Measuring the practical impact of DNSSEC Deployment

Wilson Lian *UC San Diego*
Eric Rescorla *RTFM, Inc.*
Hovav Shacham *UC San Diego*
Stefan Savage *UC San Diego*

Abstract

DNSSEC extends DNS with a public-key infrastructure, providing compatible clients with cryptographic assurance for DNS records they obtain, even in the presence of an active network attacker. As with many Internet protocol deployments, administrators deciding whether to deploy DNSSEC for their DNS zones must perform cost/benefit analysis. For some fraction of clients — those that perform DNSSEC validation — the zone will be protected from malicious hijacking. But another fraction of clients — those whose DNS resolvers are buggy and incompatible with DNSSEC — will no longer be able to connect to the zone. Deploying DNSSEC requires making a cost-benefit decision, balancing security for some users with denial of service for others.

We have performed a large-scale measurement of the effects of DNSSEC on client name resolution using an ad network to collect results from over 500,000 geographically-distributed clients. Our findings corroborate those of previous researchers in showing that a relatively small fraction of users are protected by DNSSEC-validating resolvers. And we show, for the first time, that enabling DNSSEC measurably increases end-to-end resolution failures. For every 10 clients that are protected from DNS tampering when a domain deploys DNSSEC, approximately one ordinary client (primarily in Asia) becomes unable to access the domain.

1 Introduction

The *Domain Name System* (DNS) [32], used to map names to IP addresses, is notoriously insecure; any active attacker can inject fake responses to DNS queries, thus corrupting the name → address mapping. In order to prevent attacks on DNS integrity, the *Internet Engineering Task Force* (IETF) has developed DNSSEC [4], a set of DNS extensions which allows DNS records to be digitally signed, thus preventing—or at least detecting—tampering.

Over the past several years, public enthusiasm for DNSSEC has increased significantly. In July 2010, the DNSSEC root zone (containing all top level domains) was signed; in March 2011, `.com`, the largest top level domain, was signed; in January 2012, Comcast announced that they had switched all of their DNS resolvers to do DNSSEC validation and that they had DNSSEC-signed all customer domains they were serving [30]. Moreover, protocol designs which *depend* on DNSSEC have started to emerge. For instance, DANE [20] is a DNS extension that uses DNS to authenticate the name → public key binding for SSL/TLS connections. Obviously, DANE is not secure in the absence of DNSSEC, since an attacker who can man-in-the-middle the SSL/TLS connection can also forge DNS responses.

Despite the effort being poured into DNSSEC, actual deployment of signed records at the end-system level has remained quite limited. As of February 2013, VeriSign Labs' Scoreboard[1] measured 158,676 (.15%) of `.com` domains as secured with DNSSEC. As with many Internet protocol deployments, there is a classic collective action problem: because the vast majority of browser clients do not verify DNSSEC records or use resolvers which do, the value to a server administrator of deploying a DNSSEC-signed zone is limited. Similarly, because zones are unsigned, client applications and resolvers have very little incentive to perform DNSSEC validation.

A zone administrator deciding whether to deploy DNSSEC must weigh the costs and benefits of:

- The fraction of clients whose resolvers validate DNSSEC records and therefore would be able to detect tampering if it were occurring and DNSSEC were deployed.
- The fraction of clients which fail with valid DNSSEC records and therefore will be unable to reach the server whether or not tampering is occurring.

In this paper, we measure these values by means of a large-scale study using Web browser clients recruited via an advertising network. This technique allows us to sample a cross-section of browsers behind a variety of network configurations without having to deploy our own sensors. Overall, we surveyed 529,294 unique clients over a period of one week. Because of the scale of our study and the relatively small error rates we were attempting to quantify, we encountered several pitfalls that can arise in ad-recruited

[1]Online: `http://scoreboard.verisignlabs.com/`. Visited 20 February 2013.

browser measurement studies. Our experience may be relevant to others who wish to use browsers for measurements, and we describe some of these results in Section 4.2.

Ethics. Our experiment runs automatically without user interaction and is intended to measure the behavior and properties of hosts along the paths from users to our servers rather than the users themselves. We worked with the director of UC San Diego's Human Research Protections Program, who certified our study as exempt from IRB review.

2 Overview of DNS and DNSSEC

A DNS name is a dot-separated concatenation of labels; for example, the name `cs.ucsd.edu` is comprised of the labels `cs`, `ucsd`, and `edu`. The DNS namespace is organized as a tree whose nodes are the labels and whose root node is the empty string label. The name corresponding to a given node in the tree is the concatenation of the labels on the path from the node to the root, separated by periods.

Associated with each node are zero or more *resource records* (RRs) specifying information of different types about that node. For example, IP addresses can be stored with type A or AAAA RRs, and the name of the node's authoritative name servers can be stored in type NS RRs. The set of all RRs of a certain type[2] for a given name is referred to as a *resource record set* (RRset).

2.1 Delegation

DNS is a distributed system, eliminating the need for a central entity to maintain an authoritative database of all names. The DNS namespace tree is broken up into **zones**, each of which is owned by a particular entity. Authority over a subtree in the domain namespace can be delegated by the owner of that subtree's parent. These delegations form zone boundaries. For example, a name registrar might delegate ownership of `example.com` to a customer, forming a zone boundary between `.com` and `example.com` while making that customer the authoritative source for RRsets associated with `example.com` and its subdomains. The customer can further delegate subdomains of `example.com` to another entity. Figure 1 depicts an example DNS tree.

2.2 Address resolution

The most important DNS functionality is the resolution of domain names to IP addresses (retrieving type A or AAAA RRsets). Domain name resolution is performed in a distributed, recursive fashion starting from the root zone, as shown in Figure 2. Typically, end hosts do not perform resolution themselves but instead create DNS queries and send them to *recursive resolvers*, which carry out the resolution to completion on their behalfs. When a nonrecursive DNS server receives a query that it cannot answer, it returns the name and IP address of an authoritative name server as far down as possible along the path to the target domain name. The recursive resolver then proceeds to ask that server. In this fashion, the query eventually reaches a server that can answer the query, and the resolution is complete. This recursive process is bootstrapped by hardcoding the names and IP addresses of root nameservers into end hosts and recursive resolvers.

[2]And class, but for our purposes class is always `IN`, for "Internet."

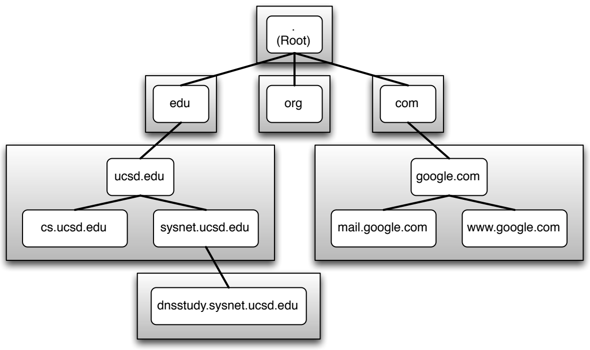

Figure 1: Example DNS name tree. Shaded boxes represent zone boundaries. Edges that cross zone boundaries are delegations.

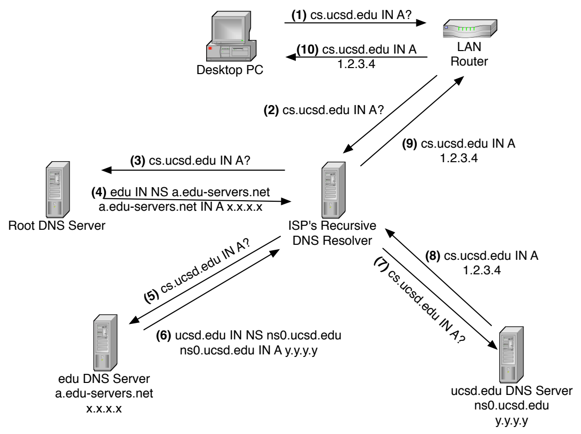
(1) cs.ucsd.edu IN A?
(10) cs.ucsd.edu IN A 1.2.3.4
Desktop PC
LAN Router
(2) cs.ucsd.edu IN A?
(9) cs.ucsd.edu IN A 1.2.3.4
(3) cs.ucsd.edu IN A?
(4) edu IN NS a.edu-servers.net a.edu-servers.net IN A x.x.x.x
Root DNS Server
ISP's Recursive DNS Resolver
(5) cs.ucsd.edu IN A?
(6) ucsd.edu IN NS ns0.ucsd.edu ns0.ucsd.edu IN A y.y.y.y
(7) cs.ucsd.edu IN A?
(8) cs.ucsd.edu IN A 1.2.3.4
edu DNS Server a.edu-servers.net x.x.x.x
ucsd.edu DNS Server ns0.ucsd.edu y.y.y.y

Figure 2: Simplified DNS address resolution procedure for `cs.example.tld`. In this example, there are at most one nameserver and one IP address per name.

