Deploying and Monitoring DNS Security (DNSSEC)

Eric Osterweil
UCLA
eoster@cs.ucla.edu

Dan Massey
Colorado State University
massey@cs.colostate.edu

Lixia Zhang
UCLA
lixia@cs.ucla.edu

Abstract—**SecSpider is a DNSSEC monitoring system that helps identify operational errors in the DNSSEC deployment and discover unforeseen obstacles. It collects, verifies, and publishes the DNSSEC keys for DNSSEC-enabled zones, which enables operators of both authoritative zones and recursive resolvers to deploy DNSSEC immediately, and benefit from its cryptographic protections. In this paper we present the design and implementation of SecSpider as well as several general lessons that stem from its design and implementation.**

I. INTRODUCTION

The DNS Security Extensions (DNSSEC) [7], [9], [8] add much needed security to the critical DNS [17] system, and deployment efforts started a few years ago. Although the total number of secure zones is still quite small, it has been growing steadily. Perhaps more importantly, there has been considerable activity at critical top level DNS zones. A number of country code top level domains (ccTLDs) have been signed (in fact some have been signed for a few years) and the number keeps growing. The U.S. Government has deployed DNSSEC in the "`.gov`" zone and all federal agencies are required to sign their "`agency.gov`" zones by the end of 2009. Furthermore, the "`.org`" zone has also deployed DNSSEC recently. Other top level domains and the root zone itself have announced plans to deploy DNSSEC. At the same time, the increasing awareness of DNS vulnerabilities has also further motivated sites to consider DNSSEC deployment. We believe that the DNSSEC deployment has reached the tipping point and all organizations should at least be aware of DNSSEC's impact on their sites, and ideally should start considering how to integrate DNSSEC into their operations.

DNSSEC is the first fully distributed cryptographic system to be rolled out in the Internet to support one of its core systems (the DNS). Due to its cryptographic underpinnings, the DNSSEC deployment faces not only those challenges that all conventional distributed-systems share, but also many new types as well. In anticipation of unforeseen deployment obstacles, we developed the SecSpider DNSSEC Monitoring System (http://secspider.cs.ucla.edu) in 2005 and have been operating this system since then. SecSpider is a scalable system whose distributed polling design and large-scale DNSSEC survey corpus allows it to provide operators with key operational data and global views of both their own sites and sites that their users may access frequently. This allows sites to learn from the current deployment, assess their own needs, and assess the success (or challenges) in any experimental deployment. Some of SecSpider's key discoveries are presented in Section V. These discoveries help underscore the critical need for distributed monitoring in order to discover and diagnose both known and unknown problems.

As of this writing, SecSpider has been an operational tool for the DNSSEC community for over 3 years. Our website has already become a useful part of many operators' practices as evidenced by the fact that it served over 2.3 million page views in 2008 alone. To help the current and future users of SecSpider get acquainted with the system, this paper describes its design and implementation, exposes some insights learned through the system's design and rollout, and illustrates some of the system's key results. Our objective is to explain how SecSpider operates for sites already using it for essential monitoring, and for sites that are considering using it in near future. Section II provides some background on DNSSEC. Section III describes the SecSpider design and Section IV presents the underlying implementation details. Finally, results obtained using SecSpider and information for potential DNSSEC operators are summarized in Section V.

II. DNS AND DNSSEC BACKGROUND

The Domain Name System (DNS) maps hostnames such as `www.ucla.edu` to IP addresses, as well as provides a wide range of other mapping services ranging from email to geographic location. In this section we introduce a basic set of DNS terminology which is used throughout the text, including resource records (RRs), resource record sets (RRsets), and zones, followed by an overview of the DNS Security Extensions.

Security was not a primary objective when the DNS was designed in mid 80's and a number of well known vulnerabilities have been identified [11], [10]. DNSSEC provides a cryptographic solution to the problem, which seems pretty simple and intuitive. To prove that data in a DNS reply is authentic, each zone creates public/private key pairs and then uses the private portions to sign data. Its public keys are stored in a new type of RR called `DNSKEY`, and all the signatures are stored in another new type of RR called `RRSIG`. In response to a query, an authoritative server returns both the requested data and its associated
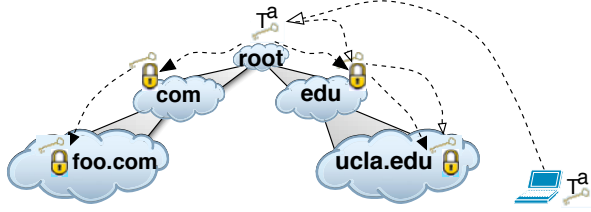
IEEE
computer
society

Figure 1. Resolvers preconfigure the root zone's public key as a trust anchor (T^a) and can then trace a "chain of trust" from that key down the DNSSEC hierarchy to any zone's key that they have encountered.

RRSIG RRset. A resolver that has learned the DNSKEY of the requested zone can verify the *origin authenticity* and integrity of the reply data. To resist replay attacks, each signature carries a definitive expiration time.

In order to authenticate the DNSKEY for a given zone, say www.ucla.edu, the resolver needs to construct a *chain of trust* that follows the DNS hierarchy from a trusted root zone key down to the key of the zone in question (this is shown in Figure 1). In the ideal case, the public key of the DNS root zone would be obtained offline in a secure way and stored at the resolver, so that the resolver can use it to authenticate the public key of edu; the public key of edu would then be used to authenticate the public key of ucla.edu.

There are two challenges in building and maintaining the chain of trust. First, a parent zone must encode the authentication of each of its child zone's public keys in the DNS. To accomplish this, the parent zone creates and signs a Delegation Signer (DS) RR that corresponds to a DNSKEY RR at the child zone, and creates an authentication link from the parent to child. It is the child zone's responsibility to request an update to the DS RR every time the child's DNSKEY changes. Although all the above procedures seem simple and straightforward, one must keep in mind that they are performed manually, and people inevitably make errors, especially when handling large zones that have hundreds or thousands of children zones. Moreover, such manual coordination is done across different administrative domains, making it more error prone.

