Network Working Group
Internet-Draft
Intended status: Informational
Expires: December 10, 2012

R. Peon
Google, Inc

June 8, 2012

Explicit Proxies for HTTP/2.0
draft-rpeon-httpbis-exproxy-00

Abstract

This document describes a method for connecting to a proxy via a secure channel, allowing, disallowing, and detecting any transforms that the proxy may perform, and allowing the proxy to connect via secure channel to another site on the user's behalf.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 10, 2012.

Table of Contents

1. Overview 3
2. Definitions 3
3. What it looks like today 4
4. Where we would like to be 4
5. Problems with end-to-end privacy? 5
6. Proposed Solution: Use Explicit Proxies 6
7. Mixed trust mode 9
8. Rejected Alternatives 9
9. Impact on other areas of the protocol 10
10. Security Considerations 10
11. Requirements Notation 11
12. Acknowledgements 11
13. Normative References 11
Author's Address 12

1. Overview

It has been proposed that a secure channel be utilized for all HTTP/2.0 traffic coming from users using browsers. As a first order effect, this would obviously improve security, but it could have the second order effect of banishing all communication over any secure channel for lack of a mechanism to allow the owners of the network to apply desired policies to the messaging going on within it. Another second order effect could be user selection away from this protocol if it doesn't meet their performance expectations. This document proposes mechanisms intended to ameleoriate these issues. This document does not exhaustively cover configuring a proxy in a browser.

2. Definitions

user-agent: The program or device which a human interacts with directly and which typically initiates the transport layer connection or session

client: Synonym for user-agent

user: The human using the user-agent

server: The computer or device which typically accepts a connection and serves data

content-provider: The server to which the user-agent ultimately wishes to speak

configured-proxy: The entity to which the user-agent is explicitly configured to speak and which the user-agent expects will connect to the content-provider on the user-agent's behalf

trusted-proxy: A configured-proxy to which the user-agent has been configured to give decryption key material

caching-proxy: A configured-proxy to which the user-agent has NOT been conigured to give decryption key material

end-to-end: From the user-agent to the content-provider

point-to-point: Between any two pairs of adjacent entities. If the entities are user-agent and content-provider, this is also end-to-end, but for any other pair of entities, it is only point-to-point.

secure-hash: A cryptographic hash resistent to preimage and collision attacks, whether as part of a hash tree (e.g. Merkle tree), hash list, etc.

3. What it looks like today

Today:

Using TLS [RFC5246] over port 443 is often the exception rather than the rule.

Many users use radio-based devices which transmit data effectively in the clear, allowing their credentials and identity to be stolen.

Many networks seem to leave the traffic on port 443 untouched and unblocked, likely as a result of both the importance of the data and the relative rarity of communications using TLS.

Entities which need to inspect traffic on port 443 today are forced to either block port 443 or to deploy an intercepting proxy and install root certs on all devices which may use the network. In the latter case, the deployed proxy impersonates both the content-provider to the user-agent, and the user-agent to the content-provider. Though there is work to allow users to detect these situations [DANE], support is not widespread.

Users are likely to see both beneficial and detrimental behavior as a result of transparent proxies.

Many, if not most, mobile devices using cellular networks use proxies

Users and sites have only one mechanism for specifying point-to-point security policy for HTTP [RFC2616], which is the scheme of the URI identifying any particular resource.

4. Where we would like to be

In the future:

A user's communications should use a channel which is point-to-point secure by default for all communications.

Users should be able to opt-out of communicating sensitive information over a channel which is not end-to-end private.

Only proxies which are known to and configured by the user should be allowed to intercept communications between the user and the content-provider.

User-agents and servers should know when a proxy is interposed in the communications channel.

User-agents should be able to detect when content requested with an https scheme has been modified by a proxy.

Caching should still be a service which can be provided by entities other than the server or user-agent.

Caches, when allowed, should be exceedingly difficult to poison.

A set of signaling semantics should exist which allows the content-provider and the user to have the appropriate level of security or privacy signaled per resource.

5. Problems with end-to-end privacy?

Entities may wish to have a proxy interposed in communication for a number of reasons including (but not limited to):

- Looking for and blocking malware
- Filtering for content by size, type, or subject matter
- Provide caching services so as to improve the user experience

Unfortunately, if communication is to be done in majority or completely using end-to-end privacy, then none of the above use-cases are possible without installing root certs on users' devices. There are several possible consequences for having only end-to-end privacy:

It may become common for entites to install root certs on users' devices, and users may become accustomed to doing this, even on their personaly owned devices. This could cause a breach of all trust-chains, and could reduce aggregate security.

Entities which own networks may decide to block port 443, else face noncompliance of policy or law. This would obviously reduce aggregate security.

Clients at the end of long, thin pipes may decide that speed matters more than security, and disable use of the new, secure protocol. This would also obviously decrease aggregate security.