2.3 DNS (in)security

The original DNS design did not provide any mechanisms to protect the integrity of DNS response messages. Thus, an active network attacker can launch a woman-in-the-middle attack to inject her own responses which would be accepted as if they were legitimate. This attack is known as *DNS spoofing.* Moreover, because recursive resolvers typically cache responses, a single spoofed response can be used to perform a *DNS cache poisoning* attack, which results in future responses to requests for the same RRset returning the bogus spoofed response. The mechanisms by which DNS cache poisoning is carried out are outside the scope of this work but have been studied more formally in [38]. DNS spoofing and cache poisoning may be used to compromise any type of DNS RR.

2.4 DNSSEC to the rescue

The *Domain Name System Security Extensions* (DNSSEC) [4], aim to protect against DNS spoofing attacks by allowing authoritative nameservers to use public key cryptography to digitally sign RRsets. Security-aware recipients of a signed RRset are able to verify that the RRset was signed by the holder of a particular private key, and a chain of trust from the root zone downwards ensures that a trusted key is used to validate signatures.

While DNSSEC adds a number of new RR types, the *DNSKEY*, *RRSIG*, *DS* only the records are relevant for our purposes; we describe them briefly here.

DNSKEY: DNSKEY records are used to hold public keys. Each zone authority generates at least one public/private key pair, using the private keys to sign RRsets and publishing the public keys in Domain Name System Key (DNSKEY) resource records.

RRSIG: When a zone is signed, a resource record signature (RRSIG) resource record is generated for each RRset-public key pair. In addition to containing a cryptographic signature and the name and type of the RRset being signed, the RRSIG RR specifies a validity window and the name of the signing key's owner.

DS: Lastly, the Delegation Signer (DS) RR type links signed zones to establish the chain of trust. Each DS RR contains the digest of one of the subzone's DNSKEY RRs.

DNSSEC's security is built on the chain of trust model. Starting from a "trust anchor," a validator attempts to trace a chain of endorsements from the root all the way to the RRset being validated; I.e., that each DNSKEY or DS record along the path and the final RRSet is correctly signed by the parent's public key. If a chain of trust can be constructed all the way to the trust anchor, then the validating resolver can have confidence that the information in that RR is correct — or at least that it is cryptographically authenticated.

Because DNSSEC is a retrofit onto the existing insecure DNS, it is explicitly designed for incremental deployment, and insecure (i.e., unsigned) domains can coexist with secure domains. Thus, DNSSEC-capable resolvers should be able to resolve unsigned domains, and non-DNSSEC resolvers should be able to resolve DNSSEC-signed domains, though of course they will not gain any security value. In order to make this work, DNSSEC records are designed to be backwards-compatible with existing resolvers, and DNSSEC resolvers are able to distinguish zones which simply are not signed from those which are signed but from which an attacker has stripped the signatures (the DS record is used for this purpose).

Unfortunately, while DNSSEC is *designed* to be backwards compatible, it is known [9] that there are some network elements which do not process DNSSEC records properly. The purpose of this work is to determine the frequency of such elements and in particular their relative frequency to elements which actually validate DNSSEC signatures and thus benefit from its deployment.

3 Methodology

In order to address this question, we conducted a large-scale measurement study of web browsers in the wild. In particular, we sought to measure two quantities:

- What fraction of clients validate DNSSEC records and therefore would be able to detect tampering if it were occurring and DNSSEC were deployed?
- What fraction of clients fail with valid DNSSEC records and therefore will be unable to reach the server whether or not tampering is occurring?

Answering these questions requires taking measurements from a large number of clients. We gathered our clients by purchasing ad space from an online advertising network; the ad network enabled us to host an ad at a fixed URL which would be loaded in an iframe on various publishers' web sites. Our ad included JavaScript code to drive the experiment and was executed without any user interaction upon the loading of the ad iframe in clients' browsers. In order to minimize sampling bias, our ad campaign did not target any particular keywords or countries.

However, because our measurements were sensitive to the reliability of the participants' Internet connections, we configured our ad campaign to target desktop operating systems, to the exclusion of mobile users.

Our client-side "driver script" (discussed in detail in § 3.1) induces participants' browsers to load 1×1-pixel images ("test resources") from various domains. This is a standard technique for inducing the browser to load resources from different origins than the containing document. These domains fall into the following three classes:

- *nosec* — without DNSSEC
- *goodsec* — with correctly-configured DNSSEC
- *badsec* — with DNSSEC against which we simulate misconfiguration or tampering by an active network attacker

The *goodsec* and *badsec* zones were signed with 1024-bit keys[3] using RSA-SHA1.

If we observe an HTTP request for a test resource, we conclude that the participant's browser was able to resolve that type of domain. Otherwise, we conclude that it was not.

These three domain classes allow us to assess the client/resolver's DNSSEC behavior. The *nosec* domain class serves as a control, representing the state of the majority of the sites on the web. Failed loads from the *goodsec* domain class allow us to measure the fraction of clients which would not be able to reach a DNSSEC-enabled site, even in the absence of an attack. Failed loads from the *badsec* domain class tell us about the fraction of clients which detect and react to DNSSEC tampering.

During each ad impression, the driver script attempts to resolve and load a total of 27 test resources. They are distributed as follows: one *nosec* domain, one *goodsec* domain, and 25 different *badsec* domains. Each *badsec* variant simulates an attack against DNSSEC at a different point in the chain of trust, and as we will see in Section 4, certain validating resolvers exhibit bugs that cause some *badsec* domains to be treated as correctly-signed.

3.1 Client-side experiment setup

Figure 3 shows how our driver script is embedded in an ad in a publisher's web page. We provide the ad network with an ad located at a static URL which is wrapped in an iframe by the ad network. The publisher places an iframe in its web page whose source points to the iframe wrapping the ad. Our ad page residing at the static URL iframes the **measurement page**, which contains the JavaScript driver program. Each instance of the measurement page and all requests generated by it are linked by a version 4 UUID [29] placed in both the URL query string and the domain name (with the exception of the measurement page, which only has it in the query string).

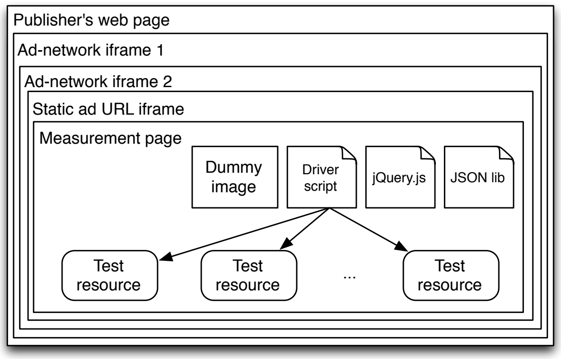

Figure 3: Client-side experiment setup

The measurement page loads a dummy ad image and 3 pieces of JavaScript which are the following:

- A minified jQuery[4] [26] library hosted by `jquery.com`
- A JSON encoding and decoding library hosted on our servers
- The experiment's JavaScript "driver script"

The measurement page body's `onLoad` handler commences the experiment by invoking the driver script. The driver script randomizes the order of a list of *nosec*, *goodsec*, and *badsec* domains then iterates over that list, creating for each domain an image tag whose source property points to an image hosted on that domain. The creation of the image tag causes the participant's browser to attempt to resolve the domain name and load an image from it. Because we need to gather data for all domains in the list before the participant navigates away from the web page containing the ad, the driver script does not wait for each image to complete its load attempt before proceeding to the next domain. Instead, it creates all of the image tags in rapid succession. The driver script also registers `onLoad` and `onError` callbacks on each image tag created to monitor whether each load succeeds or fails. When a callback for an image fires, the outcome of the load, along with

[3]We attempted to use 2048-bit keys, but at the time of the experiment, our domain registrar, GoDaddy, did not support keys that large.