In addition, the parent and child zones belong to *different* administrative authorities, each may decide independently if and when they turn on DNSSEC. This leads to the second and more problematic challenge. If the parent zone is not signed, there is no chain of trust leading to the child zone's DNSKEY. This orphaned key effectively becomes an isolated trust anchor for its subtree in the DNS hierarchy. To verify the data in these isolated DNSSEC zones, one has to obtain the keys for such isolated trust anchors offline in a secure manner. DNSSEC resolvers maintain a set of well-known "trust-anchor" keys (T^a) so that a chain of key sets + signatures (secure delegation chain) can be traced from some T^a to a DNSSEC key K lower in the tree. The original

DNSSEC design envisioned that its deployment would be rolled out in a top-down manner. Thus only the root zone's K would need to be configured in all resolvers' T^a sets and all secure delegations would follow the existing DNS hierarchy. However as of this writing, neither the root zone nor most of the Top Level Domains (TLDs) are signed, and it is unclear when a meaningful portion of the DNS TLDs will be. Without the root and top level domains deploying DNSSEC (as is the case today) there could be potentially millions of isolated trust anchors. In fact various approaches have been proposed for securely obtaining these trust anchors and are summarized in other work [22].

In addition to origin authenticity and key learning, DNSSEC also specifies a way to cryptographically deny the existence of records. This *authenticated denial of existence* is done when the zone is being signed. Each domain name has an NSEC record associated with it. This NSEC record specifies what the next domain name is after the current one, and what type of RRs exist for it. This allows a resolver to see that the name of the NSEC comes (canonically) before the query name, but it points to a name that comes *after* the query name. Thus, NSEC records allow resolvers to use the attached RRSIG records as proof that they're query does not exist. However, one side effect of this mechanism is that starting with the zone's name (the apex), a resolver can recursively ask for the next name in the zone until it sees the final NSEC record loop back to the apex. This is called NSEC walking and makes an entire zone's contents visible. A new solution called NSEC3 has been developed recently that avoids zone enumerations while still providing authenticated denial of existence [16].

III. THE SECSPIDER DNSSEC MONITORING SYSTEM

SecSpider aims to discover and address challenges faced by both the operators of secure DNS zones and the operators of secure resolvers. In order for SecSpider to provide meaningful monitoring results, we must first define what it means to *be* a DNSSEC-enabled zone; next, we must *find* DNSSEC-enabled zones to examine; and finally, we must be able to provide constructive feedback to operators so that problems can be addressed and fixed.

A. Defining DNSSEC-Enabled

The latest round of DNSSEC specifications [7], [9], [8] outline many behaviors that DNSSEC-enabled zones must adhere to. Though they start with serving keys and signatures, they go a fair distance further. SecSpider checks to see if zones serve their public keys in the DNS resource records called DNSKEYs. Then, each RRset that is returned to resolvers must be accompanied by one or more RRSIG records *and* those records must be verifiable by at least one of the DNSKEYs being served. Next, in order to provide *secure denial of existence*, DNSSEC-enabled zones must serve either valid NSEC [9] or NSEC3 [16] records whenever

a resolver queries for a name that does not exist. This allows resolvers to use DNSKEYs to cryptographically prove that a zone does not have the specific record(s) requests.

Finally, each zone is served by a *set* of name servers. In order for a zone to be considered DNSSEC-enabled, *all* of its name servers must pass these tests. The rationale behind this is that if only one name server fails to properly serve DNSSEC and a resolver happens to issue its queries to *that* name server, the resolver will likely not know to try DNSSEC queries to others in the set and just be unable to use DNSSEC. Given a zone name, these characteristics are relatively easy to verify using a sequence of DNS queries. However finding all the DNSSEC-enabled zones is an non-trivial task, as we explain next.

B. Finding DNSSEC Zones

There are numerous ways to go about building a corpus of DNSSEC zones to monitor. Our primary way to discover DNSSEC-enabled zones is letting users submit their zones directly to the SecSpider website. In addition, search engines have made a science out of crawling over the World Wide Web and learning of new web sites. As an obvious side effect, they learn of DNS zones. Thus, we routinely crawl a set of DNS zones discovered by a commercial search engine [18] to discover if any zones have recently enabled DNSSEC. Furthermore, from our corpus of DNSSEC zones, we are able to perform NSEC walking to determine if any of them have DNSSEC-enabled children. Our last process of finding DNSSEC-enabled zones comes from several DNS monitors that exist in various locations hosted by our affiliates. These monitors continuously crawl DNS name space search for zones that serve DNSKEYs (just the first of our requirements above). When these monitors find such a zone, they begin monitoring that zone. Section V shows the current number of DNS zones monitored by SecSpider and how they were obtained.

C. Looking For Trouble

Administering a DNSSEC zone requires some additional operational tasks beyond those needed to administer plain old DNS zones. For example, it is important for a zone's administrator to know she i) is maintaining a proper secure delegation from her zone's parent, ii) has a DNSKEY set that is "reasonable" in size, iii) is properly executing DNSKEYs rollovers, and iv) is not unnecessarily introducing replay vulnerabilities. By contrast, from the perspective of DNS resolver operators (who might also operate authoritative zones of their own), it is important to be able to *use* DNSSEC by learning verified DNSKEYs for zones. SecSpider is designed to meet the needs of both types of operators.

