The Futility of DNSSec

Alex Cowperthwaite
School of Computer Science
Carleton University
Ottawa, ON Canada K1S 5B6
acowpert@ccsl.carleton.ca

Anil Somayaji
School of Computer Science
Carleton University
Ottawa, ON Canada K1S 5B6
soma@ccsl.carleton.ca

***Abstract*—The lack of data authentication and integrity guarantees in the Domain Name System (DNS) facilitates a wide variety of malicious activity on the Internet today. DNSSec, a set of cryptographic extensions to DNS, has been proposed to address these threats. While DNSSec does provide certain security guarantees, here we argue that it does not provide what users really need, namely end-to-end authentication and integrity. Even worse, DNSSec makes DNS much less efficient and harder to administer, thus significantly compromising DNS's availability—arguably its most important characteristic. In this paper we explain the structure of DNS, examine the threats against it, present the details of DNSSec, and analyze the benefits of DNSSec relative to its costs. This cost-benefit analysis clearly shows that DNSSec deployment is a futile effort, one that provides little long-term benefit yet has distinct, perhaps very significant costs.**

I. Introduction

The Domain Name System (DNS) has become an vital part of the modern Internet. DNS translates human readable host names such as `www.carleton.ca` to machine routable IP addresses such as `134.117.1.32`. These operations are performed automatically as part of nearly every transaction across the Internet today.

DNS was designed in the early 1980's to be fast, efficient, and highly available in the face of network disruptions [1], [2]. These were all essential design goals given the limited and unreliable nature of the computational and networking resources at their disposal. To achieve these goals DNS used a distributed hierarchical database, a simple query response protocol, and aggressive caching with permissibly stale data. With the rapid and global growth of the Internet, the DNS system has scaled very well and maintained its core attributes of efficiency and high availability; unfortunately, it has also maintained its lack of data authenticity and integrity. These omissions have become increasingly significant as malicious activity has increased on the Internet. A falsified DNS reply can redirect an unsuspecting user to an attacker-controlled host, making them vulnerable to phishing, malware distribution or numerous other malicious activities.

DNS Security Extensions (DNSSec) addresses DNS's lack of data authentication and integrity. DNSSec provides these guarantees by using public key cryptography to sign each DNS record. DNSSec has been carefully designed to be as efficient and scalable as possible; as we will explain, however, the bandwidth, computational, and administrative overhead of DNSSec do potentially compromise DNS's essential availability properties. More significant, however, is that DNSSec solves the wrong problem: it secures hostname to IP address mappings when what is really needed is better end-to-end security guarantees. Thus we believe that the deployment of DNSSec is a futile gesture, one that will lead to minimal long-term security benefits while resulting in significant security and economic costs.

We proceed to make this argument in the rest of this paper as follows. First, we describe the Domain Name System in more detail in Section II. Next, we discuss cache poisoning attacks, the most serious current threat to DNS, in Section III. Section IV presents DNSSec and explains how DNSSec can block cache poisoning attacks. We then explore how DNSSec reduces the availability of DNS in Section V; Section VI discusses how attackers could circumvent the protections provided by DNSSec. Section VII presents potential alternatives to DNSSec adoption while Section VIII discusses more general issues relating to DNSSec and end-to-end security. Section IX concludes.

II. The Domain Name System

The Domain Name System is a distributed database for storing information on domain names, the primary namespace for hosts on the Internet. While the primary purpose of DNS is to map domain names to IP addresses, it actually supports several kinds of records. The primary record types are as follows:

- **A** records specify an IPv4 address for a host.
- **AAAA** records specify an IPv6 address for a host.
- **NS** records point to the authoritative name servers for a zone (domain).
- **CNAME** records allow the specification of host aliases, e.g. make `www.example.com` refer to `berners-lee.example.com`.
- **MX** records identify mail servers that can receive email for a domain.

The DNS system is composed of two types of systems, name servers and resolvers. Name servers return information about domains or zones when queried by DNS resolvers. DNS resolvers send queries to one or more name servers in order to service DNS requests from applications.

When a name server is configured as authoritative for its zone, its resource records are always interpreted as correct.

