Deployment Models for Backup Certificate Systems

Eric Rescorla
ekr@rtfm.com

NIST Online Trust Workshop 2013
Problem Statement

• We are concerned only with SSL/TLS servers
• The primary source of SSL/TLS identity will continue to be certificates
• Some certificates are incorrectly issued
  – CAs make mistakes
  – Sometimes CAs are compromised (or misbehave)
• Focus is minimizing the impact of misissuance
Lots of work in this area

- DANE [RFC6698]
- Certificate Transparency [draft-laurie-pki-sunlight]
- HPKP [draft-ietf-websec-key-pinning] and TACK [draft-perrin-tls-tack]
- Perspectives, Sovereign Keys, Convergence
- A bunch of survey-type ideas

What is it going to take to get widespread deployment?
Questions for Analysis

• Who needs to change their behavior? (RPs, servers, CAs, …)
• What are the benefits?
• Who gets the benefits?
• What other technology does this depend on?
• What are the downside risks?
DANE Overview (usages 0 and 1)

- Server operators publish TLSA records in DNS
  - Records can contain:
    - A CA certificate/key that must be in the path
    - An EE certificate/key that must be used by the server
  - Records MUST be authenticated via DNSSEC

- When client visits www.example.com
  - Tries to resolve a TLSA record for _443._tcp.www.example.com
  - If present, then do *both* the PKIX checks and the DANE checks
  - If absent, do *just* the PKIX checks
## DANE Deployment Summary

<table>
<thead>
<tr>
<th>Changes needed</th>
<th>Browser, server, server’s DNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benefits</td>
<td>Prevention</td>
</tr>
<tr>
<td>Scope</td>
<td>When server and client both deploy</td>
</tr>
<tr>
<td>Dependencies</td>
<td>DNSSEC deployment at clients, servers, <em>and intermediaries</em></td>
</tr>
<tr>
<td>Risks</td>
<td>Self-DoS via incorrect TLSA records</td>
</tr>
<tr>
<td></td>
<td>DNSSEC increases rate of ordinary resolution failure</td>
</tr>
<tr>
<td></td>
<td>“False positives” because of broken intermediaries</td>
</tr>
</tbody>
</table>
What happens if something goes wrong with DNSSEC?

• Example: RRset is supposed to be signed but no RRsig present
  – RP cannot tell whether a TLSA record should be present

• How can this happen?
  – Server error
  – Broken intermediaries (e.g., filter DNSSEC records)
  – An active attack

• Options
  – Assume attack and terminate the connection
  – Assume everything is OK and proceed
What does the specification say?

“An attacker who is able to divert a user to a server under his control is also likely to be able to block DNS requests from the user or DNS responses being sent to the user. Thus, in order to achieve any security benefit from certificate usage 0 or 1, an application that sends a request for TLSA records needs to get either a valid signed response containing TLSA records or verification that the domain is insecure or indeterminate. If a request for a TLSA record does not meet one of those two criteria but the application continues with the TLS handshake anyway, the application has gotten no benefit from TLSA and SHOULD NOT make any internal or external indication that TLSA was applied.” [RFC 6698; §4.1]

This is not plausible in most UAs; they must either succeed or fail.
How common are DNSSEC failures?

- DANE requires that endpoints validate DNSSEC
- In 2010 many consumer routers didn’t properly proxy DNSSEC resolution [Dietrich 2010]
  - 16 out of 33 had some DNSSEC support
  - Only 9 worked with packets > MTU
- All routers worked if you bypassed DNSSEC proxy
- Unclear how much has changed in 3 years
- How can client tell what he is behind?
- What about ISP behavior?
Impact of using DNSSEC at all

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 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 question we categorized each test client by geographic area based on its resolver’s IP address. We used the CAIDA prefix to AS mapping dataset [12] to determine the Autonomous System Number (ASN) for each 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.

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 (p < 0.0001) between badsec domains and other domains, only APNIC (Asia Pacific) shows a significant difference between nosec and goodsec (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. Our analysis of relative failure rates among RIRs employed the two-proportion z-test.

4.1.2 The Impact of Packet Size and TCP Fallback

One commonly-expressed concern with DNSSEC is that it increases the size of resolver responses and thus increases 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 bytes defined in RFC 1035 [30].

- Clients can advertise a larger maximum UDP datagram size via the EDNS0 OPT pseudo-RR [35].
- Resolution can fall back to TCP if the server supports it.