Operators can navigate from SecSpider's homepage to their own zone's drill-down page and use the drill-down page to see the status of their secure delegations (DS record) from their parent. This is designed to help detect whether a stale secure delegation exists (for example if the cryptographic data at the parent zone does not match the public key at the child zone). A number of operators have reported that this SecSpider feature has helped them correct problems ranging from errors in secondary servers not providing valid data to issues where servers provided data but resolvers could not retrieve the data due to issues with large key set sizes. When the size of the DNSSEC key set is larger than the Path Maximum Transmission Unit (PMTU) value, it leads to the DNS responses being lost. SecSpider was the first to discover and report the PMTU problems related to DNSSEC [21], [4], [6], and this discovery has subsequently pervaded the DNSSEC community [23], [5]. One example of action that has resulted from this is that organizations like the National Institute for Standards and Technology (NIST) have adjusted their recommendations from larger keys whose lifetimes are longer (less frequent rollovers) to smaller keys that are rolled over more frequently.

In general, identifying when there is an availability problem with a zone's DNSKEY set size is more challenging than simply determining when the zone itself is meeting the DNSSEC requirements. Different data delivery paths will allow different maximum size messages. What makes the size of a key set problematic is a combination of the path that DNS data may take through the Internet and the various "middle boxes" (NATs, proxies, VPNs, etc.) encountered along that path. Thus, the distributed nature of the problem suggests that a distributed monitoring system is needed to detect potential failures. Accordingly, SecSpider is not a single site, but rather it gathers data from a widely distributed set of pollers (which issue DNS queries) in different continents and different networks. Distributed monitoring is critical, because by issuing queries from multiple vantage points, SecSpider increases the chance that it will see any latent PMTU problems. Operators can then keep track of their zones' statuses on SecSpider's website to see if they need to take any action.

SecSpider tracks all of its known DNSSEC zones for consistency over time and across its multiple polling locations to detect vulnerabilities in DNSSEC deployment such as key rollover errors and records that are vulnerable to replay attacks. While capturing *all* of the RRsets for a zone would allow an exact account of its vulnerability, simply tracking its signing practices and some of its crucial infrastructure records (i.e. NS and glue records) is sufficient to indicate *when* a zone is experiencing commonly seen cryptographic operational problems. For example, tracking the proper execution of DNSKEY rollovers is done as a subset of SecSpider's vulnerability monitoring. DNSKEYs are among the types of records being tracked by SecSpider. By observing and storing past record sets and signatures, SecSpider also automatically tracks potential replay vulnerabilities for a zone's infrastructure records such as NS records, DNSKEY records, and A records associated with name

servers. In early deployment a few years ago, SecSpider began prominently featuring replay vulnerability statistics and this coincided with a measurable change in signature lifetimes and reduction in vulnerable records[19].

SecSpider summarizes the list of `DNSKEY`s that is sees *consistently* across all of its pollers into an observed DNSSEC keys file. Whenever a zone is scheduled to be polled, SecSpider issues its queries from all pollers. This means that in order for an adversary to spoof their way into SecSpider's observed keys list, they would have to fool all of its pollers positioned all around the world at the precise time they query a specific zone. Moreover, they would have to do so for all name servers of the target zone. Based on DNS best practices, zones should be served by at least 2 name servers in separate locations. The implications of SecSpider's observed keys file is that it reflects the operational reality of which keys are in use consistently and correctly.

D. Design Requirements

Based on SecSpider's roles as both a monitoring system and an operational tool for zone operators and resolver operators, several key design requirements have emerged: i) clearly the system must meet the scalability needs of DNSSEC's target deployment, ii) monitoring a distributed system (like DNSSEC) is best done by a distributed network of monitors, iii) SecSpider must be able to capture and diagnose any complex distributed behaviors of DNSSEC, iv) in order to continue to be able to diagnose unforeseen issues, the design must support a model in which diagnostics and enhancements can evolve along with the discoveries of new problems, and v) SecSpider, itself, must be robust against failures and its components (including operators that use its data) must have trusted channels to coordinated with each other over.

IV. THE SECSPIDER IMPLEMENTATION

A monitoring system for DNSSEC must be robust and secure itself. The SecSpider system's design, implementation, and deployment have produced both a useful resource and a set of more general design lessons.

SecSpider's implementation uses a 3-tier architecture in which a single master coordinates a globally distributed network of polling daemons (pollers). The first tier (front end) is responsible for serving read-only content to users, the middle tier (coordinator) performs scheduling and analysis, and the third tier is a backend database (Figure 2).

Through the design and evolution of SecSpider, several fundamental design insights have been gained that pertain to distributed monitoring of DNSSEC, as discussed in Section III-D.

A. Front End

SecSpider's front end presentation is primarily done via replicated instances of simple Apache webservers and DNS
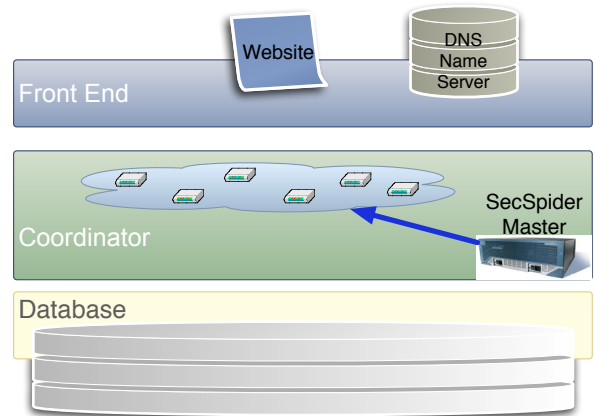

Figure 2. The 3-tier architecture of SecSpider for scalability and isolation.

name servers that run in FreeBSD jails [14]. The website remains an effective and highly accessible mechanism for inspecting SecSpider data. However, over the four year deployment, we continue to experiment with other distribution mechanisms. For example, `secspider.cs.ucla.edu` DNSSEC name servers in California and Colorado can provide some additional `DNSKEY` verification data in the form on a new type of RR, called the `DLV` record [26]. Currently, we are exploring a new peer to peer distribution mechanism that is planned to be available in late fall 2009.