To improve availability and reduce upstream server load, resolvers may also cache resource records until they have expired; while some domains specify time-to-live values on the order of minutes, it is more common for domains to permit their records to be cached for days. Cached records are not considered to be authoritative, i.e., they may contain incorrect or stale information; most consumers of DNS information, however, do not distinguish between authoritative and non-authoritative results.

Applications normally send queries to simple stub resolvers implemented as library functions. Stub resolvers forward requests to standalone recursive resolvers that perform the multiple name server queries normally required to answer any request. Recursive resolvers work best when most responses can be answered using cached queries (i.e., they see multiple requests for the same information). ISPs normally operate DNS servers with recursive, caching resolvers for their customers; alternately, users can configure their systems to connect to other open DNS servers.[1]

DNS name servers are structured as a hierarchical tree where the tree levels correspond to the dot-separated components of a domain name. A complete lookup follows a chain starting from the DNS root servers to a top level domain (TLD) down to any number of domains and subdomains below. We review the steps in a typical DNS lookup below.

1) An application (e.g., a web browser) invokes a local stub resolver to look up the IP address of `www.example.com`.
2) The stub resolver sends the request to a local DNS server (a recursive, caching DNS resolver).
3) The DNS server will check its cache for an entry for `www.example.com`. If it is does not have an entry it will request a lookup from a randomly chosen root server. The IP addresses of standard root servers are built in to DNS servers.
4) The root server will reply with the NS record describing the name of the authoritative name server for the Top Level Domain (TLD) `.com`. To reduce traffic, it will also append "the glue": the A record (IPv4 address) for the `.com` TLD name server.
5) The recursive DNS resolver will request a lookup for `www.example.com` from the `.com` authoritative name server (as specified in the returned glue A record).
6) The `.com` name server will reply with the authoritative name server record for `example.com`. It will also append the A record for the `example.com` authoritative name server.
7) The recursive DNS resolver will then ask the authoritative name server for `example.com` for the lookup of `www.example.com`
8) The name server for `example.com` will reply with an authoritative answer for `www.example.com`, likely only an A record.
9) Optionally, the recursive DNS resolver will add or update this entry in its cache.
10) The recursive DNS server will then return a non-authoritative answer (an A record) to the requesting application's stub resolver.

Each DNS query and response is normally sent in a single UDP packet. The query and response use the same structure; the query is sent with the response field blank. A 16 bit query ID field used to distinguish outstanding requests [5], [6].

Note how multiple characteristics of DNS make it both efficient and highly available. Queries are simple and small (there are few record types and the amount of information in each is small), minimizing bandwidth and storage overhead. Full and partial results are easily cached (i.e, servers cache results from every level of the hierarchy). And, DNS's database is distributed in a simple, scalable way that is (relatively) easily debugged and tuned—i.e., if you don't get an appropriate response, you can quickly identify what server gave you incorrect information.

III. DNS Cache Poisoning

While DNS has many positive characteristics, note that DNS has no integrity, authenticity, or confidentiality guarantees. Confidentiality can be added by encrypting queries (e.g., via IPsec); integrity and authenticity cannot be so easily added, however, because of the distributed nature of DNS. DNS responses can be modified or completely forged almost trivially by an intruder-in-the-middle attack. Indeed, because DNS normally uses UDP, attackers do not even need to hijack TCP connections. The weaknesses of DNS are actually even worse, however, because the existing DNS infrastructure can be used to deliver malicious responses.

Specifically, malicious responses can originate from legitimate, otherwise uncompromised servers through the use of cache poisoning attacks [7]. With standard DNS cache poisoning attacks, a victim can be redirected to a malicious host through the injection of a forged A record. The key idea behind such attacks is that a caching resolver has no way to authenticate received data; it will accept any query response that "looks right." An attacker even has many chances to generate authentic-looking responses as unexpected responses will simply be discarded.

While such attacks are problematic enough, in the summer of 2008 Dan Kaminsky described a direct extension to known DNS cache poisoning techniques that made them much more dangerous and reliable. Kaminsky's primary observation was that instead of spoofing the final A record of the destination, an attacker could spoof any one of the query responses in the lookup process. By forging an NS record, an attacker could take over an entire domain, up to and including a TLD such as `.com`.