[4]We used jQuery to minimize browser compatibility issues.

info about the browser, are sent via jQuery's AJAX `POST` mechanism to a PHP logging script on our servers. Once the driver script detects that all image tags have invoked either an `onLoad` or `onError` callback, it creates a final image tag whose source domain is a unique *nosec* domain (`UUID.complete.dnsstudy.ucsd.edu`). A DNS query for such a domain serves as a "completion signal" and allows us to identify UUIDs where the user did not navigate away from the ad publisher's page before completing the trial. We discarded the data from any page load which did not generate the completion signal.

3.2 Identifying successful DNS resolution

Our original intent was to use `onLoad` and `onError` handlers attached to each test resource's image tag to measure the outcome of the HTTP requests for test resources. If the `onLoad` handler was called, we would record a successful HTTP request; if instead the `onError` handler was called, we would record a failed HTTP request. These results are reported back to our servers via AJAX `POST`. However, we found 9754 instances of the `onError` handler firing, the test resource subsequently being loaded, and then the `onLoad` handler firing. For another 1058 test resource loads, the `onLoad` handler fired, despite our receiving neither the corresponding DNS lookups nor the HTTP requests for the test resources in question. Consequently, we looked to different avenues for identifying resolution success.

Because we are not able to ascertain the result of a DNS lookup attempt via direct inspection of the DNS caches of our participants and their recursive resolvers, we must infer it from the presence of an HTTP request whose `Host` header or request line specifies a particular test resource's domain name as an indicator of DNS resolution success. Thus, if we observed a completion signal for a particular UUID but did not observe an HTTP request associated with that UUID for a certain test resource type, we infer that the DNS resolution for that UUID-test resource pair failed. Note however that we can record a completion signal after observing just a DNS query for it: what matters is whether the driver script attempted to load the completion signal resource, not whether it succeeded in doing so.

This strategy has the potential to over-estimate the number of DNS resolution failures due to TCP connections that are attempted and are dropped or aborted before the HTTP request is received by our servers. The only source of this type of error that we are able to control is our HTTP servers' ability to accept the offered TCP-connection load at all times throughout the experiment. We describe our serving infrastructure in Section 3.4. We believe it is sufficiently robust against introducing this type of error.

3.3 Cache control

Because requests fulfilled by cache hits do not generate HTTP and DNS logs that we can analyze, we took measures, described in Table 1, to discourage caching. Most importantly, the use of a fresh, random UUID for each ad impression serves as a cache-buster, preventing cache hits in both DNS resolvers and browsers.

If, despite our efforts, our static ad page is cached, causing the measurement page to be requested with a cached UUID, we must detect it and give the current participant a fresh UUID. To this end, we used a memcached cache as a UUID dictionary to detect when the measurement page was loaded with a stale UUID. If this occured, the stale measurement page was redirected to one with a fresh UUID.

3.4 Serving infrastructure

To run our study, which generates large bursts of traffic, we rented 5 `m1.large` instances running Ubuntu 10.04 on Amazon's Elastic Compute Cloud (EC2). All 5 instances hosted identical BIND 9 (DNS), nginx (HTTP), and beanstalkd (work queue) servers. The nginx servers supported PHP 5 CGI scripts via FastCGI. Tables 2 and 3 show the adjustments made to the servers' configuration parameters to ensure a low rate of dropped connections.

One instance ran a MySQL server, another ran a memcached server. To increase our EC2 instances' ability to accept large quantities of short TCP connections, we configured our machines to timeout connections in the FIN-WAIT-2 state after only a fraction of the default time and to quickly recycle connections in the TIME-WAIT state. This was accomplished by setting the sysctl variables *tcp_fin_timeout* and *tcp_tw_recycle* to 3 and 1, respectively.

3.4.1 DNS & BIND 9

All 5 EC2 instances ran identical BIND 9 DNS servers providing authoritative DNS resolution for all *nosec*, *goodsec*, and *badsec* domains. We used Round Robin DNS to distribute load across all 5 DNS and web servers. In order to reduce the chance of load failures due to large reply packets, our DNS servers were configured (using BIND's `minimal-responses` option) to refrain from sending unsolicited RRs that are not mandated by the DNS specification. Specifically, we only send the extra DNSSEC RRs in response to queries which include the DNSSEC OK option (approximately two thirds of all queries).

Type	Value	Used on
HTTP header	`Cache-Control: no-cache, must-revalidate`	static ad page, measurement page, driver script
HTTP header	`Expires: Sat, 26 Jul 1997 00:00:00 GMT`	static ad page, measurement page, driver script
HTML `<meta>`	`http-equiv="Pragma" content="no-cache"`	static ad page, measurement page
HTML `<meta>`	`http-equiv="Expires" content="−1"`	static ad page, measurement page

Table 1: Description of the HTTP and HTML anti-caching measures and their uses.

worker_processes	8
worker_rlimit_nofile	65,535
worker_connections	65,000

Table 2: Non-default nginx server config params.

PHP_FCGI_CHILDREN	50
PHP_FCGI_MAX_REQUESTS	65,000

Table 3: Non-default PHP FastCGI config params.

Our 5 BIND servers are authoritative for all domain names used in our study except for the domain of the static ad URL that iframes the measurement page. Because we were not interested in measuring resolution of those domains we hosted their authoritative servers on Amazon Route 53 DNS service.

3.5 Data gathering

Our analysis of the behavior of participants' browsers and resolvers is based on the following 3 data sources: nginx access logs, BIND request logs, and MySQL tables containing the outcomes and browser info reported by AJAX `POST` messages.

Nginx was configured to use its default "common log format" [34], which includes a timestamp of each request, the URL requested, the user agent string, among other details about the request and its corresponding response. However, BIND's log format is hardcoded and compiled into the binary. Its default logging behavior only provides basic information about queries (e.g., a timestamp, the domain name, the source IP and port). It does not provide information about replies and excludes certain important diagnostic fields. We modified and recompiled BIND to add enhanced logging of requests and replies. Log lines for requests were modified to include the request's transaction ID and the value of the sender's UDP payload size from the EDNS0 OPT RR (if present) [39]. We added support for reply logs that include the transport-layer protocol in use, the size of the reply, and the transaction ID. With these additional log details, we are able to link requests to replies and determine if a lookup fell back to TCP due to truncation of the UDP reply. BIND logs are also used to identify the UUIDs for which a completion signal was sent as well as to determine which resolvers were associated with a particular UUID.

The client-side driver script AJAX `POST`s the outcome of each test resource load along with additional metadata regarding the experiment and the state of the browser environment under which it is running. These data are logged by our servers.

3.6 Experiment scheduling

In our preliminary test runs of the study, we found that the successful load rates for test resources varied depending on the time of day at which the experiment was conducted. To account for this variability, we conducted an extended study lasting for a full week. Every two hours, we paid ad network enough for 10,000 impressions to be shown.

4 Results

In this section we describe the results of our measurements. We begin by providing an overview of our data. Then, in Section 4.1 we describe our measurements of the differential impact of DNSSEC on resolution success. Finally, in Section 4.2, we describe a number of confounding network artifacts that plague any attempt to use advertisement surveys to measure small signals against the background of a noisy Web environment.

Over the course of the 84 segments of our week-long experiment, we collected data from 529,294 ad impressions, receiving DNS queries from 35,010 unique DNS resolvers. Figure 4 shows the distribution of unique resolvers performing resolution for each UUID. The distribution has a long tail, although 98% of UUIDs used at most 25 resolvers. We mapped each resolver's IP address to its ASN and found that 92.75% of the clients surveyed were observed using recursive resolvers whose IP addresses resided in the same ASN, and 99.12% used resolvers

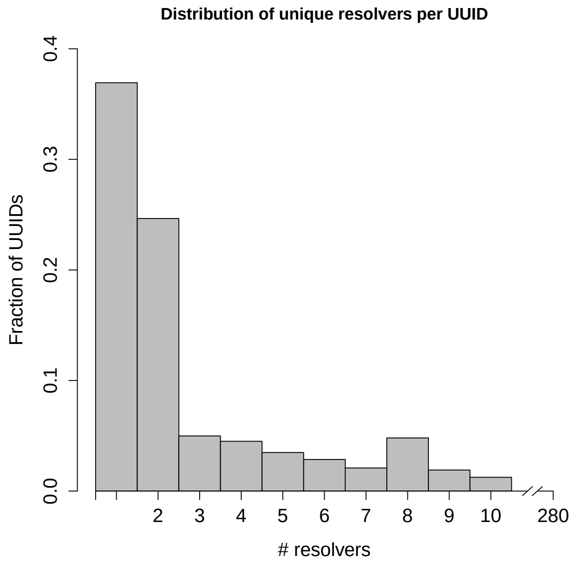

Figure 4: Distribution of the number of unique resolvers observed performing DNS resolution per UUID. Tail not shown.