Regardless of the actual SecSpider information distribution mechanism, the front end tier is restricted from communicating with the other tiers. All of the components of this tier serve read-only data to users, and do not have any online communication channel with SecSpider's backend processing. Thus, all data that needs to be presented is pushed to these front end boxes and served directly from them. The only channel of communication from the front end to the other components is the user submitted zone names. In other words, a user may request SecSpider to add a particular zone to its monitored set and this in turn populates a local front-end repository. The middle-tier (coordinator) is not notified of or interrupted by submissions, and instead simply polls the front-end repository for submissions when the coordinator is ready. This design illustrates that the monitoring and processing duties of a system (especially a security system) can be, and perhaps should be, implemented as isolated and atomic services whenever possible. The application of this principle has allowed SecSpider's front end to decouple the scaling requirements of the presentation layer from the polling and analysis that is done in the other tiers. If the system becomes subject to an attack or an intrusion the backend can remain unaffected.

B. Coordinator

This component is the heart of SecSpider. The coordinator is so named because it interfaces with SecSpider's set of distributed pollers and "coordinates" the process by which they interrogate and measure various aspects of each zone. In particular, every query issued by SecSpider ultimately begins at the coordinator.

Active Polling: When a polling cycle starts, SecSpider creates a new schedule of the zones it will poll. It does this to reduce the odds that an adversary can observe SecSpider, guess its query pattern, and attempt to coordinate spoofed replies. After generating the query schedule for a run, SecSpider must poll each zone from all of its pollers in near synchrony; otherwise data inconsistencies are likely to happen if the polling by different pollers is spread out over time. Because SecSpider makes a conservative assumption that inconsistencies in the results obtained by different pollers signal potential faults or attacks, the polling policy attempts to minimize its own bias by reducing the time between polls from different locations.

The polling itself is not as simple as issuing a set of DNS queries and then recording the answers. As described in Section III, all of the name servers for each zone are probed to determine if they meet the requirements of being DNSSEC-enabled. Furthermore, when queries by some, but not all, pollers fail, SecSpider must retry them with different parameters; we discuss this further below. Because of the state needed for each zone on each poller, SecSpider uses separate threads for each zone from each poller.

SecSpider uses a *queue-poller-queue* design (seen in Figure 3) to schedule zones from the backend, issue polling queries, and store the results back to the backend. The first queue allows scheduling to ramp up at its own pace, the polling is an orchestrated set of queries to distributed pollers that can take its own time, and the second queue allows polling results to be serialized at whatever pace the backend can support. The choice of a queue-poller-queue model illustrates a common property of distributed polling systems and a useful approach; polling is an inherently I/O bound process that can often proceed in fits and starts. A decoupled design such as the queue-poller-queue model has allowed SecSpider to maintain the greatest amount of polling throughput that both the backend and the polling can jointly accommodate. If, for example, a poller is slow to respond, the system will eventually time it out, but the design schedules its queries at the granularity of zones. Thus, all queries to any zone's name servers will be issued at the same time, and a slow poller will not cause a head of line blocking problem.

After the polling schedule is created, each zone object is entered into the *loader queue*. Tasks from this queue are consumed by a pool of threads. Each SecSpider poller is assigned to several threads in the pool. In order to ensure
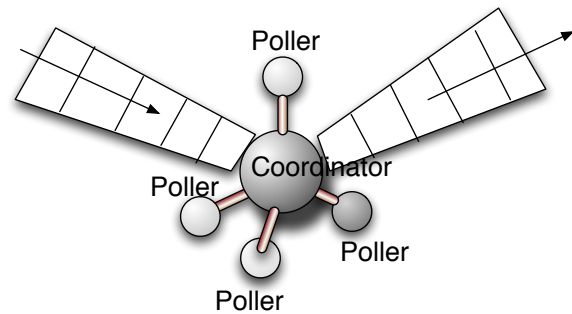

Figure 3. The queue-poller-queue model is one in which producers and consumers of queue have a loosely-coupled relationship to the polling work.

that a zone is polled at the same time from each poller, tasks that are dequeued from the loader queue are then split amongst one thread from each poller and their polling is started at the same time. The reasons for actually splitting the zone out and using threads instead of simple event loops is that each zone may seem quite different from each poller's vantage point.

When querying for a zone's DNSKEYs, one might naively assume that the data for that zone would be equally accessible from any vantage point. However, this is not always the case. In particular, some pollers can have notably different response rates. We have observed that when a poller has difficulty querying for a DNSKEY, it can manifest itself in one of two ways: i) a *truncated* response is received[1] or ii) the query times out and no response is received. Truncated messages are anticipated; they are designed to prompt resolvers retry their query either over TCP or with a larger message size. However, persistent unresponsive behavior is not. DNSSEC timeouts can result from unforeseen interactions within the Internet and the Path Maximum Transmission Unit (PMTU).

As we discussed in Section III-C, one important recent observation is that messages can be dropped if they are too large. In this case, the server replied correctly and fit the DNS response inside the resolver's specified maximum packet size, but the network between the server and resolver (e.g. the poller) did not permit this large packet size and the response does not reach the poller. When SecSpider sees that a poller has been unable to get a DNSKEY it performs a *PMTU walk* where varying maximum query message sizes are tried to determine what the proper size should be. We discuss this further in Section V.

After each zone has been polled (and potentially PMTU walked) from each poller, it is enqueued in SecSpider's *dumper queue*. This allows polling to occur at its own rate

---

[1]Note that DNS messages can indicate that the nameserver only had room to send *some* of the data it wanted. This is indicated to resolvers by setting the TC bit in the DNS message header.

while serialization and storage to the backend can be done asynchronously from the dumper queue.