Unfortunately, both of these mechanisms can cause problems for some intermediaries [7,8,11]. Our experiment allows us to directly measure these effects. Our measurements suggest that packet size is a major contributor to failures: Out of all the resources for which we served DNS for the test resource, the failure rate for goodsec test resources whose DNS resolution fell back to TCP was 6.011%, approximately, 10 times the the failure rate of those that completed over UDP, 0.6127%. For nosec domains, lookups never fell back to TCP and the failure rate was 0.6% for UDP. The similar UDP failure rates for nosec and goodsec suggest that the major cause of the excess failures we observe with goodsec is errors when the client has fallen back to TCP and only indirectly due to the increase in packet size which caused the fallback.

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

Source: Lian et al. (in submission, 2013)
User Agent Vendor Incentives?

- UAs must decide whether to implement and rely on DANE
  - Users rarely change the system defaults
  - And this has real costs
- UA vendors have no control over the user’s network environment
  - And this environment can change when the user moves
- The network environment changes very slowly
  - Even for much more compelling applications like voice and video
  - ... which still need extensive NAT/firewall traversal mechanisms
HPKP Overview

• Server can provide a Public-Key-Pins or Public-Key-Pins-Report-Only HTTP header
  – Lists hashes of public keys which must appear in the server’s certificate chain
  – Client remembers these hashes ("pins" for future use)

• Once pinned client does both PKIX checks and verifies that one of the pinned keys is present

• If pin check fails
  – Public-Key-Pins → fail
  – Public-Key-Pins-Report-Only → report error to report-uri

• TACK is conceptually similar but operates at the TLS layer not the HTTP layer
# HPKP Deployment Summary

<table>
<thead>
<tr>
<th>Changes needed</th>
<th>Browser (already in Chrome, under development in Firefox), server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benefits</td>
<td>Prevention (or detection)</td>
</tr>
<tr>
<td>Scope</td>
<td>When server and client both deploy</td>
</tr>
<tr>
<td>Dependencies</td>
<td>None</td>
</tr>
<tr>
<td>Risks</td>
<td>Attack on first use</td>
</tr>
<tr>
<td></td>
<td>Self-DoS via incorrect pinning</td>
</tr>
</tbody>
</table>
Self-DoS via Incorrect Pinning

- What happens if a host has key X and advertises a pin for key Y
  - This would create a self-DoS for the duration of the pin
- Complete PIN failures are unrecoverable for pin lifetime
  - Pin lifetimes need to be long in order to work
- HPKP has three mechanisms to prevent this
  - Current connection must be valid with proposed pin
  - Must advertise multiple keys ("backup pin")
  - Report-only mode allows server to discover all keys currently in use
HPKP Server Incentives

- Publishing a pin provides security with set of pin-verifying clients
  - Currently about 20-30% of user’s browsers
- Primary risk is self-DoS
- Currently very few sites publish pins
  - About 300 static pins in Chromium
  - Unknown how many published pins (but rumor is it is small)
    - Less than 1000 HSTS sites [Ristic 2013]
- Why is this number so low?
Certificate Transparency Overview

• Participating CAs publish all certificates they issue
  – Provide servers with proofs*

• Clients check whether certificates have a proof of publication
  – Assuming they are supposed to
  – Any certificate which should have a proof but does not is rejected

• Server operators (or some service) can check for certificates which should not exist
  – Did someone else obtain a certificate for my domain?
  – Actually dealing with misissued certificates is out of scope

*Insert crypto magic here
# CT Deployment Summary

<table>
<thead>
<tr>
<th>Changes needed</th>
<th>Browser, server, notary service, CA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benefits</td>
<td>Detection</td>
</tr>
<tr>
<td>Scope</td>
<td>Participating CAs and servers who check</td>
</tr>
<tr>
<td>Dependencies</td>
<td>Robust revocation (currently nonexistent)</td>
</tr>
<tr>
<td>Risks</td>
<td>Breakage of non-participating CAs (whenever CT is required)</td>
</tr>
</tbody>
</table>
Limits of CT Detection

- CT detects misissuance by *participating CAs*
  - Does nothing about non-participating CAs
- Server isn’t primarily worried about misissuance by *his CA*
  - ... but he is worried about other CAs
  - And it can’t control them
- Attacker can pick any CA to attack
  - Attacker difficulty is security of the weakest non-participating CA
  - Poorly run CAs seem likely not to participate
CT Deployment Incentives

- Requires participation by CAs
  - In principle servers can self-publish
  - ... but clients need to know when proofs are expected
- CAs have little incentive to participate
  - Unless browsers require proofs from all CAs
  - ... which breaks the world
- Browsers can only require proofs once nearly all CAs already publish
- Classic collective action problem
Summary

• None have seen widespread deployment

• All have severe collective action problems

• Minimally, need support on both clients and servers
  – CT and DANE both require support elsewhere

• Hard to deploy any of these without breaking stuff
Questions?