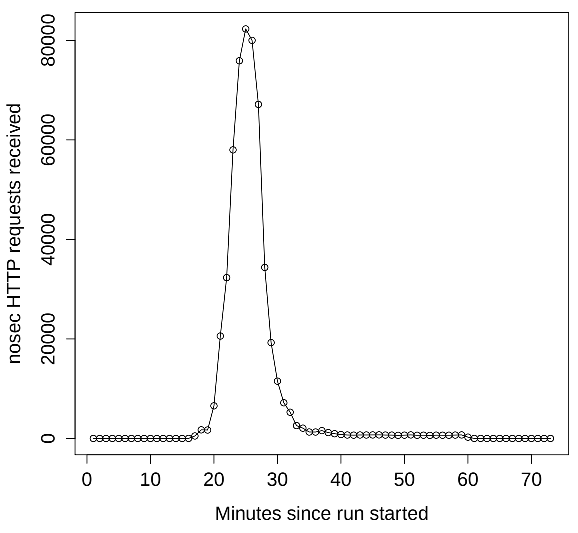

Figure 5: Plot showing total number of requests received during each minute after the start of a run, aggregated over all runs.

Class	Failure rate	CI 0.99
nosec	0.7846%	0.7539% - 0.8166%
goodsec	1.006%	0.9716% - 1.042%
badsec	2.661%	2.649% - 2.672%

Table 4: Failure rates for each class of test resource.

in two or fewer ASNs. This is consistent with our expectation that most users use their default DNS resolvers provided by their ISPs, while a small percentage of "power users" might configure their systems to take advantage of open resolvers such as Google Public DNS.[5]

As shown in Figure 5 each ad buy results in a delay of approximately 20 minutes from the time we released funds to the ad network, at which point impressions start to appear. Incoming traffic spikes for 15 minutes, peaking around 25 minutes into the run and tapering off for the remainder of the first hour.

We also witnessed considerable drop-off at each stage of executing experiment code in the participants' browsers. Figure 6 illustrates the number of UUIDs observed reaching each stage of the experiment. 15.88% of the ad impressions that we paid for did not even manage to load the driver script and only 63.02% of the impressions we paid for actually resulted in a completed experiment. This compares favorably with past studies. For instance, prior work by Huang et al. [22], which also used ad networks to recruit participants to run experiment code, had only a 10.97% total completion rate.

4.1 DNSSEC Resolution Rates

The first question we are interested in answering is the impact on load failure rates of introducing DNSSEC for a given domain. Table 4 shows the raw failure rates across each class of test resource, where the failure rate is defined as one minus the quotient of the number of successful test resource loads and the number of attempted resource loads across all UUIDs for which we received a completion signal. This table is sufficient to draw some initial conclusions. First, as evidenced by the low failure rate of *badsec* domains the vast majority of end hosts and their recursive resolvers do not perform DNSSEC validation. If all end hosts or recursive resolvers verified DNSSEC, we would expect a *badsec* failure rate of 100%, instead of the observed value of 2.661%. Thus, the increased security value of DNSSEC-signing a domain is relatively low, as most resolvers will not detect tampering against DNSSEC-signed domains.

Second, DNSSEC-signed domains—even validly signed domains—have a higher failure rate than non-DNSSEC-signed domains: just DNSSEC-signing a domain increases the failure rate from around

[5]`https://developers.google.com/speed/public-dns/`

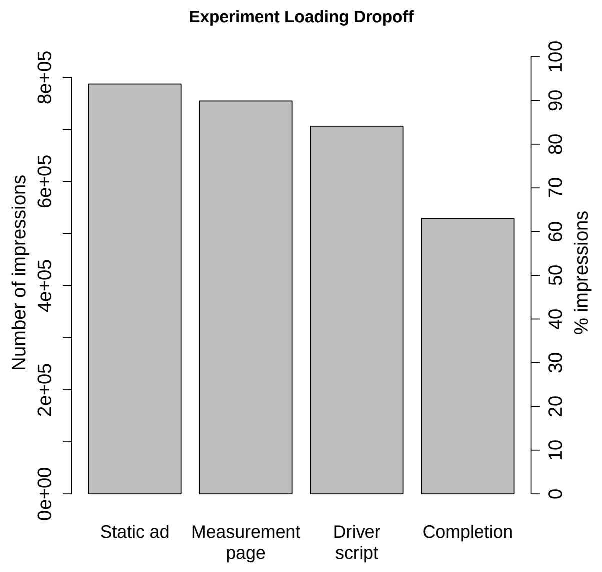

Figure 6: Plot of UUIDs that reached each stage of the experiment.

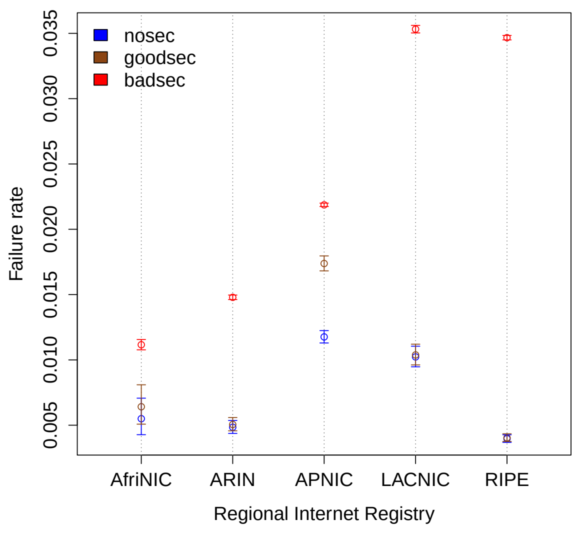

Figure 7: Failure rates broken down by resolver IP RIR. Error bars indicate a 95 percent binomial proportion confidence interval.

0.7846% to 1.006% (though this value is very sensitive to geographic factors, as discussed in the following section). While this is not a huge difference, it must be compared to the detection rate of bad domains, which is also very small. Moreover, because resolvers which cannot process DNSSEC at all appear to "detect" bogus DNSSEC records, the *badsec* failure rate in Table 4 is actually an overestimate of clients behind DNSSEC-validating resolvers, which is probably closer to 1.655% (the difference between the *badsec* and *goodsec* rates).

4.1.1 Geographic Effects

As mentioned above, the raw numbers are somewhat misleading because the failure rates are very geographically dependent. In order to explore this dependence we categorized each test case (UUID) by geographic area based on the resolver IP observed performing resolution for a domain containing the UUID.[6] We used the CAIDA prefix-to-AS mapping dataset [11] to determine the Autonomous System Number (ASN) for each for client's resolver IP address and then assigned each client to the Regional Internet Registry (RIR) which is responsible for that AS, as listed in Table 5.

[6]If there was more than one resolver associated with a particular UUID, our analytics package chose one arbitrarily during the process of merging the records. If we restrict our analysis to clients which only use one resolver, the overall error rate goes down, but our results are qualitatively similar, with the error rates being 0.0046, 0.0055, and 0.0119, for nosec, goodsec, and badsec, respectively.

As shown in Figure 7, resolution failure rates vary widely by region, as does the difference in resolution rates between *nosec*, *goodsec*, and *badsec*. In particular, while all five regions show a significant difference (2-proportion z-test, $p < 0.0001$) between aggregate *badsec*-domain outcomes and *nosec* & *goodsec* outcomes, only APNIC (Asia Pacific) shows a significant difference between *nosec* and *goodsec* (McNemar's test, $p < 0.0001$). While AfriNIC (Africa) shows a qualitative difference, we do not have enough data points to determine whether it is statistically significant. Note that in general APNIC seems to have an elevated resolution failure rate; LACNIC (Latin America) does as well but still does not show a significant difference between *nosec* and *goodsec*. We drilled down into the resolvers responsible for anomalous failure rates and present our findings in Sections 4.1.3, & 4.1.4.

4.1.2 The Impact of Packet Size and TCP Fallback

One commonly-expressed concern with DNSSEC is that it increases the size of DNS responses and, consequently, failure rates. Ordinarily, DNS requests and responses are carried over UDP, which limits the maximum size of the responses. DNS has two mechanisms to allow responses larger than the 512-byte limit defined in RFC 1035 [33]:

- Resolution can fall back to TCP if the server supports it.