After the zone polling has concluded, SecSpider re-examines each zone in order to logically build the secure delegation hierarchy. SecSpider uses the existing DS records (and checks to see if they verify properly) to identify where valid secure delegations exist, and what the roots of the observable islands of security are. This allows people to observe DNSSEC's rollout state, but also lets operators check that their delegations are verifying properly.

The last stage in SecSpider's active polling is NSEC walking. As described in Section II, each zone forms a chain of NSEC records that point from one domain name to the "next" canonical name in the zone, *and* specify what types of records exist for that name. Though this was done so that a name server could provide secure denial of existence, it allows us to start at one name and learn all of the other names and record types in the zone. When there are no more names, the last valid name in the zone points back to the beginning (the zone apex). SecSpider walks over each zone's names looking for new DS records for child zones. Whenever a name exists with a DS record, it indicates that the zone has a secure delegation for that name. Of course, this doesn't always mean there actually is a valid zone attached to that delegation. So, SecSpider uses that name to check if there is a zone there that is DNSSEC-enabled. If so, then the zone is permanently added to the polling corpus. Though this seems very simple and straight forward, there are some (largely undisclosed) complexities involved with NSEC walking in practice.

While the format and protocol surrounding NSEC records makes the concept of walking very clear, certain operational practices make it difficult to do this at any significant scale. One of the first issues with NSEC walking is that many zones (especially ccTLDs) are quite large. Walking their entire set of domains takes a non-trivial amount of time. Another and more subtle limitation of blindly NSEC walking is that some name servers host both a parent *and* its children zones. The implication for NSEC walking is that one name may be in a parent, then the next would be the apex of a child. After reaching the end of the *child* zone, the NSEC will point back to its own apex (not the apex the NSEC walker started with). Thus, the NSEC walk can get caught in a loop (seen in Figure 4) and simply fail to ever walk passed the first child zone. Because of operational realities like these, SecSpider's NSEC walking randomly jumps forward by creating a random domain name that would be canonically sorted *after* the current domain name and using the random name in the next query. During this entire process, SecSpider sets a time limit on each zone and after any walk has exceeded that time, SecSpider moves to the next zone. Therefore, each walk is a random sampling of a large zone's records, and over time SecSpider visits an increasing number of them.
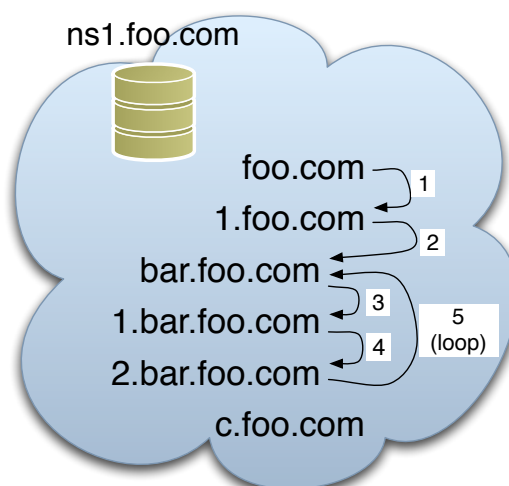

Figure 4. When a name server hosts a parent and any of its children zones, NSEC walking will loop when the child zone loops back to its own apex rather than continuing with the parent zone's names.

Polling Infrastructure: Each of SecSpider's pollers is a lightweight DNS repeater called rdnsD [24]. These pollers accept DNS queries from SecSpider over a secure channel and then re-issue them either to their local resolver, or to a name server that SecSpider specifies. The coordinator simply issues a DNS query to the desired poller, with a DNS OPT resource record code in the message. This code tells the poller where to re-send the message. The poller simply strips the OPT code from the query and issues it to the specified server. This allows SecSpider to query specific authoritative DNS name servers from any of its polling vantages. The communications between the coordinator and all of its pollers (seen in Figure 5) are secured by using TSIG [25] symmetric authentication (one of DNS' standard symmetric key facilities). Each poller uses a separate TSIG key to authenticate its message exchanges with the SecSpider's coordinator. This ensures that an adversary can neither send her own messages through SecSpider, nor spoof messages that belong to SecSpider once polling has begun.

This design illustrates a fundamental tradeoff one faces when designing a distributed polling system: a more central design may choose to hold query logic close to a central coordinator and keep remote simple *thin-client* pollers (as SecSpider does), or distribute the query logic to smarter *fat-client* pollers. The SecSpider design has demonstrated some tangible benefits gained from the thin-client approach. A fat-client approach might seem more appealing from, for example, an efficiency point of view because the coordinator might issue a single request to "begin polling" and all DNS queries could then be issued directly from each poller and results summarized before being sent back to the coordi-

Figure 5. The locations of SecSpider's current pollers.

Figure 6. The ERD of SecSpider's database.

nator. This design would minimize round trips between the coordinator and its pollers. On the other hand, the thin-client approach requires one round trip from the coordinator to the poller and another round trip from the poller to a name server for each query, and this can add noticeable network overhead when compared to a fat-client system. However, monitoring systems like SecSpider are used to discover, diagnose, and illustrate unforeseen problems with an evolving deployment system. Thus, the polling patterns needed by SecSpider are constantly evolving. For example, before the PMTU problem was identified, SecSpider only issued a single generic DNSKEY query for each of a zone's name servers. After discovering the PMTU problem, SecSpider began using a finite state machine to determine details of how to issue DNSKEY queries with different buffer sizes for different name servers of the same zone, using both UDP and TCP, and more. Having a thin-client system allowed this change to be easily implemented, tested, and deployed at the coordinator without any upgrade or change to the remote pollers that are hosted by different organizations. Thus, in the face of an ever-changing set of requirements, the thin-client approach has allowed SecSpider the greatest agility in discovering and addressing new problems.