Kaminsky also improved the reliability of cache poisoning attacks. Previously, if an entry was cached attackers would have to wait for the cached entry to expire before they could attempt to replace it; thus, it was relatively hard to poison

[1] Traditionally recursive resolvers and name servers were both implemented as part of the same application (e.g., BIND 9 and earlier[3]); Newer DNS server designs, however, divide these functions into separate programs (e.g., `tinydns` and `dnscache` of djbdns [4]).

the cache of popular domains. Kaminsky, however, noted that by querying an obscure subdomain not already cached (i.e. `reallyobscuresubdomain.example.com`) an attacker could force a lookup. The lookup would go directly to the authoritative namesever for the domain, assuming it was cached. This creates an opportunity for an attack to forge a reply with no answer for the obscure subdomain but include an additional update for the A record of the authoritative nameserver. The opportunistic caching resolver will update their cache with the false record. This method allows an attacker to repeat the attack many times giving them a greater chance of returning an answer that will be accepted, successfully poisoning the cache. [8]

The easiest to implement defense against these cache poisoning attacks is to make responses harder to predict. Clearly the 16-bit query ID field does not have enough entropy to prevent brute force attacks. This entropy can be supplemented by randomizing the outgoing request port. If every available port could be used, that would give us 32 bits (as UDP ports are 16 bit quantities); while we cannot reach this upper bound in practice, we can get close enough to make cache poisoning attacks much harder. Even with such improvements , however, it is still possible to mount DNS cache poisoning attacks. To prevent forged requests, ideally we need the kind of authentication guarantees that are best provided by cryptographic mechanisms. DNSSec was designed to provide precisely these guarantees.

IV. DNSSEC

DNSSec is a set of extensions to DNS that are designed to authenticate and protect the integrity of DNS responses through the use of public key-based digital signatures. The design of DNSSec started in the mid-1990s, motivated by the growing recognition of the dangers of DNS cache poisoning [7]. The first version of DNSSec was officially published as RFC 2065 in January 1997 [9]; an improved version followed in RFC 2535 (March 1999)[10]; after trial deployment experience, the current version of DNSSec was finalized in RFC 4033-4035 in March 2005 [11], [12], [13].

DNSSec allows domain owners to digitally sign resource records. One key design decision in DNSSec is that name servers do not implement any cryptography; signatures are generated offline, while signatures are checked by resolvers. While this design choice minimizes the computational overhead of DNSSec, it also greatly complicates the process of authenticating DNS responses because the results returned by every authoritative name server in a query chain must be individually verified. Thus, for resolving `www.example.com`, the responses from the root, `.com`, and `example.com` name servers must each be authenticated independently.

The information necessary to authenticate responses is stored in a set of new DNS record types:

- **RRSIG** records contain digital signatures for other resource records. Each regular DNS resource record should have its own RRSIG record (except for the DNSKEY record).
- **DNSKEY** records contain public keys for verifying RRSIG records. Note that a zone may have multiple DNSKEY records.
- **DS**, or designated signer records, store cryptographic hashes of the public keys of a zone's children. DS records are what allow `.com` to specify what key should sign `example.com`'s records, thus enabling trust chains. DS records should be signed with a corresponding RRSIG record.
- **NSEC and NSEC3** records provide authenticated denial of existence. NSEC provided this information by specifying two hostnames between which no other hostnames exist. NSEC allows attackers to enumerate the hosts in a domain; NSEC3 [14] addresses this problem by specifying the ranges in terms of hashes of domain names rather than the domain names themselves.

To authenticate a resource record, a resolver first checks that the RRSIG signature on the requested record was generated by the zone's key, as specified by its DNSKEY record. To make sure the DNSKEY record has the right value, the corresponding DS record of the zone's parent is checked to see if it has the hash of the DNSKEY key. Then, to verify the DS record is genuine, its RRSIG has to be checked against the parent zone's DNSKEY public key. This process repeats until we get to the root zone; here, we assume the resolver already knows the authentic public keys belonging to the root name servers.

Note that DNSSec clearly defeats cache poisoning attacks. Responses can only be forged if the underlying cryptography is broken. Because resolvers will only forward or cache authentic responses, the resolver's cache would remain uncorrupted. Unfortunately this benefit comes along with significant costs.