The assumptions and potential consequences in this section should be carefully deliberated.

6. Proposed Solution: Use Explicit Proxies

Since failing to allow for some kind of interception technique may reduce aggregate security, it is suggested we design for and allow explicit proxies which may examine contents as they pass by. This is not something to be done lightly, and so we must be careful to allow the user-agent to select an appropriate level of trust. We must also provide for a fallback in cases where the proxy or server does not adhere to this proposal so that in the worst case we get what we have today.

For the purpose of this document, it is assumed that the user locates a piece of paper upon a wall and reads it, typing these proxy settings into a configuration field for their user-agent. This is obviously not the only possible configuration mechanism, but it may, sadly, be the most secure. It is assumed that alternate distribution techniques may be discussed.

For HTTPS schemes, if a proxy is configured:

The user-agent MUST indicate to the content-provider that it is going through a proxy via the tunnel as the first bit of information presented.

If a link in an HTML [RFC2854] document includes a secure-hash of the data, e.g. <a href="https://foo.bar.com/baz" hash=0ACC827F19F36BA>, and the link was provided in a secure manner (e.g. over an end-to-end communications channel that includes integrity checks), then the user-agent MAY request the indicated resource directly from the explicitly configured proxy. The configured proxy MAY serve the data from its cache, but it MUST NOT modify the content in any way. The user-agent MUST reject any received data if is unable to verify its integrity or if the secure-hash does not match the hash over the provided data. The presence of the secure-hash is, itself,

signals that the content may be fetched in a non-private channel. RFC6249 [RFC6249] provides some descriptions of cryptographic-hash implementations.

The above need not be the only mechanism for providing signals that data can be served via a cache, but the data MUST always be verifiably unmodified. As an example, other mechanisms for providing secure-hashes could be:

The URI syntax [RFC3986] could be modified to include a hash of the contents.

A signed manifest may be sent which includes one or more secure-hashes of byte-ranges of the data, either fetched via a separate resource or sent in-band. The signature, would, of course, have to be validated before the secure-hashes were to be trusted, and before any requests were sent in a non end-to-end private channel. RFC6249 [RFC6249] provides some descriptions of such mechanisms. Bittorrent provides another example [BITTORRENT].

Within the TLS Tunnel, frames may be sent describing the secure-hash of a byte-range of the data, by request from the user-agent. Multiple secure-hashes may be returned if the byte-range mapping on the server differs from that requested by the client. This alternative is most interesting for its ability to describe dynamically generated data (such as live video) safely and in a way that can be consumed by the client in a closer-to-real-time manner, and for supporting range-requests. If implemented in a protocol similar to SPDY [SPDY], this could be accomplished with HEADER frames or equivalent.

If the user agent has no means of verifying the data is unmodified, the user-agent either MUST tunnel through the proxy by doing a CONNECT with a TLS tunnel, or the user-agent MUST reuse a previously-created tunnel offering the same security. If the proxy refuses this communication the user-agent MUST attempt a connection directly to the content-provider, avoiding the proxy.

We're proposing that user-agents allow for two levels of security for any proxy:

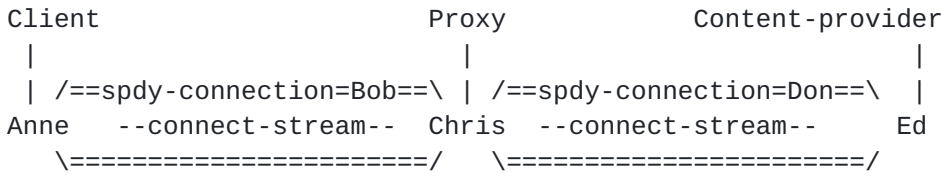
Trusted Proxy - all data sent and received is inspected by the proxy

Caching Proxy - only data which could be served from the cache is inspected by the proxy

It is also possible for a user-agent to use a proxy in both configured and trusted-proxy modes. This document does not include mechanisms for signaling this.

In the case where the proxy connects to the content-provider using a secure, multiplexed connection, the following diagram and description would apply:

Anne = the client or user-agent
Bob = point-to-point secure multiplexed connection from Anne<->Chris
Chris = the proxy
Don = point-to-point secure multiplexed connection from Chris<->Ed
Ed = the content-provider

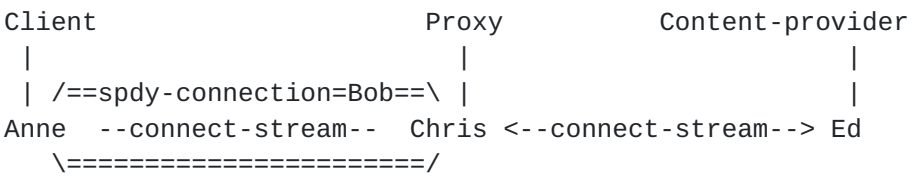


In the case where the user-agent has been configured with Chris as a trusted-proxy, either Anne's connect-stream MUST use either a null-cipher, or Anne MUST provide the decryption key material to Chris immediately after tunnel establishment, and before any data traverses the tunnel.
In the case where the user-agent has been configured with Chris as a caching proxy, Anne's connect-stream SHOULD NOT use the null cipher, and Anne SHOULD NOT provide the decryption key material to Chris.