Content Generation: After the active polling has finished, SecSpider generates the content that it will push to its front-end. It is important to note that SecSpider's data is cryptographically signed and verified by its various components starting when its pollers receive answers from name servers until a user downloads it. The DNS messages (including queries from the coordinator and responses sent by the pollers back to the coordinator) use TSIG cryptography, the coordinator makes a local secure connection to the backend database, DNSKEY data served through web site is signed by SecSpider's operational GPG key, and all DLV records are signed and verifiable by the dlv.secspider.cs.ucla. edu DNSKEY. Thus, the security between each component of SecSpider speaks to the general principle that as components become loosely coupled, it becomes increasingly important to secure their communications against any intervention
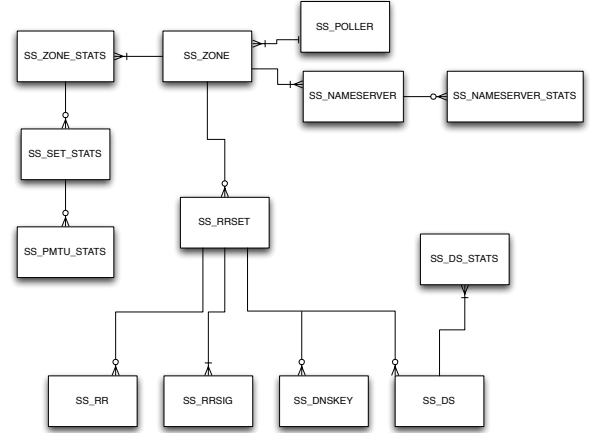
(such as insider threats or compromised components).

After SecSpider performs internal verification of the retrieved data, it creates a drill-down page for each monitored DNSSEC zone that lists many monitoring statistics, among those are the list of the zone's DNSKEY records and its DS record(s). These are presented in HTML, but also in separate flat files. These files include the actual RRSIG values, and a list of which pollers observed the values. The purpose of these files is to let operators and scripts easily download the records for any given zone, and easily see whether any discrepancy exists among the records observed by all the pollers. If an adversary spoofs the results for a subset of pollers, the discrepancy will reveal the attacks. Even if an adversary manages to spoof the results of all the SecSpider pollers, given the data and values are all publicly available, vigilant zone owner can still notice the attacks by looking up SecSpider results periodically.

Furthermore, all of the keys that are seen consistently across SecSpider's pollers and across long enough time periods are then entered into SecSpider's global trust-anchors file. This is one of SecSpider's main contributions. The implications of SecSpider's distributed framework, and cryptographic checks are that it can furnish resolvers with verified DNSKEY records in a readily usable form of trust-anchors *today*. Furthermore, SecSpider uses its GPG key to cryptographically sign each of these flat files so that if someone is interested in verifying the origin authenticity they can do so.

C. Database

SecSpider stores all of its data in a MySQL relational database. The schema is split into two major types of table: i) DNS tables and ii) statistics tables. The Entity Relationship Diagram is seen in Figure 6.

DNS tables are designed so that all of the DNS RRsets and RRs that SecSpider tracks are categorized as either being

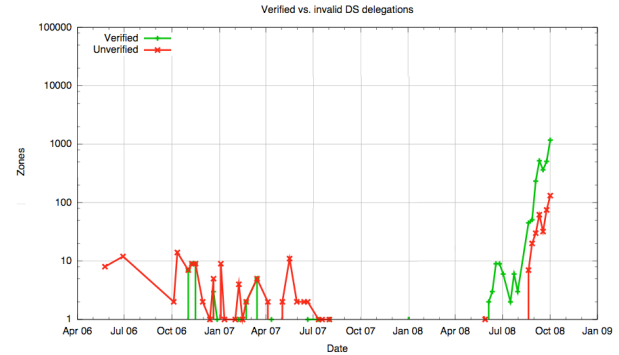| Submission Source	Percent
User Submissions	30.79%
DNS Monitors	51.28%
Web Crawling	10.75%
NSEC Walking	7.18%

Table I

Figure 8. This Figure shows that as the number of secure delegations has grown (the upper curve), so too has the incidence of misconfigured secure delegations (the lower curve).

verifiable DNSSEC records or not. Verifiable records are those with valid (i.e. not expired) RRSIG records attached. These are records that SecSpider uses to maintain the state of DNSSEC-enabled zones. Other records are considered as un-verified and are only used for historical measurements and analysis.

Statistic tables track various types of data that are observed during each run. A good example is the SS_NAMESERVER_STATS table in which each row has an observation date and tracks what a name server reports as its version number[2], and statistics such as whether it is DNSSEC-enabled. These tables are all transactional (meaning they only grow over time).

D. Implementation Summary

As of this writing, SecSpider has been an operational tool for the DNSSEC community for over 3 years. Our website [1] has already become a useful part of many operators' practices as evidenced by the fact that it served over 2.3 million page views in 2008 alone. We currently track approximately 18,000 DNS zones in our database. SecSpider pollers operate at NL Net Labs (Netherlands), Colorado State University (US - Central), Tsinghua University (China), Cable Modem in Los Angeles (US - West), Toshiba Corporation (Japan), Switch (Switzerland), and Telx (US - East). SecSpider's backend database currently uses several tables with multi-millions of rows. To date, the statistics describing the name servers of zones, the availability of DNSKEY RRsets, and the transitions that zones make between DNSSEC-enabled and not have generated tables with 17 million, 15 million and 17 million records respectively.

V. SECSPIDER MONITORING RESULTS

SecSpider has an increasingly long history of providing operational insights into DNSSEC's behavior [2], [27], [4]. In addition to that, SecSpider's data has also demonstrated its usefulness to academic research [21], [19], [20], [15].