V. DNSSEC AVAILABILITY ISSUES

While DNSSec does provide strong integrity and authenticity guarantees for DNS responses, it also requires significantly more computer resources and requires a significant investment in key management. Both of these factors reduce DNS's availability and so counterbalance the potential security gains from DNSSec.

A. Resource Usage

DNSSec significantly increases the computational and space footprint of DNS. DNSSec requires additional computation for generating and verifying records. These costs are either offline or are borne by requesting clients and/or their local resolvers; as such, cryptographic overhead shouldn't be a problem for most DNSSec deployments.

The most obvious source of increased space usage with DNSSec is the use of additional records: every regular record requires a RRSIG signature, along with at least one nameserver-specific DNSKEY and verifying DS (from the parent) records. While the space impact of this increase isn't too large given the relatively small size of DNS zone files, these additional records must also be communicated, increasing DNS's bandwidth requirements significantly.

	No.of packets	Total Size (bytes)	Amplification factor
DNS Reply	1	208	2.9
DNSSec Reply (actual)	1	1215	17.3
DNSSec Reply (expected)	1	2257	32.2
DNSSec Reply Chain (actual)	5	3045	43.5
DNSSec Reply Chain (expected)	5	11865	169.5

Fig. 1. Amplification relative to one 70 byte DNS query packet for `www.dnssec.se`.

Given today's computers and networks, this increased resource consumption should be very manageable under normal circumstances. Root and TLD DNS servers, however, have been and will continue to be targets of distributed denial-of-service (DDoS) attacks [15], [16]. The increased bandwidth requirements of DNSSec will require significant hardware, especially bandwidth, upgrades to maintain the same level of DDoS resistance. Any outage or latency at the root or TLD servers will be felt through out the internet.

The increased bandwidth consumption is particularly worrisome because of its asymmetric nature: small DNS requests can result in large DNSSec replies. This characteristic makes DNSSec-enabled systems more attractive for DNS DDoS amplification attacks. Such attacks happen when a DDoS attack is passed through DNS servers to amplify the volume of data being sent to a target. Amplification attacks are feasible because it is so easy to forge the source address of UDP packets; thus, any open resolver can be used to send packets to almost any host on the Internet [17], [18].

To evaluate the possibility of the amplification attack, we recorded DNS lookups using Wireshark and the DNS lookup utility `dig` (See Figure 1). We looked up the host `www.dnssec.se` using the L root server and the open `.se` DNSSec test server. We chose these servers because they offer some of the strongest DNSSec support at the time of this writing. We found that many servers do not fully implement DNSSec as RFC 4033–4035 specify, so we report both the actual values observed on the wire and the expected values based on the RFCs. Specifically, we manually reconstructed the full chain of DNS replies that is supposed to be returned when the checking disabled (CD) flag is set. From our data it is clear that DNSSec greatly increases the opportunity for an amplificaion attack.

DNS amplification attacks can be mitigated in multiple ways. DNS resolvers can be shielded from external requests through firewall rules or through IP address-based request filters on DNS servers themselves. Unicast Reverse Path Forwarding (URPF), defined in RFC 3074 [19], limits the scope of IP address spoofing. Such defenses, however, have only been partially delpoyed; further, even where delpoyed limited forms of IP address spoofing are still possible. With DNSSec deployment attackers will need far fewer DNS amplifiers to perform potent denial of service attacks, making DNS amplification attacks that much more feasible.

While neither the increased DDoS susceptibility and the increased potential for DNS DDoS amplification attacks should, by themselves, stand in the way of DNSSec deployment, they are both clear drawbacks to a transition to DNSSec.

B. Key Management

We think a more significant DNSSec disadvantage will be key management in practice. DNSSec has a number of provisions designed to make key management easier. It allows a zone to have as many signing keys (DNSKEY records) as they wish, and those keys can be divided into separate functions, e.g., Zone Signing Keys (ZSK) and Key Signing Keys (KSK). The fundamental fact is, however, is that DNS administrators will have to become public-key infrastructure (PKI) administrators as well. They will have to secure private signing keys, decide who has access to them, and generate signatures for new and old information as records change and as keys expire. DNS administrators are already busy people; certificate management is a new, non-trivial responsibility for which many (arguably most) DNS administrators do not have the background, training, or experience. The most obvious potential issue with such inexperienced DNSSec administrators is that they will store and use keys insecurely, making themselves vulnerable to attack. While such direct attacks may be a concern, we think such issues will be minor compared to two others: DNSSec-caused denial of service and false alarms.

