Certificate Pinning Extension for HSTS

By Chris Evans cevans@google.com and Chris Palmer palmer@google.com
Last updated 12 Sep 2011

Introduction

We propose to extend the HSTS HTTP header to enable a web site to express to UAs which certificate(s) UAs may expect to be present in the site's certificate chain in future connections. We call this "certificate pinning". The Chrome/ium browser ships with a static set of pins, and individual users can extend the set of pins (chrome://net-internals/#hsts). Although effective, this does not scale. This proposal addresses the scale problem.

Deploying certificate pinning safely will require operational and organizational maturity due to the risk that HSTS Hosts may "brick" themselves by pinning to a certificate that becomes invalid. We discuss potential mitigations for those risks. We believe that, with care, site operators can greatly reduce the risk of MITM attacks and other false-authentication problems for their users without incurring undue risk of bricking the site.

This document extends the version of HSTS defined in http://tools.ietf.org/html/draft-ietf-websec-strict-transport-sec-02 and follows that document's notational and naming conventions.

Server and Client Behavior

To set a pin, HSTS Hosts use a new STS extension directives (*STS-d-ext*) in their HSTS response header field: *pins*.

```
STS-d-ext-pin       =     “pins” OWS “=” OWS [fingerprints]
fingerprints        =     fingerprint
                          / fingerprint “,” fingerprints
fingerprint         =     fp-type “/” base64-digits
fp-type             =     “sha1”
                          / “sha256”
```

Here is an example response header field using the *pins* extension:

```
Strict-Transport-Security: max-age=500; includeSubDomains; pins=sha1/
4n972HfV354KP560yw4uqe/baXc=,sha1/IvGeLsbqzPxdI0b0wuj2xVTdXgc=
```

Upon receipt of this header field, the UA will note the HSTS Host as a Known Pinned HSTS Host. When connecting to a Known Pinned HSTS Host, the UA will compare the public key fingerprint(s) in the Host’s certificate chain to the pinned fingerprints, and will fail closed unless at least one public key in the chain has a fingerprint matching one of the pinned fingerprints. (Following the HSTS specification, TLS errors for HSTS sites must be hard, with no chance for the user to click through any warnings or errors. We treat fingerprint mismatch in the same way.)

The pin list appearing in an HSTS header MUST have at least one pin matching one of the public key fingerprints in the chain that was validated for the HTTPS connection. This defends against HTTP header injection attacks (see below).

UAs MUST cache pins for Known Pinned HSTS Hosts, and MIGHT AS WELL do so in the same manner as other HSTS metadata. If the *maxAge* directive is present in the HSTS response header, the HSTS metadata — including fingerprints in the *pins* directive — expire at that time.

Revocation

In the event of pin mismatch, clients MUST check whatever revocation mechanism is available, and attempt to discover whether the certificate with the mismatching fingerprint has been revoked. For example:

```
offending_certificate = cert in certificate_chain where cert.public_key.fingerprint ==
mismatched_fingerprint

revoked = offending_certificate.serial_number in revoked_serials
```

If the offending certificate has been revoked, it is unpinned and the UA can re-evaluate the pin list. If there are no pinned fingerprints left on the pin list, the browser downgrades the host from Known Pinned HSTS Host to Known HSTS Host.

The revocation mechanism could be an extant mechanism such as CRL or OCSP, or a new one such as browser updates, whitelists, blacklists, or an in-built CRL. This document is agnostic about the revocation mechanism(s) UAs may use.

Un-pinning

Certificates that are invalidated for any reason and by any means — revocation by extant or future means, expiration, blacklisting — are effectively removed from the pin list. Certificates whose intermediary or root signers are revoked are also effectively invalidated and removed from the pin list.

HSTS Hosts can un-set pins in clients (*un-pin*) by setting a *pins* directive that contains no pins. UAs MUST NOT obey the un-pinning directive unless the empty *pins* directive is set on a response in a TLS connection that was authenticated with one of the previously pinned public keys.

Risks of Pinning, and Mitigations

Deployment Guidance

To recover from disasters of various types, as described below, we recommend that HSTS Hosts follow these guidelines.

- Have a safety net. Generate a backup key pair, get it signed by a different (root and/or intermediary) CA than your live certificate(s), and store it safely offline. Set this *backup pin* in your *pins* directive.
    - Having a backup certificate was always a good idea anyway.
- It is most economical to have the backup pin signed by a completely different signature chain than the live certificate, to maximize recoverability in the event of either root or intermediary signer compromise.
- Periodically exercise your backup pin plan — an untested backup is no backup at all.
- Have a diverse certificate portfolio. Pin to a few different roots — owned by different companies if possible.
- Start small. Set a *maxAge* of minutes or a few hours. Gradually increase *maxAge* as you gain confidence in your operational capability.
    - But your backup pin won't fail, because you periodically test it.
- Pre-establish contacts at browser vendors to discuss last-ditch options and response time.