Recent events such as [12] have served as timely forcing-functions for DNSSEC's deployment. As of this writing we

have begun to see an unprecedented level of adoption for DNSSEC. Table I shows the break down of how SecSpider has learned of DNSSEC zones through its different discovery channels. However, more interesting is Figure 7 which shows that after the summer of 2008 (when the [12] security advisory was issued), the adoption rate of DNSSEC has significantly increased. This makes it a critical time for tools and observations to inform operators and augment DNSSEC in places where its design is failing. One such place is in the context of how resolvers securely learn the *correct* DNSKEYs for the zones they encounter.

SecSpider's data set has also proven useful in tracking operational practices surrounding the secure delegation hierarchy. This was discussed during a recent operation presentation [3]. Figure 8 shows that recent increases in the number of secure delegations have corresponded with increases in misconfigured secure delegations (the lower curve) as well.

Perhaps one of the most significant findings from SecSpider's measurement apparatus to date is the quantification and demonstration of the previously unobserved PMTU problem that DNSSEC faces. When a timeout occurs for a DNSKEY query, SecSpider presumes a large message size resulted in a PMTU problem[3]. A recent study [13] done at one of the DNS root servers found roughly half of the global DNS traffic observed requested message sizes of 4,096 bytes. Therefore, SecSpider begins with this same value. If that value causes a failure, SecSpider then retries the query (from the same poller) with half the message size. SecSpider uses this process to increase and decrease message sizes (following a binary search, depicted in Figure 9) until it determines what the proper request size *should* be in order to get an un-truncated message in one try.

While our pollers now have logic added to resolve a PMTU issue, a typical resolver does not. If the resolver uses

---

[2]Users can query name servers for a record called VERSION.BIND of type CH, and by convention many name servers will report their software name and version number.

[3]Note, SecSpider checks name servers to ensure they are online and reachable before querying for any DNS data.
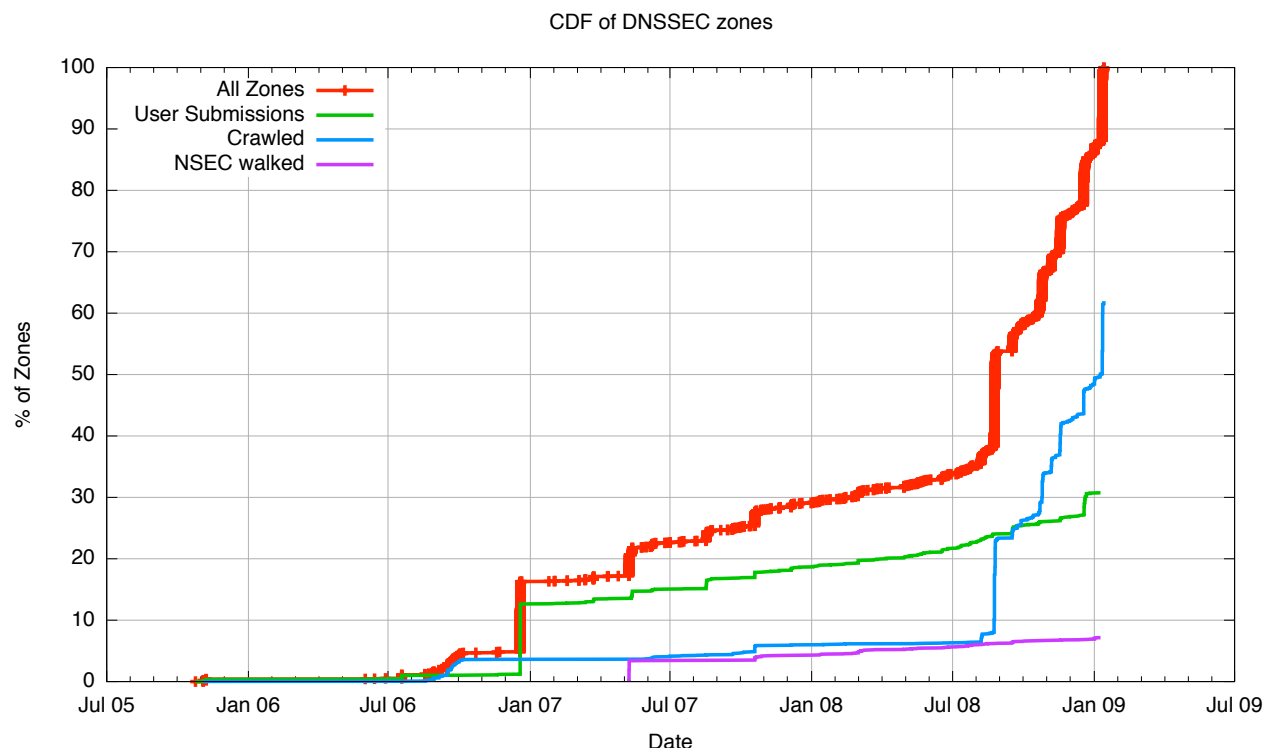
Figure 7. Here we see that the adoption rate of DNSSEC has dramatically increased since the summer of 2008.

a maximum message size that the path between it and server will not support it does not know that it is *falsely* advertising a maximum message size that the *network path* cannot support. This is further complicated because the maximum message varies depending on both which server the resolver is querying and which path routing provides to that server. During a PMTU failure, a resolver receives no response and therefore will *not* fallback to TCP and generally will *not* try a different message size. Thus, some resolver/server pairs effective DoS themselves and the corresponding DNSSEC zones appear unreachable. Without an external monitoring project such as SecSpider, the problem is hard to detect.