Ideally, DNSSec would be used such that any signature verification error would cause a DNS lookup to fail. If DNSSec is deployed this way, however, any key management or signature generation error would result in a denial of service. Sign using the wrong key—site goes down. Sign with an expired key—site goes down. Forget to update signatures when a key expires—site goes down as well. Anecdotally we know users occassionally experience certificate verification errors with SSL [20]; when every domain on the Internet must manage keys, and when every hostname must be appropriately signed, user-visible key management errors will inevitably go up dramatically.

We hypothesize that these errors will be so common that it will not be feasible for DNSSec to cause DNS queries to fail; instead, DNSSec authentication errors will be used to generate warnings that users or administrators can choose to follow or ignore—just as with SSL certificates. And, as with SSL certificates, most users will choose to ignore these warnings [21].

In other words, key management with DNSSec will be problematic enough that either users will have to tolerate a significant loss in system availability due to non-malicious authentication failures, or they will become accustomed to ignoring DNSSec warnings, eliminating any security benefit DNSSec provides.

VI. Circumventing DNSSec Security

When properly deployed, DNSSec does guarantees integrity and authenticity of DNS data. Even if we assume that DNSSec is used optimally, however, that does not mean that we get what we really want, namely secure communications. Attackers have two basic strategies for circumventing DNSSec: they can attack from below or from above.

Attackers can get "below" DNSSec by targeting other aspects of network communication. DNS, at best, tells a computer the IP address of another computer. An attacker who mounts an intruder-in-the-middle attack (using, say, ARP spoofing on a local network [22] or BGP route hijacking [23], [24]) can potentially take over any IP address they desire. Thus, DNS data can be completely secure yet attackers can still make victim machines connect to malicious hosts. Targetting the actual IP communications with the host instead of the DNS lookup, an attacker easily circumvents any security gained through DNSSec.

Of course, all this assumes that the DNS hostname we are attempting to resolve *is the right hostname*. Phishing attacks have demonstrated that users are easily fooled into visiting malicious sites via links containing fraudulent hostnames [25]. Users do not understand the significance of hostname components and their ordering. Indeed, why shouldn't a regular user visit a URL with `www-mybank.secure.com` when their bank is really located at `www.mybank.com`? A proper explanation requires a basic understanding of DNS—something that we cannot expect regular users to know about.

DNSSec is a (potentially) strong defense surrounded by insecurity both above and below. While DNSSec deployment might reduce the incidence of DNS-specific attacks, we cannot see how it would have a significant impact on the number or severity of security incidents on the Internet. It is simply too easy for an attacker to use other means to get systems and users to connect to malicious hosts.

VII. Beyond DNSSec

While there are alternatives to DNSSec such as DNSCurve [26] that are more efficient and that arguably have better security properties, to address the above concerns we need to realize that DNS, by itself, is not the security problem that we need to address. What is needed are better ways to provide end-to-end security: when a person uses a computer to interact with a remote host, we need ways to authenticate that remote host to defend against intruder-in-the-middle attacks and attacker attempts to masquerade as that remote host. DNSSec does not provide end-to-end authentication because IP-level traffic is not authenticated and because people can easily confuse malicious domain names with legitimate ones (or, they'll just ignore malicious domain names).

Today we have two technologies in wide use for providing end-to-end authentication. One, SSL/TLS [27] as it is implemented on the web, authentication occurs using trust chains grounded in public keys bundled with web browsers. The other is Secure Shell (SSH) [28], where the public keys of remote hosts are either cached or supplied by an administrator. With SSL, a third party (a certificate authority), generally unknown to a user, vouches that the host being accessed is the one denoted by its domain name (e.g., Verisign says that we are connecting to `www.bankofamerica.com` when browsers visit that domain), and with the new extended validation certificates, that third party says what organization controls that server (e.g., Verisign says that `www.bankofamerica.com` belongs to Bank of America Corporation). Alternately, with SSH on our first connection we cannot authenticate the host; however, on subsequent connections SSH authenticates the remote host by correlating the domain name and server-supplied public key with the key information that was saved previously. So long as you connect to the same hosts you have in the past, you are protected from attackers masquerading or intercepting traffic at the network level.