Disasters Relating to Compromises of Certificates

Disaster: the private key for your pinned leaf is lost or stolen.

- If you set a backup pin, you have a smooth transition. Deploy it.
- You SHOULD attempt to get the certificate revoked by whatever means available (extant revocation mechanisms like CRL or OCSP, blacklisting in the UA, or future revocation mechanisms). If the browser is able to learn of the revocation/blacklisting, it will un-pin the certificate.
    - Note that extant revocation mechanisms are known to be unreliable. Do not depend on them.

- If you also pinned to roots or intermediaries, simply get a new leaf re-issued from one of those.
    - You could also pin multiple leaves, as well as to multiple signers, and have the spare leaf ready to go.

Disaster: your root or intermediary CA is compromised.

- This disaster will affect many sites (HSTS Hosts and other), and will likely require a client software update (e.g. to revoke the signing CA and/or the false certificates it issued).
- Certificates that are invalidated for any reason and by any means are effectively un-pinned, allowing sites to gracefully degrade from Known Pinned HSTS Host to Known HSTS Host.
- If you have a backup pin whose signature chain is still valid, deploy it. In this case, your site need not even degrade from Known Pinned to Known.

Disasters Relating to Certificate Mismanagement

Disaster: your leaf certificate expires.

- Deploy your backup pin.
- Note that when evaluating a pinned certificate, the UA MUST un-pin the fingerprint if the certificate has expired. If a pin list becomes empty, the UA downgrades the host from Known Pinned HSTS Host to Known HSTS Host. The usual HTTPS validation procedure now applies.
- Get any CA to sign a new cert with updated expiry, based on your existing, unchanged public key.
    - And/or, deploy your backup pin and/or have a CA sign an all-new key.
    - Continue to set pins in your HSTS header, and UAs will upgrade from Known HSTS Host to Known Pinned HSTS Host when the fingerprint(s) refer(s) to valid certificate(s) again.
- Beware that if you pin to a cert that expires plus a non-expired cert for which you have lost your key pair, you're in trouble.

Disaster: your CA is extorting you approaching renewal / expiry time.

- If your backup pin chains to a different signer, deploy it. (Then get a new backup pin.)
- The time running up to renewal can be used to serve additional HSTS public key hashes, pinning to new root CAs.
    - Hosts can also disable pinning altogether as described above.
- If you are pinned to leaves or your own intermediary, you can simply get a different root CA to sign your existing public key.
- If you fail to get new certs in time, and you are pinned only to the one root CA, the solution is simple; see the section on leaf certificate expiry above.

Disasters Relating to Vulnerabilities in the Known HSTS Host

Disaster: your site has HTTP header injection.

- The attacker could set additional pins for certificates he controls, for arbitrary *maxAge*s, allowing him to undetectably MITM clients.
    - Combined with a successful MITM attack later, the attacker could brick the site (for particular users in the scope of the MITM) by pinning only fingerprints that he controls.
- The attacker could disable HSTS and pins.
- Header injection vulnerabilities are in general more severe than merely disabling pinning for individual users.

Disaster: your site suffers full server-side compromise.

- The attacker could set pins for public keys they control — including, now, the host's formerly-legitimate keys — and set a high *maxAge* to get clients to pin to the attacker's impostor site for a long time. For as long as the attacker can get UAs to visit the impostor site rather than the true site, UAs will believe they are pinned to a legitimate host. (After the attacker loses the power to direct UAs to the impostor site, but before the pin hits its *maxAge*, the result is DoS.) Recovering from this is likely to require extraordinary measures such as pin revocation (see Ideas, below).
- Because the solution is pin revocation (see below), this is a catastrophic failure.

Interactions With Built-in HSTS Lists

UAs MAY choose to implement built-in certificate pins, alongside any built-in HSTS opt-in list. UAs MUST allow users to override a built-in pin list, including turning it off.

Where HSTS response header field directives conflict with built-in HSTS metadata, the response header overrides the built-in metadata. If the response header is partial, e.g. if it lacks a *pins* directive, the built-in metadata (e.g. pins) still applies. UAs SHOULD implement this behavior by first initializing HSTS metadata from the built-in set (if any), then update that set with saved metadata from HSTS response header fields, and then continually update the set as new HSTS response header fields arrive.

Usability Considerations

When pinning works to detect impostor Known Pinned HSTS Hosts, users will experience denial of service. UAs SHOULD explain the reason why. If it happens that true positives (actual attacks) outnumber false positives (sites bricking themselves by accident), the feature will gain a positive reputation. Note that pinning has started life with a good reputation because it provoked the discovery of the DigiNotar CA compromise. (When DigiNotar signed a certificate for *.google.com in August 2011, Chrome users discovered the attack due to the pre-loaded pins for Google domains.)