Zone administrators cannot control the PMTU and the PMTU varies depending on the resolver location. However, the zone administrator can learn that PMTU issues are occurring with SecSpider pollers. A response to a `DNSKEY` query message is often the most problematic response. Some zones will create responses that contain 4 public keys and 4 signatures. Contrast this with the response for a typically IP address query whose answer would contain a small number of `A` (IPv4) or `AAAA` (IPv6) records and signatures. The net result is that selecting a large key set may simplify some key rollover operations and selecting large key sizes may increase cryptographic security, but the cost is message size and if the message is too large some portions of the Internet will be unable to resolve data from the zone.
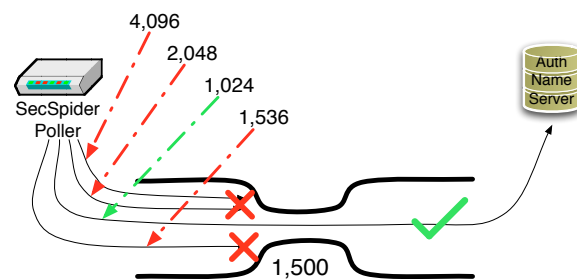

Figure 9. Here one can see how the message size begins at 4,096 when the PMTU is actually 1,500. After a failure, 4,096 is halved to 2,048 which also fails. The next unit is 1,024, which succeeds. The binary search then continues to search for the exact value and increases the message size until the exact size is found.

SecSpider illustrates this on zone drill-down pages so that administrators can be aware of the problem and consider the trade-offs between the benefits of large key sets and the benefits of global reachability.

VI. CONCLUSIONS AND IMPLICATIONS FOR OPERATORS

We feel that operational evidence supports our view that a distributed global monitoring has been, and will continue to be, a pivotal part of ensuring the success of the DNSSEC deployment. The above results are just a

sample of the how monitoring plays an essential role in the deployment of DNSSEC. The data is freely available and widely used in the DNSSEC operational community. The data from SecSpider is influencing operational practices and helping inform the larger DNS operation community by both identifying challenges and highlighting successes. Individual zones interested in deploying DNSSEC are encouraged to track their progress on SecSpider. Drill-down pages provide a site with a snapshot of how their zone is viewed from multiple location around the world. SecSpider also provides one with a view of secure sites that your resolvers may be accessing. If a site is having trouble obtaining secure records a particular zone, SecSpider can be used to determine how other resolvers view the same data.

REFERENCES

[1] SecSpider. http://secspider.cs.ucla.edu/.

[2] Secspider: Distributed dnssec monitoring. In *NANOG 44 Tools BoF*, 2008. http://www.nanog.org/meetings/nanog44/abstracts.php?pt=ODg1Jm5hbm9nNNDQ=\&nm=nanog44.

[3] The state and challenges of the dnssec deployment. In *NANOG 44 DNSSEC BoF*, 2008. http://nanog.org/meetings/nanog44/presentations/Sunday/Osterweil_DNSSEC_N44.pdf.

[4] Availability problems in the dnssec deployment. In *RIPE 58*, 2009. http://www.ripe.net/ripe/meetings/ripe-58/content/presentations/dnssec-deployment-problems.pdf.

[5] Draft secure domain name system (dns) deployment guide. In *NIST Computer Security Division*, 2009. http://csrc.nist.gov/publications/drafts/800-81-rev1/nist_draft_sp800-81r1-round2.pdf.

[6] Network path problems in dnssec's deployment. In *IETF 75 - dnsext*, 2009. http://www.ietf.org/proceedings/75/slides/dnsext-3.pdf.

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

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

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

[10] D. Atkins and D. Austein. Threat Analysis of the Domain Name System (DNS). RFC 3833, August 2004.

[11] Steven M. Bellovin. Using the domain name system for system break-ins. In *Proceedings of the Fifth Usenix Unix Security Symposium*, pages 199–208, 1995.

[12] CERT. Cert vulnerability note vu#800113, 2008.

[13] The Measurement Factory. Dns survey: October 2007, 2007. http://dns.measurement-factory.com/surveys/200710.html.

[14] P. Hope. Using Jails in FreeBSD for Fun and Profit. *Login: The Magazine of Usenix & Sage*.

[15] Andrew J. Kalafut, Craig A. Shue, and Minaxi Gupta. Understanding implications of dns zone provisioning. In *IMC '08: Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*, pages 211–216, New York, NY, USA, 2008. ACM.

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

[17] P. Mockapetris and K. J. Dunlap. Development of the domain name system. In *SIGCOMM '88*, pages 123–133, 1988.

[18] A. Ntoulas, G. Chao, and J. Cho. The infocious web search engine: improving web searching through linguistic analysis. In *WWW '05: Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 840–849, 2005.

[19] Eric Osterweil, Daniel Massey, and Lixia Zhang. Observations from the DNSSEC Deployment. In *The 3rd workshop on Secure Network Protocols (NPSec)*, 2007.

[20] Eric Osterweil, Vasilis Pappas, Dan Massey, and Lixia Zhang. Zone state revocation for dnssec. In *LSAD '07: Proceedings of ACM Sigcomm Workshop on Large Scale Attack Defenses*, 2007.

[21] Eric Osterweil, Michael Ryan, Dan Massey, and Lixia Zhang. Quantifying the operational status of the dnssec deployment. In *IMC '08: Proceedings of the 8th ACM SIGCOMM conference on Internet measurement*. ACM, 2008.

[22] Eric Osterweil and Lixia Zhang. Interadministrative challenges in managing dnskeys. *IEEE Security & Privacy*, 7(5):44–51, 2009.

[23] Scott Rose. Revised nist guidance for zsk sizes. http://mail.shinkuro.com:8100/Lists/dnssec-deployment/Message/2328.html?Language=.

[24] NetSec Colorado State. rdnsd polling. https://www.netsec.colostate.edu/dnsmonitor/secure.html.

[25] P. Vixie, O. Gudmundsson, D. Eastlake 3rd, and B. Wellington. Secret Key Transaction Authentication for DNS (TSIG). RFC 2845, May 2000.

[26] S. Weiler. DNSSEC lookaside validation (DLV). RFC 5074, SPARTA Inc., November 2007.

[27] Duane Wessels. Kaminsky vulnerabilty leads to increased dnssec adoption. https://www.dns-oarc.net/node/156.