Increasing HTTP Transport Confidentiality with TLS Based Alternate Services

Patrick McManus <pmcmanus@mozilla.com>
Mozilla Firefox Networking Module Owner
co-author of draft http://tools.ietf.org/html/draft-nottingham-httpbis-alt-svc-01
For the W3C/IAB workshop on Strengthening the Internet Against Pervasive Monitoring

Postulate #1: Internet eavesdropping is an attack. [http://tools.ietf.org/html/draft-farrell-perpass-attack-03]
Postulate #2: Existing protocols enable this beacuse we were naive and we can do better. [http://tools.ietf.org/html/draft-thomson-perpass-statement-00]
Postulate #3: https:// is the best solution - but it is not mandatory and is only chosen by ¼ of web traffic by volume

PART I: Encryption As The New PlainText

Alternate Services (http://tools.ietf.org/html/draft-nottingham-httpbis-alt-svc-01) defines a mechanism for routing http:// URIs (which are of course traditionally carried over plain text) over TLS encrypted HTTP/2 channels. This creates an encrypted but un-authenticated transport that servers can deploy with a simple software upgrade. This is more robust than plaintext in passive eavesdropping environments. It does not have strong dependencies on the global PKI or name services so It can be implemented using a zeroconf server deployment experience in a way that https:// configurations cannot offer.

This is an effort to encrypt some of the current plaintext web by lowering the traditional barrier to entry. It has obvious limitations. Do not let this effort distract you from the most important security win we know how to get: https:// is more secure than http:// in every way and failing to adopt https is generally a mistake. However, as others have noted, the web makes this mistake all the time. The decision of whether or not to run https:// is made almost unilaterally by the server and there is ample data to suggest that ¾[1] of the time the server declines to embrace the full true security it offers.

As part of our efforts to bring HTTP/2 to the Web, Mozilla has been clear we should not implement a new plain text web transport protocol. **In my opinion the IETF should recognize that the privacy needs of human participants on the Web combined with the insecure infrastructure that it runs on top of demand that conforming HTTP/2 applications always utilize encryption.**

TLS provides an obvious and well vetted mechanism for achieving more encryption. The core question is whether to restrict this to https:// or to extend it to http:// with a relaxed un-authenticated profile.

I have created an experimental build of Firefox that implements unauthenticated encryption for http:// URIs over HTTP/2. This should help us gain experience and data, though at this time of writing the implementation is still brand new. Its future inclusion in the product is undecided and will be informed by this experience. I'm actively seeking partners in experimentation and I hope the STRINT workshop would be a good place to find like minded technologists

The build currently uses the plain text HTTP/1 response header defined by Alternate-Services draft for bootstrapping and implements the "h2t" profile from http://tools.ietf.org/html/draft-nottingham-http2-encryption-02. I'm interested in exploring methods other than HTTP/1 as a bootstrap as well - see the DNS discussion at the end of this position paper.

Links to the most current builds are included at the bottom of our HTTP/2 progress wiki page at https://wiki.mozilla.org/Networking/http2

[1] telemetry.mozilla.org

PART II: Unauthenticated TLS Is Not An Easy Choice

"A Journey of 1000 miles begins with a single step"

vs.

"First, do no harm."

I have great respect for my peers who believe that rather than pursuing encryption for http:// we should be promoting true security via https-only. Some of my own team in included in that camp - their world is one I wish to live in. However, I personally believe that the realities of today's Internet mean significant good will be done in the short term by taking action and bringing un-authenticated ciphertext to http:// through simple server upgrades. The alternative is to wait for the web to transition its content to https://.

Let's brainstorm some historical reasons why http:// is used instead of https:// for HTTP/1:

1. https:// requires interaction with the PKI, and the PKI is hard to use. The PKI has monetary costs, expertise requirements, and maintenance burdens that carry operational risks such as expiring certificates taking your website offline without notice.
2. The PKI uses a consistent global namespace and many firewalled environments are not deployed that way. (e.g. printer.localdomain) Embedded devices may not be able to deploy https://, and thus HTTP/2, at all if https:// is the only HTTP/2 path.
3. Operational costs of https:// are perceived to be higher - CPU, memory, hosting charges, bandwidth overhead, etc..
4. https:// can be slower - RTT dependent handshake overhead (especially bad on mobile), third party revocation checking, and incremental bandwidth costs all matter.
5. https:// hosting requires ever scarcer global IP addresses and is incompatible with name based virtual hosting because a small, but significant, amount of traffic is initiated with legacy non SNI capable clients.
6. The mixed-content security policies of browsers prohibit https:// based resources from including some types of subresources (e.g. javascript or websockets) from http:// based origins. Sites that have a dependency on a third party that does not offer https:// are blocked from transitioning themselves.
7. Operational models that rely on data inspection - perhaps for routing, cookie management, auditing, or data loss prevention.

How much of the above reasoning is disrupted by the introduction of encrypted http:// and the current state-of-the-art TLS?