SSH gives us easy-to-use end-to-end authentication (and more) so long as we connect to the same remote hosts; it provides no authentication for connections to new hosts. SSL gives us remote host authentication so long as we trust the certificate authority keys included with our browsers, we verify the domain name we are visiting is, indeed, the correct one, and we cancel connections to hosts that generate certificate errors. Of course, all three of these assumptions are not actually true: sometimes certificate authorities are organizations that many people distrust [29], [30] and users regularly ignore malicious URLs and certificate errors [25]. As we discussed earlier, however, these issues also apply to DNSSec or, indeed, any scheme that relies on the DNS namespace and arbitrary third-party signers of keys.

Both SSH and SSL are built on top of DNS and they both provide the authentication DNS lacks. Even better, they provide host authentication, not host IP address authentication, although with some issues. Thus, it is reasonable to argue that SSH and SSL together provide remote host authentication for all users who want or need such authentication, and they do so in a way that also provides end-to-end security, thus obliviating the need for DNSSec. Yet neither address the bigger problem of bridging the gap between hostnames and the remote host (and organization) a user actually intends to access.

VIII. Discussion

The Domain Name System is efficient, fault tolerant, and fundamentally insecure—much as is the case with TCP/IP. Both are foundational technologies for the Internet. Since the mid-1990's, many researchers have worked on developing secure variants of both protocols, namely DNSSec and IPsec; neither of these more secure proposals have been widely deployed. Note that while DNSSec by itself can be circumvented by IP-level attacks, IPsec and DNSSec together provide a very secure foundation. So, why haven't they been widely deployed?

Part of the reason is simple laziness (in the most reasonable sense): both of these transitions would involve significant ongoing cost because of the burden of key management, and without widespread deployment their security benefits are minimal. The more fundamental reason, though, is that complete

layer-level changes are simply infeasible today on the Internet. They may happen over time, but only if circumstances are just right. The normal case is to expect fragmentation—some will adopt the new technology while others won't. IPv6 and software patches are in the same boat as DNSSec and IPsec—some will deploy but all will not. In this environment, the more a solution requires universal or near-universal acceptance to provide benefits, the more likely it will never provide those benefits.

While DNS cannot provide end-to-end security without a cryptographic extension such as DNSSec, we could make DNS immune to cache poisoning attacks, the main source of DNS insecurity, with a variety of relatively minor changes (e.g., by increasing the size of the query ID field from 16 to 64 bits)—that is, assuming that the existing best practice of randomizing source ports and query IDs is not sufficient protection. To get end-to-end security, we simply need to encourage more sites to adopt SSL or, as appropriate, SSH. SSL is already ubiquitous and SSH provides benefits even in limited deployments.

Some advocates of DNSSec are arguing that it should be adopted not because of the security guarantees it provides, per se, but because of the new, more secure applications that could be built on top of it [31], [32]. But this argument assumes widespread deployment, something that has not happened and may never happen in a way that allows other applications to be built on top of it. Rather than waiting for a technology that has been in development for fifteen years to suddenly take off, why not use existing technologies to achieve the same ends?

But pushing for better end-to-end security at the DNS level (using DNSSec, SSL, or SSH) misses the larger security issues of authenticating destinations for network communications, particularly web-based communications initiated by users. The fundamental issue here is that the hostnames namespace is not one that regular people understand. People understand brands and, to a lesser extent, company names. Given the numerous and still growing number of top-level domains, there exists no clean mapping between brands, companies, and hostnames. Many Internet users have discovered a very simple strategy for dealing with this confusion: they do not access sites via domain names and URLs; instead, they use search engine queries. Such use of search engines can sometimes lead to users accessing the wrong site (this happened with Facebook recently [33]). Accessing hosts via search engines is potentially insecure because there is no guarantee that the link the search engine returns is the one the user actually wants; however, this method is fast, relatively reliable, and understandable to end users. If we wish to secure Internet communications, we must take into account how computer users actually behave, not how we would like them to behave.