Name	Abbreviation	Frequency	Percentage
African Network Information Centre	AfriNIC	10,914	2.062%
American Registry for Internet Numbers	ARIN	75,577	14.28%
Asia-Pacific Network Information Centre	APNIC	200,366	37.86%
Latin America and Caribbean Network Information Centre	LACNIC	62,925	11.89%
Réseaux IP Européens Network Coordination Centre	RIPE NCC	179,492	33.91%
Unclassifiable		20	$< 0.001\%$

Table 5: Table listing the 5 Regional Internet Registries (RIRs). The Frequency and Percentage columns indicate the number and relative prevalence of UUIDs for which at least one DNS query originated from each region.

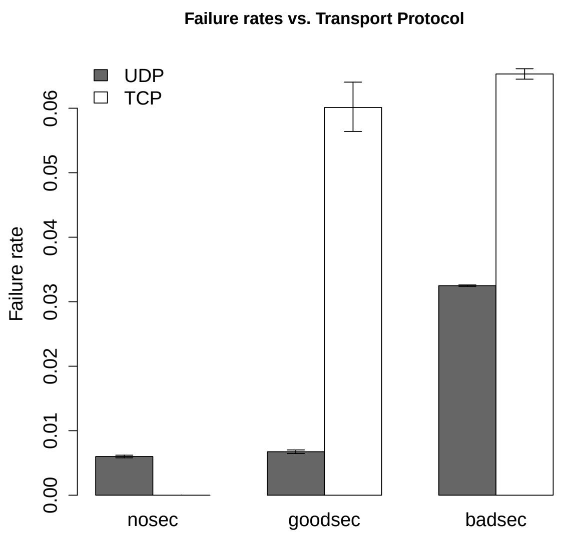

Figure 8: Failure rates broken down by DNS transport protocol. Error bars indicate a 95 percent binomial proportion confidence interval.

- Clients can advertise a larger maximum UDP datagram size via the EDNS0 OPT pseudo-RR [39].

Unfortunately, both of these mechanisms can cause problems for some intermediaries [7, 8, 10]. Because the resolver behavior is observable on the server, we can directly measure the impact of these strategies on test resource load failures.

In order to look more closely at these effects, we first filtered out the data for the 4,739,669 (33.25%) lookup requests we received which did not have the DNSSEC OK flag set. The DNSSEC OK flag announces the query source's willingness to receive DNSSEC RRs, and thus when it is not set, our resolver simply sends the requested records without the DNSSEC RRs.[7] Non-DNSSEC OK lookups for DNSSEC resources appear to have similar success rates to *nosec* resources. Out of the remaining 9,516,394 (66.75%) transactions where DNSSEC OK was indicated, 4.22% of *goodsec* and 4.064% of *badsec* lookups fell back to TCP. These TCP lookups had dramatically higher failure rates: 6.011% for *goodsec* and 6.531% for *badsec* compared to 0.6742% for *goodsec* and 3.249% for *badsec* when UDP was used. For *nosec*, resolution never fell back to TCP, and the failure rate of 0.6% [8]. was similar to that for *goodsec* with UDP. Figure 8 summarizes these findings.

The similar UDP failure rates for *nosec* and *goodsec* suggest that it is the TCP fallback that results from DNSSEC's increased response sizes, and not the bigger responses themselves, that is the major contributor to the elevated *goodsec* failure rate.

TCP fallback in the DNS resolution for one component of a web page can have a negative impact on the load rate of other components on the page, even if their DNS lookups do not themselves fall back to TCP. If we partition the UUIDs into those that fall back to TCP for at least one test resource and those that never fall back to TCP, we find that the *nosec* failure rates are 1.0791% for the former and 0.7617% for the latter. We have not explored these effects in detail, but it seems likely that the failed resolution slows down the retrieval of the rest of the resources, thus causing failures.

We also found that accurate path MTU prediction is crucial for maintaining high resolution success. For 13,623 test resources (0.0953% of the 14,291,174 total), we observed that recursive resolvers overestimated the UDP path MTU, advertised an inflated

[7]If multiple queries were present we considered the DNSSEC OK flag to be set if any of the queries had it. 2.771% of test resources exhibited variation in this flag.

[8]This *nosec* failure rate is lower than the one found in Table 4 because, to be consistent with the *goodsec* and *badsec* failure rate calculations in this section, it excludes failed test resource loads for which we did not observe a DNS lookup attempt.

value via EDNS0, and subsequently had to retry the lookup with a smaller advertised value. Test resources whose lookups included this path MTU discovery behavior failed to load 14.09% of the time compared to 2.519% for those that did not.

4.1.3 Case Study: *badsec-b8* validation anomaly

We compared the failure rates of the *badsec* domains and observed that the *badsec-b8* variant exhibited a significantly lower failure rate (1.480%) than all other *badsec* types (McNemar's test applied pairwise against each other *badsec* variant, $p < 0.01$). In *badsec-b8*, we simulated an invalid DNSKEY RRSIG RR by incrementing the labels field of the RR data and signing it with a correctly-authenticated key. The labels field in an RRSIG RR is used for matching RRSIGs to the RRsets they authenticate when wildcards are involved. For example, if a zone declares the `*.foo.com` wildcard name, then RRSIGs for the RRsets of names matching `*.foo.com` (e.g., `www.foo.com`) would have a labels field value of 2. Section 5.3.1 of RFC 4035 [5] stipulates that an RRSIG RR must have a labels field value less than or equal to the number of labels in the owner name of the RRset that it authenticates.

To identify resolvers responsible for this validation anomaly, we first partitioned the set of UUIDs by the IP address of the resolver associated with that UUID. Using the partitioned dataset, we identified 124 resolvers whose failure rate for *badsec-b8* was significantly lower than that of each of the other *badsec* variants (McNemar's test, $p < 0.01$). Moreover, for 123 of these resolvers, the *badsec-b8* and *goodsec* failure rates did not significantly differ at the .01 level (McNemar's test).

With the cooperation of one of the ISPs whose resolvers exhibited the validation anomaly, we were granted access to query their closed resolvers and were able to manually reproduce the errant validation. We also added -1, +2, and +100 to the RRSIG labels field values and found that the resolvers incorrectly accepted all of the increased values, but not the decreased value, suggesting that the DNS server implementation in use reversed the inequality for testing the labels field.

We were unable to devise a cache-poisoning attack that leverages this validation error under any reasonable threat model.

4.1.4 Case Study: *badsec-c12* validation anomaly

The failure rate of the *badsec-c12* variant (2.521%) differed significantly from those of all other *badsec* domains and was the second lowest, after *badsec-b8* (McNemar's test, $p < 0.01$). The *badsec-c12* subdomain attacks the DNSSEC chain of trust by not providing the RRSIG RR for the test resource's type `A` RRset. A properly-validating server should not consider the affected `A` RRset validated unless it were able to retrieve and validate its RRSIG.

All 32 of the resolvers in our dataset that exhibited this validation anomaly belonged to the same /22 subnetwork controlled by one particular ISP, as did 45 of the 49 resolvers for which McNemar's test showed significantly-different ($p < 0.01$) failure rates between *nosec* and *goodsec*. Customers using this ISP's recursive resolvers suffer from the worst of both worlds. They are not only more vulnerable to a man-in-the-middle attack against DNSSEC, but also less likely to be able to access a domain with DNSSEC enabled than one without. We were unable to obtain access to the ISP's closed recursive resolvers to try to manually reproduce the incorrect validation behavior.

Due to the reduced size of responses omitting RRSIG RRs for type A queries, no *badsec-c12* DNS resolutions fell back to TCP, and for the 32 resolvers exhibiting the validation anomaly, *badsec-c12*'s failure rate (1.245%) was significantly less than that of *goodsec* (13.81%) (McNemar's test applied separately for each resolver, $p < 0.01$).

4.1.5 Case Study: Comcast

In January 2012, Comcast announced that it had finished deploying DNSSEC within its network and that its residential customers would thenceforth be protected by DNSSEC-validating DNS resolvers [30]. We identified dynamic Comcast IP end hosts in our dataset using a list of IP prefixes published by Comcast [13]. One should expect that Comcast end hosts in our dataset would fail on *goodsec* at a lower than average rate and *badsec* at a higher than average rate. Indeed, the 582 Comcast end hosts observed exhibited a 0.1718% failure rate for *goodsec* and a 92.5636% failure rate for *badsec*. For comparison, Comcast end hosts failed on *nosec* domains 0.1718% of the time. This result is consistent with the expectation that a network that is properly configured for DNSSEC will have identical behavior for *nosec* and *goodsec*. Our measurements indicate that the majority of the difference between Comcast's *badsec* failure rate and 100% is caused by users who are not using Comcast for recursive resolution and therefore do not benefit from Comcast's DNSSEC verification; if we exclude end-hosts that use resolvers outside of Comcast's AS the *badsec* failure rate improves to 98.6544%.