In *all* cases, the user-agent must indicate that it is traversing a proxy within the connect-tunnel, and indicate whether the proxy is a trusted-proxy or caching-proxy.

In the case where the proxy cannot use a secure, multiplexed connection when connecting to the content-provider, the following diagram and description would apply. It is hoped that this case will never be selected as a spec-compliant implementation for forward proxies as it offers significantly less scalability than the option proposed above.

```
Anne = the client or user-agent
Bob = point-to-point secure multiplexed connection from Anne<->Chris
Chris = the proxy
Ed = the content-provider
```



In both trusted-proxy and caching-proxy cases, Anne must select a non-null cipher for use in the tunnel. In both the trusted-proxy and caching-proxy cases, the proxy simply forwards the tunnel's bytes to Ed and vice-versa. In the case where the user-agent has been configured with Chris as a trusted-proxy, Anne MUST provide the decryption key material to Chris immediately after tunnel establishment, and before any data is sent along the tunnel. In the case where the user-agent has been configured with Chris as a caching-proxy, Anne SHOULD NOT provide the decryption key material to Chris. If Anne selected a null-cipher in this case, Chris MUST reject the connection, forcing Anne to attempt to fallback on a separate connection.

In *all* cases, the user-agent must indicate that it is traversing a proxy within the connect-tunnel, and indicate whether the proxy is a trusted-proxy or caching-proxy.

7. Mixed trust mode

If a signaling mechanism is created which reliably indicates that some data be used in caching-proxy mode, where other data be retrieved in trusted-proxy mode, the user-agent MAY create two tunnels (one for each mode). A mechanism to signal this is not included in this document. A single tunnel MUST NOT be used, as it is impossible to both allow inspection and privacy simultaneously using mechanisms as proposed here.

8. Rejected Alternatives

Do nothing -
This assumes that port 443 won't be blocked, which is contrary to the assumptions in this document.

100% trust of the proxy at all times -
Hopefully it is obvious why this is bad.

Signed policy per response -
With this mechanism, the user-agent would expect a piece of signed metadata with each message from the content-provider. This has the disadvantage of being fairly chatty, but the worst part is the requirement to be shuttling around private-key material on the part of the content-provider. This material should be held closely; anything requiring its constant use will likely undermine its secrecy. It is also expensive to be signing things all the time...

Signed policy per domain or origin [RFC6454] -
With this mechanism, the user-agent would expect a piece of signed metadata with the first response to a request. The one-domain-one-policy seems too restrictive for today's site compositions, and doesn't match HTTP's current caching semantics.

9. Impact on other areas of the protocol

If we assume that we'll see widespread 'Caching Proxy' deployment, and we assume that the proxies will multiplex multiple incoming streams over fewer outgoing connections, then we may lose some of the current benefits of header compression, and must worry about endpoint per-connection stream limits.

If the mechanisms in this document are widely used, mechanisms which reduce the number of RTTs for the TLS handshake phase(s) including anything which allows for fast, efficient, safe resumption would be of high importance.

10. Security Considerations

Everything in this document should be carefully scrutinized as everything in this document impacts security.

A particular worry is the 'Trusted Proxy' mode. Does its presence undermine end-to-end security so much that the benefit of the proposal is moot? In particular, would users as a whole or in a significant part of the population be inclined to ignore (hopefully

dire) warnings about using 'Trusted Proxy' mode when they didn't own the proxy?

11. Requirements Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

12. Acknowledgements

[RFC6454];

13. Normative References

[BITTORRENT] Cohen, B., "The BitTorrent Protocol Specification", BITTORRENT 11031, February 2008, <http://www.bittorrent.org/beps/bep_0003.html>.

[DANE] Hoffman, P. and J. Schlyter, "DANE", April 2012, <http://tools.ietf.org/html/draft-ietf-dane-protocol.txt>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2854] Connolly, D. and L. Masinter, "The 'text/html' Media Type", RFC 2854, June 2000.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6249] Bryan, A., McNab, N., Tsujikawa, T., Poeml, P., and H. Nordstrom, "Metalink/HTTP: Mirrors and Hashes", RFC 6249, June 2011.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454,

December 2011.

[SPDY] Belshe, M. and R. Peon, "SPDY PROTOCOL", <http://tools.ietf.org/html/draft-mbelshe-httpbis-spdy>.

Author's Address

Roberto Peon
Google, Inc

Email: fenix@google.com