We believe that, in general, DoS is a better failure mode than user account/session compromise or other result of TLS compromise.

UAs MUST have a way for users to clear current pins that were set by HSTS. UAs SHOULD have a way for users to query the current state of Known (Pinned) HSTS Hosts.

Economic Considerations

If pinning becomes common, site operators might become incentivized to choose CAs that get compromised less often, or respond better to compromise. This will require information to flow into the market, and for people to interpret no news post-compromise as bad news. Pinning itself will provide some of that information, as will sources like UA vendor communications, the EFF SSL Observatory, the Qualys SSL survey, etc.

The disaster recovery plans described above all incur new costs for site operators, and increase the size of the certificate market. Arguably, well-run sites had already absorbed these costs because (e.g.) backup certificates from different CAs were necessary disaster recovery mechanisms even before certificate pinning. Small sites — which although small might still need to provide good security — may not be able to afford the disaster recovery mechanisms we recommend. (The cost of the backup certificate is not the issue; it is more the operational costs in safely storing the backup and testing that it works.) Thus, low-risk pinning may be available only to large sites; small sites may have to choose no pinning or potentially bricking their site (up to the *maxAge* window). This is not worse than the status quo.

Ideas

Pin Revocation

To un-brick hosts in a less-unscalable way, the UA vendor could send the UA updates about pins to revoke, and then clients note the host as Known HSTS instead of Known Pinned HSTS. Host operators could request of UA vendors to revoke a pin. This could be part of agl's CRL replacement plan, or it could be a different update system. Messages could be signed with a private key matching a public key baked into the UA.

Any revocation-related idea interacts with agl's proposed replacement for CRLs/OCSP: https://docs.google.com/a/google.com/document/d/1Lndc89nZDm8MMsbR6Hgee9xM2EbVWLKLsJwsTvUz5E4/edit?hl=en_US

Requiring Backup Pins

Because bricking risk mitigation requires a backup pin, UAs could require that the pins directive have at least two fingerprints, at least one of which does not match any of the public keys in any of the certificates in the chain. (This idea due to tsepez.)

Prepopulating Pin Lists with Googlebot

To deal with the bootstrap problem, why not inform Chrome of what Googlebot sees?

Tools to Assist Creation of Header (nlidz)

Provide tools that take x509 certificates and offer header to add to webserver configuration. Provide webserver / webapp helpers that would ensure header is safe (I guess this is taken care of by the fallback in case of signature mismatch on first pinning encounter) and working.

Visibility Into the Chain of Trust (nlidz)

For full protection, websites need to pin external resources loaded (like ssl.gstatic.com for mail.google.com). It gets to be a similar battle to mixed content.

Pinning Subresources

Many sites have pages that load subresources from domains not under the control, or under only partial control, of the main site's operators. For example, popular sites often use CDNs, and CDN customers may have only limited, if any, ability to influence the configuration of the CDN's servers. (This long-standing problem is independent of certificate pinning.)

To a limited extent, the *includeSubDomains* HSTS directive can address this: if the CDN site has a name that is a subdomain of the main site (e.g. assets-from-cdn.example.com points to CDN-owned servers), and if the main site's operators can guaranteeably keep up-to-date with the CDN's server certificate fingerprints — perhaps as part of example.com's contract with the CDN — then the problem may be solved.

CDNs MAY, and SHOULD, also use certificate pinning independently of any of their customers.

Although one can imagine an extension to this specification allowing the main resource to set pins for subresources in other domains, it is complex and fragile both from technical and business perspectives. The UA would have to accept those pins for the subresource domains ONLY when loading resources from the subdomains as part of a page load of the main site. The independence of the two domains' operations teams would still pose synchronization problems, and potentially increase the bricking risk.

Therefore, except in simple cases, this document leaves the cross-domain subresource problem to future work. Operational experience with HSTS-based certificate pinning should guide the development of a plan to handle the problem.

Pinning Without Requiring HTTPS

Some site operators would like to take advantage of certificate pinning without requiring HTTPS, but having clients require pins in the event that they do connect to the site with HTTPS. As specified above, the current HSTS-based mechanism does not allow for this: clients that receive the *pins* directive via HSTS will also therefore require HTTPS — that is the purpose of HSTS after all. To have an additional directive, e.g. *mode*=optional, would not work because older clients that support HSTS but not the *mode* extension would effectively require HTTPS.

Alternatives include (a) putting the *pins* directive in a new header instead of extending HSTS; and (b) some kind of hack like setting *maxAge*=0 and having an additional directive to keep the pins alive (e.g. *pinMaxAge*). These alternatives seem ugly to us and we welcome suggestions for a better way to support this deployment scenario.