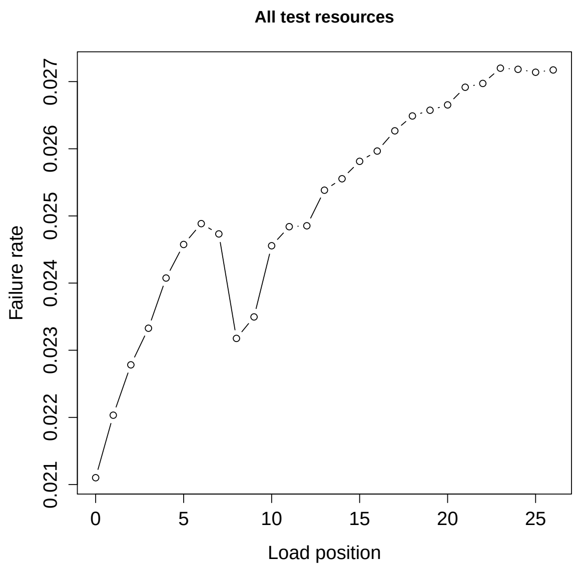

Figure 9: Plot of failure rate across all test resource types versus load order.

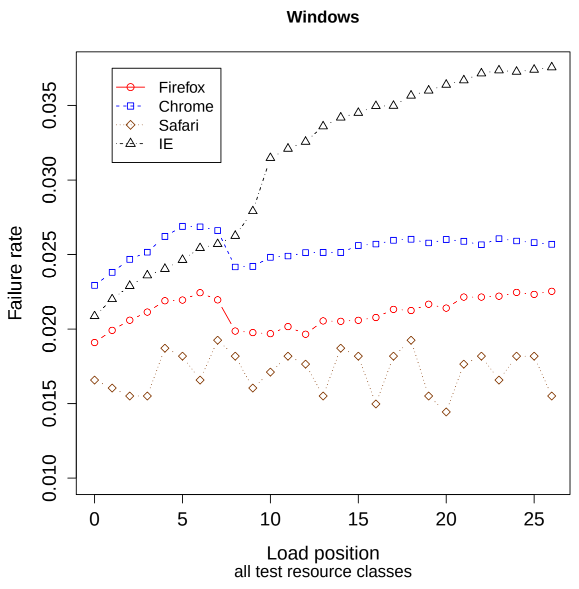

Figure 10: Failure rate versus load order across all test resources for Windows clients.

Percentile	Test duration (seconds)
50%	9
90%	31
95%	50
98%	100

Table 6: Percentiles from the distribution of test durations, measured from the time of the measurement page load to the time of the completion signal.

4.2 Measurement Difficulties

Because our primary measurement endpoint is the browser's failure to retrieve a resource, we are very sensitive to any other sources of failure other than the ones we are attempting to measure; by contrast, many previous studies such as [22] measured between multiple different success outcomes, which were distinguishable from failures. In order to minimize these effects, we investigated other potential sources of failure closely, as described below.

4.2.1 Resource Load Sequence

Recall from Section 3.1, that test resource loads are initiated one after another in a random order. Because the test takes some time (see Table 6) to complete, there are a variety of conditions which can cause the test to abort prematurely. This suggests that the order in which resources are loaded may impact the error rate.

Figure 9 shows the overall failure rate versus load position (note that the first resource is at position 0). While the overall trend seems consistent with failures getting progressively worse with later resources, the sharp spike and then subsequent decline between positions 5 and 9 seems anomalous. In order to explore this further, we broke down the the failure rate by browser and operating system.

As Figures 10 and 11 make clear, Chrome and Firefox on Mac and Windows both show the same pattern of a failure spike around resources 5-8, whereas the same browsers on Linux (Figure 12) as well as both Safari and Internet Explorer show a generally linear trend (though the break around resource 9 for Internet Explorer is also puzzling). We leave the explanation of these anomalies for future work.

4.2.2 Latent UUIDs

Our analysis uncovered 3616 UUIDs for which we received completion signals without corresponding measurement page loads during the one-week experiment window. We refer to these UUIDs as **latent UUIDs** because we observed DNS and HTTP requests for FQDNS that included them in our logs prior to the start of our experiment window. There are two plausible explanations for the existence of latent UUIDs:

1. **Browser caching.** Modern web browsers cache users' recent and open tabs to allow for restoration of the browsing session in case of

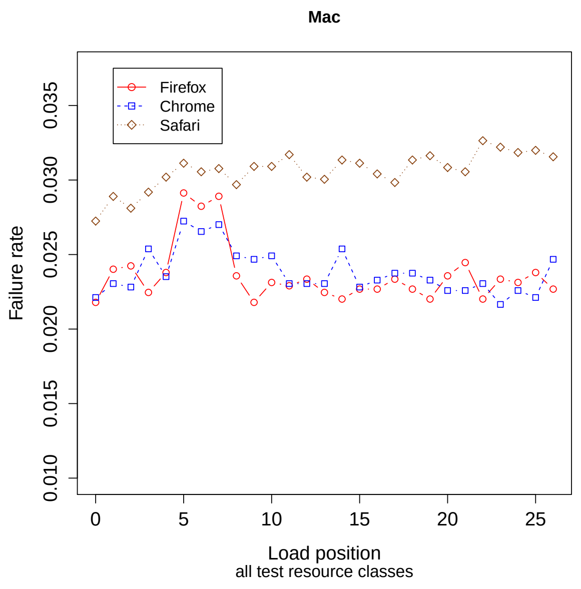

Figure 11: Failure rate versus load order across all test resources for Mac clients.

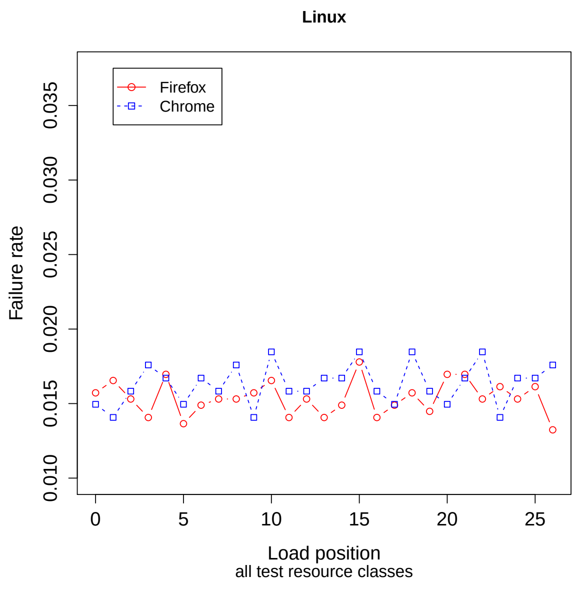

Figure 12: Failure rate versus load order across all test resources for Linux clients.

a crash, browser termination, or accidental tab closure. Half of the 18 latent UUIDs that had HTTP requests during the experiment window appeared within the first 33 hours of the experiment window, and 11 of them loaded the measurement page within the 24 hours leading up to the start of the experiment window. Thus, it is plausible that browser caching explains some of the latent UUIDs.

2. **DNS caching with eager renewal.** To improve DNS cache hit rates and, consequently, reduce client latency, Cohen and Kaplan [12] proposed a caching scheme wherein DNS caches issue unsolicited queries to authoritative nameservers for cached RRs whose TTLs have expired, even if no client queried for the RR at the time of the renewal. This mechanism is a documented feature in the Cisco IOS Distributed Director [1] and has been implemented by others [45]. Our log data strongly supports this explanation, as all latent UUIDs (by definition) appeared in the DNS logs, but only 18 had HTTP requests during the experiment window.

Latent UUIDs are not included in our analysis, as we cannot guarantee that our log data extends far enough into the past to cover them. Furthermore, our analysis only includes UUIDs for which we observed both the measurement page load and completion signal within the experiment window.