What we need are ways for users to specify remote hosts precisely and securely, using a namespace that does not require a computer science degree to understand and properly use. We then need a way to handle authentication failures in a way that people are more likely to follow, not ignore them. These are hard questions, ones that need further research.

The arguments we have made against DNS and DNSSec have been made separately by many others; our contribution here is to bring them together and to put them in perspective versus the real security problem of end-to-end security. For administrators, we hope this work will help them better evaluate security technologies such as DNSSec. Also, we hope that once researchers see the limited benefits and the significant drawbacks of DNSSec that they will move on to developing incrementally deployable solutions that address the true end-to-end security needs of users on today's Internet.

IX. Conclusion

DNSSec is designed to improve Internet security by providing authentication and integrity protection for DNS. While DNSSec would potentially protect against DNS cache poisoning, it does so at the cost of increased susceptibility to DDoS attacks, increased utility for DDoS amplification attacks, and significant human administrator overhead for key management. The key management issue is particularly important because poor key management (which we expect to be the rule, not the exception) will result in denials of service or, more likely, in users being trained to ignore DNSSec-based warnings.

These significant tradeoffs take place against a backdrop of other, simpler methods for defending against cache poisoning attacks such as source-port randomization and DNSSec's inability to provide end-to-end authentication. While DNSSec could be extended to provide such protection, we already have solutions that provide essentially the same guarantees: SSL and SSH.

The true futility of DNSSec, however, arises from the fact that it addresses the wrong problem. Users do not understand the semantics of DNS well enough to distinguish between malicious and legitimate hosts. While extended validation SSL certificates are a small step in the right direction, interface issues [34] and user reliance on search engines instead of URLs [33] make them a partial solution at best.

Rather than adopt DNSSec, we argue that the Internet community should invest in making DNS more robust to cache poisoning attacks without relying upon a PKI infrastructure while SSL and SSH should be more widely adopted. Meanwhile, research is needed in ways to close the gap between user perception and remote host authentication.

Acknowledgements

We wish to thank the members of Carleton Computer Security Laboratory and the anonymous reviewers for their suggestions.

This work was supported by Canada's Natural Sciences and Engineering Research Council (NSERC) through the Internetworked Systems Security Network (ISSNet) and Discovery Grant programs.

References

[1] P. Mockapetris, "Domain names: Concepts and facilities," RFC 882, Internet Engineering Task Force, Nov. 1983. [Online]. Available: http://www.ietf.org/rfc/rfc882.txt

[2] ——, "Domain names: Implementation specification," RFC 883, Internet Engineering Task Force, Nov. 1983. [Online]. Available: http://www.ietf.org/rfc/rfc883.txt

[3] I. S. Consortium, "Isc bind." [Online]. Available: https://www.isc.org/software/bind

[4] D. J. Bernstein, "dnjdns." [Online]. Available: http://cr.yp.to/djbdns.html

[5] P. Mockapetris, "Domain names - concepts and facilities," RFC 1034 (Standard), Internet Engineering Task Force, Nov. 1987, updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592. [Online]. Available: http://www.ietf.org/rfc/rfc1034.txt

[6] ——, "Domain names - implementation and specification," RFC 1035 (Standard), Internet Engineering Task Force, Nov. 1987, updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343. [Online]. Available: http://www.ietf.org/rfc/rfc1035.txt

[7] S. Bellovin, "Using the domain name system for system break-ins," in *The 5th Usenix UNIX Security Symposium*. Usenix, June 1995.

[8] S. Friedl, "An illustrated guide to the kaminsky dns vulnerability," Aug 2008. [Online]. Available: \url{http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html}

[9] D. Eastlake 3rd and C. Kaufman, "Domain Name System Security Extensions," RFC 2065 (Proposed Standard), Internet Engineering Task Force, Jan. 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2065.txt

[10] D. Eastlake 3rd, "Domain Name System Security Extensions," RFC 2535 (Proposed Standard), Internet Engineering Task Force, Mar. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2535.txt

[11] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS Security Introduction and Requirements," RFC 4033 (Proposed Standard), Internet Engineering Task Force, Mar. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4033.txt