1. [PKI is hard] Removing the authentication requirements removes this objection.
2. [Certs have naming and management challenges] Removing the authentication requirements removes this objection and makes HTTP/2 suitable for embedded applications.
3. [Operational Costs] This concern applies equally to TLS in both an https and http context. The situation is organically improving as very large deployments have proven that it is possible to economically operate in a TLS context (https://www.imperialviolet.org/2011/02/06/stillinexpensive.html). TLS for http:// may provide an opportunity for operators to reassess whether or not this is still a legitimate concern.
4. [Speed] http:// with TLS has a speed advantage over https:// with respect to revocation checking as there is no need to check the revocation status of an unauthenticated certificate. However it is worth noting that many of the other speed considerations of TLS deployment have been undergoing recent improvements in ways that would help all TLS deployments: false start, ocsp stapling, and CRL sets are all improving responsiveness of TLS. TLS 1.3 has similar goals.
5. [SNI] Alternate-Services provides a significant advantage to http:// over traditional https:// with respect to the SNI objection because it is only re-routing HTTP/2. HTTP/2 aware clients are required to implement SNI with TLS. This property means that a server can reliably do name based hosting

over TLS without creating a backwards compatibility problem with plaintext legacy HTTP/1 clients.

6. [Mixed Content] Maintaining http:// URIs in a TLS context also means those pages maintain their same origin and security policies associated with that origin - there are no new security guarantees or requirements without authentication. That means that sites with mixed-content constraints can continue to run http:// schemes over TLS without triggering mixed-content blocking.
7. [Content Owner Relies on Data Transparency] The content owner is in a position to actively manipulate the data stream and execute the equivalent of a man-in-the-middle attack on the unauthenticated http:// data stream if it needs to do so for the purposes of load balancing, data auditing, etc. Using encryption as the new cleartext is only meant to deter passive attack scenarios.

Unauthenticated TLS for http:// provides relief for 5 or 6 of the 7 cited https:// objections and so on balance I believe it positively incentivizes TLS deployment.

The most convincing argument for pursuing an https:// only path suggests that deploying HTTP/2 over http:// simultaneously creates a dis-incentive for running the superior https://. After all, operators can get the performance and scalability benefits of HTTP/2 without having to transition their content to a new scheme. The fact that many of the usual https:// objections are becoming less relevant as time goes by (e.g. more third party services are offered in https://, and the fraction of SNI-incompatible clients is dropping) are supporting arguments for taking a watchful waiting approach.

Deciding what to do is difficult. It requires a lot of guessing about how many people will benefit (and to what extent) in each scenario. Reasonable people continue to vigorously disagree.

PART III: Alt-Svc Development Experience

tl;dr; Interested parties should read http://tools.ietf.org/html/draft-nottingham-httpbis-alt-svc-01 or its successors. The alternate service concept provides routing information about how a particular origin (e.g. http://host:port) can be accessed at different host and port and, in this case, with TLS. While this is initially done with a plaintext http/1 response header, it is not much different than the redirection done with a CNAME DNS record. Specifically the origin of the transaction remains the same but the actual connection details are re-routed.

In the initial implementation the Alt-Svc response triggers the asynchronous creation of another connection using the alternate route. Only when that channel has been created is the original channel abandoned. This is done to prevent the performance penalty that a redirect to an alternate host would have when compared to continuing to use the established channel. The new route is cached for a duration specified by the Alternate Service announcement and future connections use that route directly.

This approach, while easy to deploy, has a couple of problems. Chiefly, it relies on plaintext HTTP/1 for bootstrapping. A successful HTTP/2 deployment should aim for the long term retirement of HTTP/2 and building in this dependency runs contrary to that goal.

More importantly, it implies that data needs to first be transferred over plaintext HTTP/1 with an HTTP/2-TLS capable service complete the bootstrap. The client is forced to choose between taking a round trip delay by sending innocuous data hoping for an Alternate-Service reply (e.g. an OPTIONS request), or sending regular data in plain text to a server that could have done TLS if we had only known. A small amount of cookie data exposed in this way would be a particular tragedy.

The browser competitive landscape puts a premium on responsiveness and it is pragmatically impossible to choose the conservative solution. The result is Alternate Services implementations bootstrapped through

HTTP/1 will leak some information.

DNS records, such as SRV but possibly other types too, provide an alternative. The Alternate-Services information could be looked up in parallel with the origin host's A/AAAA record and then content providers offering an alternate service would not have to rely on an HTTP/1 plain text transaction to bootstrap its use.

Some are concerned a DNS approach will fail because of reliability and responsiveness problems. I believe that a DNS approach can be combined with the HTTP response approach and provide large benefits even if it is not perfect. For example, if the DNS approach had a 5% failure rate that would normally be completely unacceptable, but in the case DNS and HTTP bootstrapping were both deployed it would mean a 95% reduction in the amount of plaintext HTTP/1 bootstrap transactions which is a clear win.

Nick Hurley, another Mozilla hacker, and I deployed a test in the nightly builds of Firefox to test how effective DNS SRV resolution is on the open web. The test simply looked up two records in parallel from the same domain using some cache busting techniques along the way - one A record and one SRV record. We tracked the correlation between receiving the known correct answers and their arrival times.

For an alternate services use case the results of the SRV test were promising.

1. Only ~3% of successful A lookups were not paired with a successful SRV lookup
2. ~95% of the SRV responses were received within ~1 round trip time of the A reply being received.

The implication is that Alternate Services over DNS can eliminate about 92% of the HTTP/1 bootstrap requests that would be made without a combined DNS/HTTP-1 approach. This includes all SRV replies received within the time it would take to establish the TCP channel to the plaintext origin server.

I am interested in finding other STRINT participants interested in more fully vetting this approach.