5 Discussion

The benefit from DNSSEC-signing a domain is upper-bounded by the number of clients which actually validate DNSSEC-signed records. As our measurements show, the fraction of clients which do so is less than 3%.[9] Moreover, this benefit is only obtained if DNSSEC either deters attacks or allows detection of attacks. By contrast, for a site with worldwide users, our results indicate that deploying DNSSEC in the current environment amounts to a self-inflicted partial attack on one's own site on the order of 0.2214% (the difference between the *goodsec* and *nosec* failure rates). For a site without significant Asian usage, the tradeoff looks more attractive, and for a site with largely Asian usage it looks less attractive.

The major source of increased failure rates from DNSSEC deployment appears to be that increased packet sizes force clients into DNS over TCP rather than DNS over UDP. The failure rate for DNS over TCP is approximately 10 times larger than DNS over UDP. This phenomenon is strongly localized to Asia/Pacific browsers.

Some potential future developments could change this calculation. First, a significant number of ISPs could deploy validating resolvers. As shown by the

[9]Here we interpret the *badsec* failure rate as an upper bound on the fraction of end users protected by DNSSEC validation because some fraction of the failures may be due to the fact that DNSSEC was enabled rather than validation failure

Comcast data in Section 4.1.5, unilateral deployment of DNSSEC by ISPs can have a very large impact on the behavior of their customers. If a few of the large ISPs were to deploy validating resolvers, our measured *badsec* failure rate would no doubt have been much higher and their customers would have obtained some level of defense against attackers outside the ISP's network. (Validation at the ISP resolver level does not provide defense against attackers located between the resolver and the customer.)

Second, there could be widespread deployment of a technology such as DANE that depends on DNSSEC. As mentioned above, DNSSEC for A records does not provide security against on-path attackers, who can intercept the traffic between the client and the server. Defending against such attackers requires some sort of cryptographic protocol such as SSL/TLS. By contrast, if DANE is used to attest to end-user certificates, then DNSSEC combined with DANE-based certificates can provide security against on-path attackers and thus a significantly greater benefit. Even with DANE, however, the collective action problem of simultaneous client and server deployment persists. In fact, it is worse since DANE's security requires that the client do DNSSEC validation — ISP-level validation is not sufficient.

Our results also serve as a caution for future researchers: Advertisement network based studies — especially those which attempt to measure success or failure — are very sensitive to variation in client and network behavior. In particular, there is significant variation both between browsers and operating systems and by request order within the same browser/operating system pair. These variations are of the same order of magnitude as the signal we are trying to measure and thus present a significant challenge. Additionally, they may indicate actual problems with the browsers — or at least opportunities for improvement. We are currently working with browser vendors to attempt to determine the reason for these anomalies.

6 Related Work

Several research groups have performed measurements related to DNSSEC.

The SecSpider project [35, 36, 43] has surveyed DNSSEC-secured zones since the DNSSEC rollout, quantifying the rollout using metrics of availability, verifiability, and validity. Deccio et al. [14, 15] surveyed representative DNSSEC domains for misconfiguration. Both of these projects focus on properties of the authoritative DNS server zone data, rather than the behavior of resolvers or caches.

Several research groups have attempted to characterize the overhead (to clients, servers, and the network) from deploying DNSSEC. Ager, Dreger, and Feldmann [2] used a trace of DNS traffic as the basis for a testbed experiment. They noted the possibility of overhead arising from packet fragmentation. Wijngaards and Overeinder [42] described the design of a DNSSEC resolver and validator and compared its performance to an ordinary DNS resolver. Migault, Girard, and Laurent [31] measured the overhead of DNSSEC resolution in a lab setting, including the NSEC3 option.

Gudmundsson and Crocker [18] measured the deployment of DNSSEC-capable resolvers using traces of DNS queries made to the `.org` servers. Glynn [17] surveyed DNSSEC deployment in Ireland, highlighting the possibilty that large responses would suffer fragmentation, and noting the geographic variation in client path MTU.

Dietrich [16] reports on a study of one component in DNSSEC resolution: users' home DSL routers. With the cooperating of network operators, the study's authors tested the behavior of 36 routers in a testbed environment. The DNS proxies in more than half of these routers were incompatible with DNSSEC; several of the tested routers could not be used with DNSSEC even if their internal DNS proxy were bypassed.

Herzberg and Shulman [19] describe several challenges to wide-scale DNSSEC deployment. They observe that large-response fragmentation not only reduces performance but can be the basis of downgrade and forgery attacks on permissive resolvers.

Pappas and Keromytis [37] performed a distributed measurement of resolution failures in the aftermath of the May 5, 2010 signing of the DNS root. They made resolution attempts from hundreds of geographically dispersed nodes (e.g., Tor exit nodes), which allowed them to observe the behavior of many DNS resolvers. Whereas their conclusions focused on the effects of rolling out DNSSEC on the DNS root-level servers, our measurements target the DNS in its steady state behavior nearly two years later.

Krishnan and Monrose [28] performed a large-scale measurement of browser DNS prefetching and characterized its security and privacy implications. Using a trace-based cache simulator, they showed that the additional overheads induced by prefetching would increase the overhead of deploying DNSSEC.

The Netalyzr platform [27] allows interested users to measure and report on properties of their Internet connection. Netalyzr has uncovered widespread DNS manipulation by ISPs [40, 41].

Zhang et al. [44] included client-side DNS measurement code in a software package used by millions. They identified several ISPs that manipulate DNS results, allowing them to proxy and modify Web searches.

Ager et al. [3] asked friends to run DNS measurement code on their systems. Their data anlysis focused on DNS performance.

Honda et al. [21] asked IETF colleagues to run a measurement tool, TCPExposure; this tool generated TCP segments with various properties, allowing Honda et al. to observe how middleboxes between clients and their servers handle different TCP extensions. Of all the related work, Honda et al.'s is the closest in spirit to ours. They sought to measure the compatibility of hypothetical future protocols with deployed middleboxes; we measure the interaction between DNSSEC and today's network infrastructure.

Rather than deploy custom software to users, we wrote JavaScript code that triggers DNS resolution, and served this code in an ad we placed with a display ad network. This strategy for enlisting users was pioneered by Barth, Jackson, and their coauthors [6, 25], who used it to measure the Web platform. This strategy was also recently used by Huston and Michaelson [23, 24] to measure the deployment of DNSSEC-capable resolvers and to describe their geographic distribution. Unlike our study, Huston did not also measure the prevalence of DNSSEC-intolerant resolvers.

7 Summary

While DNS name resolution is a key part of the Internet infrastructure, it has long been known that it is seriously insecure. DNSSEC is designed to repair that insecurity. We report on a large ad network based study designed to measure both the current state of deployment *and* the extent to which deploying DNSSEC-signed domains creates collateral damage in the form of failed resolutions of valid domains.

Our measurements confirm previous reports that DNSSEC deployment is proceeding quite slowly. Less than 3% of clients failed to retrieve resources hosted on DNSSEC-signed domains with broken signatures. This indicates that either these clients — or their resolvers — are not doing DNSSEC validation or they are not hard-failing on broken validations, which is effectively the same as not validating at all. Moreover, about 1.006% of clients fail to retrieve validly DNSSEC-signed resources (as compared to 0.7846% of unsigned resources. In other words, for every ten clients a site protects by using DNSSEC, it self-DoSes about one client. This effect is principally due to TCP fallback to accomodate larger DNSSEC packet sizes and is strongly localized to Asian users.

Finally, we report on a number of new measurement artifacts that can affect the results of advertising network based studies, including some browser-specific anomalies which may reveal opportunities for improvement in those browsers. In future work, we hope to explore further the specific causes of these anomalies.

Acknowledgements

The authors thank the anonymous reviewers and our shepherd, Tara Whalen. We also thank Duane Wessels, Casey Deccio, Cynthia Taylor, and Stephen Checkoway for their feedback on the paper, Philip Stark for suggestions about the analysis, and Collin Jackson for his help in acquiring an advertising network advertiser account. This material is based upon work supported by the MURI program under AFOSR Grant No. FA9550-08-1-0352.

References

[1] Distributed director cache auto refresh. `https://www.cisco.com/en/US/docs/ios/12_2t/12_2t8/feature/guide/ftrefrsh.pdf`.

[2] B. Ager, H. Dreger, and A. Feldmann. Predicting the DNSSEC overhead using DNS traces. In R. Calderbank and H. Kobayashi, editors, *Proceedings of CISS 2006*, pages 1484–89. IEEE Information Theory Society, Mar. 2006.

