant should simply take the domain through an insecure step, removing the DS record from the parent for a time and thereby stopping validation at the parent in a provably insecure way. But that approach implies that the purpose of DNSSEC is only to secure the traditional uses of the DNS. We can already tell that this will not be the case, as efforts to leverage DNSSEC to build new security models for the online world are well underway. A domain that relies on DNSSEC in order that its SSL keys are useful will not be able to go through an insecure step without giving up all those extra advantages. If we believe that DNSSEC is a path to new security applications on the Internet, then "go insecure" is not good advice.

G. How DNSSEC affects the required behaviour of the actors

In III-C, we provided an outline of who must do which steps. An important feature in that series is that change control over the zone is required to move from one actor to another. But under DNSSEC, the new operator (Charlie in our example) also needs to provide signatures over the data in the zone.

We work under the assumption that private keys cannot (or at least in practice will not) be transferred from one operator to another. This means that the data in the zone operated by Bernice is signed by Bernice's private key, and Charlie cannot get that key. So Charlie has to introduce his own key to the DNSKEY set. In effect, a key rollover has to be performed during the transition period. Charlie adds his key to the DNSKEY set at the same time his nameservers are added to the NS RRset. This ensures that the additions are signed in such a way that they are validatable at the moment they are added.

Once Bernice gives up change control on the zone, her DNSKEY record remains in the zone. In addition, a DS record in the parent corresponding to her key remains in the parent. Charlie now signs the DNSKEY RRset with his key, and adds a DS record corresponding to his key to the parent. At this point, no matter whether a querying client queries Charlie or Bernice, the answer is still validatable.

Once Charlie removes Bernice's nameservers from the NS RRset, however, things begin to change. As long as the DS record in the parent corresponding to Bernice's key remains, then answers from Bernice will still validate. Once the DS record is removed, then answers from Bernice will be treated as bogus. So, when the DS record corresponding to Bernice's key is removed at the parent, Bernice must stop answering queries for example.tld. Otherwise, child-centric resolvers will treat those answers as bogus.

IV. Conclusion: Next Steps

Having determined who each of the actors are, the next task is to outline what each actor must do and when he or she must do it, and to describe the minimal co-operation that must be expected from any DNS operator at handover time. That description can serve as a basis for future operator contracts that define the responsibilities of the various parties.

References

[1] R. Elz and R. Bush. Clarifications to the DNS Specification. RFC 2181 (Proposed Standard), July 1997. Updated by RFCs 4035, 2535, 4343, 4033, 4034, 5452.

[2] E. Lewis and A. Hoenes. DNS Zone Transfer Protocol (AXFR). RFC 5936 (Proposed Standard), June 2010.

[3] P.V. Mockapetris. Domain names - concepts and facilities. RFC 1034 (Standard), November 1987. Updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592, 5936.

[4] S. Morris, J. Ihren, and J. Dickinson. Dnssec key timing considerations. draft-morris-dnsop-dnssec-key-timing-02.txt, March 2010: work in progress.

[5] M. Ohta. Incremental Zone Transfer in DNS. RFC 1995 (Proposed Standard), August 1996.