[12] ——, "Resource Records for the DNS Security Extensions," RFC 4034 (Proposed Standard), Internet Engineering Task Force, Mar. 2005, updated by RFC 4470. [Online]. Available: http://www.ietf.org/rfc/rfc4034.txt

[13] ——, "Protocol Modifications for the DNS Security Extensions," RFC 4035 (Proposed Standard), Internet Engineering Task Force, Mar. 2005, updated by RFC 4470. [Online]. Available: http://www.ietf.org/rfc/rfc4035.txt

[14] B. Laurie, G. Sisson, R. Arends, and D. Blacka, "DNS Security (DNSSEC) Hashed Authenticated Denial of Existence," RFC 5155 (Proposed Standard), Internet Engineering Task Force, Mar. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5155.txt

[15] D. Moore, C. Shannon, D. Brown, G. Voelker, and S. Savage, "Inferring Internet denial-of-service activity," *ACM Transactions on Computer Systems (TOCS)*, vol. 24, no. 2, p. 139, 2006.

[16] N. Brownlee, K. Claffy, and E. Nemeth, "DNS measurements at a root server," *GLOBECOM-NEW YORK-*, vol. 3, pp. 1672–1676, 2001.

[17] R. Vaughn and G. Evron, "DNS amplification attacks," 2006. [Online]. Available: http://www.isotf.org/news/DNS-Amplification-Attacks.pdf

[18] V. Paxson, "An analysis of using reflectors for distributed denial-of-service attacks," *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 3, pp. 38–47, 2001.

[19] F. Baker and P. Savola, "Ingress Filtering for Multihomed Networks," RFC 3704 (Best Current Practice), Internet Engineering Task Force, Mar. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3704.txt

[20] C. Herley, "So long, and no thanks for the externalities: The rational rejection of security advice by users," in *Proceedings of the New Security Paradigms Workshop (NSPW)*, 2009.

[21] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, , and L. Cranor, "Crying wolf: An empirical study of ssl warning effectiveness," in *Proceedings of the 18th USENIX Security Symposium*, 2009.

[22] R. Wagner, "Address resolution protocol spoofing and man-in-the-middle attacks," *The SANS Institute*, 2001.

[23] A. Kapela and A. Pilosov, "Stealing the internet - a routed, wide-area, man-in-the-middle attack," in *Defcon 16, Las Vegas*, August 2008.

[24] R. Kuhn, K. Sriram, and D. Montgomery, "Border gateway protocol security," July 2007.

[25] R. Dhamija, J. Tygar, and M. Hearst, "Why phishing works," in *Proceedings of the SIGCHI conference on Human Factors in computing systems*. ACM, 2006, p. 590.

[26] D. J. Bernstein, "DNSCurve: Usable security for DNS." [Online]. Available: http://dsncurve.org

[27] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246 (Proposed Standard), Internet Engineering Task Force, Aug. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5246.txt

[28] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," RFC 4251 (Proposed Standard), Internet Engineering Task Force, Jan. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4251.txt

[29] E. Felten, "Mozilla debates whether to trust chinese CA," February 2010. [Online]. Available: http://www.freedom-to-tinker.com/blog/felten/mozilla-debates-whether-trust-chinese-ca

[30] J. Corbet, "China internet network information center accepted as a mozilla root CA," *Linux Weekly News*, February 2 2010. [Online]. Available: http://lwn.net/Articles/372264/

[31] P. Vixie, "What DNS is not," *Communications of the ACM*, vol. 50, no. 12, pp. 43–47, December 2009.

[32] M. S. Mimoso, "Kaminsky interview: DNSSEC addresses cross organizational trust and security," June 2009. [Online]. Available: http://searchsecurity.techtarget.com/news/interview/0,289202,sid14_gci1360143,00.html

[33] T. Maly, "I have some opinions about the RWW facebook login hilarity," February 2010. [Online]. Available: http://quietbabylon.posterous.com/i-have-some-opinions-about-the-rww-facebook-l

[34] J. Sobey, R. Biddle, P. C. van Oorschot, and A. S. Patrick, "Exploring user reactions to new browser cues for extended validation certificates," in *Proceedings of the 13th European Symposium on Research in Computer Security*. Springer, 2008, pp. 411–427.