[3] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig. Comparing DNS resolvers in the wild. In M. Allman, editor, *Proceedings of IMC 2010*, pages 15–21. ACM Press, Nov. 2010.

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

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

[6] A. Barth, C. Jackson, and J. C. Mitchell. Robust defenses for cross-site request forgery. In P. Syverson and S. Jha, editors, *Proceedings of CCS 2008*, pages 75–88. ACM Press, Oct. 2008.

[7] R. Bellis. DNS Proxy Implementation Guidelines. RFC 5625 (Best Current Practice), Aug. 2009.

[8] R. Bellis. DNS Transport over TCP - Implementation Requirements. RFC 5966 (Proposed Standard), Aug. 2010.

[9] R. Bellis and L. Phifer. Test report: DNSSEC impact on broadband routers and firewalls. Online: `https://www.dnssec-deployment.org/wp-content/uploads/2010/03/DNSSEC-CPE-Report.pdf`, Sept. 2008.

[10] R. Braden. Requirements for Internet Hosts - Application and Support. RFC 1123 (Standard), Oct. 1989. Updated by RFCs 1349, 2181, 5321, 5966.

[11] Caida routeviews prefix to as mappings dataset (pfx2as). `http://www.caida.org/data/routing/routeviews-prefix2as.xml`.

[12] E. Cohen and H. Kaplan. Proactive caching of DNS records: Addressing a performance bottleneck. *Computer Networks*, 41(6):707–26, 2003.

[13] What are comcast's dynamic ip ranges? `http://postmaster.comcast.net/dynamic-IP-ranges.html`.

[14] C. Deccio, J. Sedayao, K. Kant, and P. Mohapatra. A case for comprehensive DNSSEC monitoring and analysis tools. In R. Clayton, editor, *Proceedings of SATIN 2011*, Apr. 2011. Online: `http://conferences.npl.co.uk/satin/agenda2011.html`.

[15] C. Deccio, J. Sedayao, K. Kant, and P. Mohapatra. Quantifying and improving DNSSEC availability. In G. Rouskas and X. Zhou, editors, *Proceedings of ICCCN 2011*. IEEE Communications Society, July 2011.

[16] T. Dietrich. DNSSEC support by home routers in Germany. Presented at RIPE 60, May 2010. Online slides: `http://ripe60.ripe.net/presentations/Dietrich-DNSSEC_Support_by_Home_Routers_in_Germany.pdf`.

[17] W. J. Glynn. Measuring DNS vulnerabilities and DNSSEC challenges from an irish perspective. In R. Clayton, editor, *Proceedings of SATIN 2011*, Apr. 2011. Online: `http://conferences.npl.co.uk/satin/agenda2011.html`.

[18] Ó. Gudmundsson and S. D. Crocker. Observing DNSSEC validation in the wild. In R. Clayton, editor, *Proceedings of SATIN 2011*, Apr. 2011. Online: `http://conferences.npl.co.uk/satin/agenda2011.html`.

[19] A. Herzberg and H. Shulman. Towards adoption of dnssec: Availability and security challenges. Cryptology ePrint Archive, Report 2013/254, 2013. `http://eprint.iacr.org/`.

[20] P. Hoffman and J. Schlyter. The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA. RFC 6698 (Proposed Standard), Aug. 2012.

[21] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda. Is it still possible to extend TCP? In P. Thiran and W. Willinger, editors, *Proceedings of IMC 2011*, pages 181–94. ACM Press, Nov. 2011.

[22] L.-S. Huang, E. Y. Chen, A. Barth, E. Rescorla, and C. Jackson. Talking to yourself for fun and profit. In H. J. Wang, editor, *Proceedings of W2SP 2011*. IEEE Computer Society, May 2011.

[23] G. Huston. Counting DNSSEC. Online: `https://labs.ripe.net/Members/gih/counting-dnssec`, Sept. 2012.

[24] G. Huston and G. Michaelson. Measuring DNSSEC performance. Online: `http://www.potaroo.net/ispcol/2013-05/dnssec-performance.html`, May 2013.

[25] C. Jackson, A. Barth, A. Bortz, W. Shao, and D. Boneh. Protecting browsers from DNS rebinding attacks. *ACM Trans. Web*, 3(1), Jan. 2009.

[26] jquery: The write less, do more, javascript library. `http://jquery.com`.

[27] C. Kreibich, B. Nechaev, N. Weaver, and V. Paxson. Netalyzr: Illuminating the edge network. In M. Allman, editor, *Proceedings of IMC 2010*, pages 246–59. ACM Press, Nov. 2010.

[28] S. Krishnan and F. Monrose. An empirical study of the performance, security and privacy implications of domain name prefetching. In S. Bagchi, editor, *Proceedings of DSN 2011*, pages 61–72. IEEE Computer Society and IFIP, June 2011.

[29] P. Leach, M. Mealling, and R. Salz. A Universally Unique IDentifier (UUID) URN Namespace. RFC 4122 (Proposed Standard), July 2005.

[30] J. Livingood. Comcast completes dnssec deployment. `http://blog.comcast.com/2012/01/comcast-completes-dnssec-deployment.html`.

[31] D. Migault, C. Girard, and M. Laurent. A performance view on dnssec migration. In H. Lutfiyya and Y. Diao, editors, *Proceedings of CNSM 2010*, pages 469–74. IEEE Communications Society, Oct. 2010.

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

[33] P. Mockapetris. Domain names - implementation and specification. RFC 1035 (Standard), 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, 5936, 5966, 6604.

[34] nginx httplogmodule. `http://wiki.nginx.org/HttpLogModule`.

[35] E. Osterweil, D. Massey, and L. Zhang. Deploying and monitoring DNS security (DNSSEC). In C. Payne and M. Franz, editors, *Proceedings of ACSAC 2009*, pages 429–38. ACM Press, Dec. 2009.

[36] E. Osterweil, M. Ryan, D. Massey, and L. Zhang. Quantifying the operational status of the DNSSEC deployment. In K. Papagiannaki and Z.-L. Zhang, editors, *Proceedings of IMC 2008*, pages 231–42. ACM Press, Oct. 2008.

[37] V. Pappas and A. D. Keromytis. Measuring the deployment hiccups of DNSSEC. In J. L. Mauri, T. Strufe, and G. Martinez, editors, *Proceedings of ACC 2011*, volume 192 of *CCIS*, pages 44–53. Springer-Verlag, July 2011.

[38] S. Son and V. Shmatikov. The hitchhiker's guide to DNS cache poisoning. In S. Jajodia and J. Zhou, editors, *Proceedings of SecureComm 2010*, volume 50 of *LNICST*, pages 466–83. Springer-Verlag, Sept. 2010.

[39] P. Vixie. Extension Mechanisms for DNS (EDNS0). RFC 2671 (Proposed Standard), Aug. 1999.

[40] N. Weaver, C. Kreibich, B. Nechaev, and V. Paxson. Implications of Netalyzr's DNS measurements. In R. Clayton, editor, *Proceedings of SATIN 2011*, Apr. 2011. Online: `http://conferences.npl.co.uk/satin/agenda2011.html`.

[41] N. Weaver, C. Kreibich, and V. Paxson. Redirecting DNS for ads and profit. In N. Feamster and W. Lee, editors, *Proceedings of FOCI 2011*. USENIX, Aug. 2011.

[42] W. C. Wijngaards and B. J. Overeinder. Securing DNS: Extending DNS servers with a DNSSEC validator. *Security & Privacy*, 7(5):36–43, Sept.–Oct. 2009.

[43] H. Yang, E. Osterweil, D. Massey, S. Lu, and L. Zhang. Deploying cryptography in Internet-scale systems: A case study on DNSSEC. *IEEE Trans. Dependable and Secure Computing*, 8(5):656–69, Sept.–Oct. 2011.

[44] C. Zhang, C. Huang, K. W. Ross, D. A. Maltz, and J. Li. Inflight modifications of content: Who are the culprits? In C. Kruegel, editor, *Proceedings of LEET 2011*. USENIX, Mar. 2011.

[45] Z. Zhang, L. Zhang, D.-E. Xie, H. Xu, and H. Hu. A novel dns accelerator design and implementation. In C. S. Hong, T. Tonouchi, Y. Ma, and C.-S. Chao, editors, *Proceedings of APNOMS 2009*, volume 5787 of *LNCS*, pages 458–61. Springer-Verlag, Sept. 2009.