httpbis
Internet-Draft
Intended status: Best Current Practice
Expires: November 23, 2018

J. Brzozowski
Comcast Cable
K. Beevers
NS1
J. Cariello
Google
J. Colton
Squarespace
L. Jacob
Bloomberg
J. Leddy
Comcast Cable
J. Shaul
Akamai
L. Steinberg
CTM Insights
May 22, 2018

Trusted Traffic
draft-jjmb-httpbis-trusted-traffic-00

Abstract

Current methods for managing traffic through content inspection tend to process all sessions similarly. Internet traffic examples like DDoS mitigation require all data to pass through one of a limited number of scrubbing centers, which create both natural choke points and the potential for widespread collateral damage should a center become overloaded. Similar issues exist with email SPAM and malware filtering, traffic shaping, etc. We propose a method to utilize existing HTTP and HTTPS protocols that enables destinations to temporarily confer trust on sources, and for trusted traffic to be routed and processed differently from untrusted traffic.

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 November 23, 2018.

Table of Contents

1. Introduction

In the wake of several high profile DDoS attacks, the authors

convened a series of meetings to investigate alternatives to routing all traffic through centralized "scrubbing centers". It was believed that the widespread use of consumer appliances was creating a rapidly expanding edge that is increasingly being compromised. As the edge could potentially grow faster than centralized services could

increase capacity, we envisioned a structural issue in which mitigation services could become overwhelmed. The impact would be magnified through collateral damage, affecting all users of the mitigation service vs just those being attacked. We began looking for ways to allow certain traffic, once trusted by a destination, to bypass scrubbing centers when headed to that destination. Only traffic from a source untrusted by the destination required additional scrutiny. This would both reduce load on the scrubbing centers and allow trusted traffic to flow unimpeded even if a scrubbing center was overwhelmed.

As we investigated solutions, it became apparent that the underlying issue was more generalized than we originally understood. What was lacking was a pairwise trust model that could affect how traffic is treated. This would permit destinations to designate that they trust a source, and the network edge would apply different treatment (service transformations) when and only when a source communicates with a destination that trusts it. Example applications of this approach include an express lane that lets trusted traffic bypass DDoS scrubbing centers, the ability to apply deep malware scanning or pfishing filtering to untrusted email senders or content providers, and traffic shaping like rate limits for untrusted DNS queries. Distributing the validation of trusted traffic to a network edge creates resiliency by eliminating the use of centralized aggregation points, which become targets, for trusted traffic.

A version of this approach was explored for non-HTTP/S traffic. in it, IPv6 Segment Routing Headers were used to transmit Trust and Assertion Tokens, as well as to direct messages containing Assertion Tokens through an edge Validator. Implementation and testing of this approach was deferred to focus on the HTTP/S use case.

Also discussed was that this mechanism is consistent with a delegated or federated trust model. Destinations that lack the ability to directly determine whether a source should be trusted can leverage the trust determined by 3rd parties. This is accomplished by having

the 3rd party, once trust is established, reference or load an image belonging to the destination's domain. The destination can then, if properly referred by the 3rd party, set a Trust Token cookie for the destination's domain.

Several key principles were established to guide our efforts:

1. Trust is pairwise between a destination and a source

2. A destination decides to trust to a source based solely on its own decision making process

3. Trust is temporal

4. Untrusted traffic flows unimpeded

5. No new protocols would be defined

6. We would leverage a distributed set of services to validate that traffic is trusted in order to provide resiliency and scale

7. Validation of trust would be stateless, with no decryption keys or tables

2. Assumptions

The motivation for this work falls into the following categories:

- DNSSEC signed zones
- DNS references (where applicable) for use to bootstrap control protocol (see below)
- Secure transport for beaconing or control protocol (QUIC, HTTPS, etc.)
- Control Protocol Definition (this document)

3. Approach

Our concept of a distributed processing layer at the edge was further

influenced by feedback from the service provider participants. Based on their input, we decided to separate the implementation of a processing layer from the network itself, thereby avoiding potential conflicts with things like Net Neutrality principles. Service providers could comfortably route to or via destinations requested by a client, without the network layer making value decisions around message treatment

The approach selected borrowed heavily from the concept of web cookies. In essence, a layer 3 cookie became a Trust Token ("TT") that was offered by a destination to a source. Each destination is free to define its own mechanism for deciding whom to trust and for how long, as over-trusting (or under) only affects that destination. The use of a token allowed us to work with devices behind a Network Address Translator ("NAT"), which was an important consideration for consumer households that had a mix of potentially trusted devices (e.g. a computer) and untrusted devices (e.g. a webcam).

Trust Tokens contain several fields: the public name or IP address of the destination issuing it, the public name or IP address of a Validator Service ("Validator"), the public IP address of the device being trusted, an expiration time beyond which the token is not valid, and a cryptographic signature to guard against fraudulently generated or manipulated tokens. A protocol to issue and refresh expiring trust tokens is proposed in this document. Note that fields which are readily available are not required to be duplicated in the TT.

Traffic from a source that has been trusted by a destination has to assert that it is currently trusted, and route its traffic through a nearby Validator. The assertion is accomplished by inserting an Assertion Token ("AT"), a subset of the full TT containing the expiration time and the cryptographic signature. The validator service is then responsible for inspecting the AT and testing the cryptographic signature against a combination of the destination IP address, the public IP address of the source, and the expiration time. If successful, and the expiration time has not yet been reached, the message is forwarded. If not, the message is discarded.

A successfully validated message SHOULD be forwarded directly to the

destination through the use of a tunnel such as a GRE [RFC2784] between the Validator and destination. The Validator would see the destination as the next hop and route appropriately. The Validator MAY learn routes through the tunnel but MUST NOT announce those routes to ensure that only traffic specifically sent to the Validator is tested.

It is expected that the service provided by Validators is computationally much more efficient than the service transformation being applied to untrusted traffic (e.g. DDoS mitigation). This efficiency allows Validators to be widely distributed at the edge of a large network. All Validators MAY be assigned a common Anycast address [RFC1546] to simplify the effort of generating TTs, and to assist in creating resiliency should one or more Validators become unavailable. Alternatively, the server issuing TTs can check the health of each validator to which it has an active GRE, and use a formula such as geolocating the client's IP to determine the nearest healthy Validator. As Validators become unavailable, the traffic will naturally migrate away as new TTs are issued and existing ones are refreshed.

4. HTTP and HTTPS

HTTP and especially HTTPS created some practical implementation issues. The first was that the browser environment doesn't generally have access to set socket options for an HTTP GET or POST outside of

web sockets. Implementing this solution would require modifying the browser source code. The use of a cookie construct helped as we were able to leverage browser cookies to hold the TT. A browser plugin detects a message to a destination for which there is a matching, unexpired cookie and rewrites the hostname or SNI field to prepend the AT. A wildcard version of the destination address with any prepend MUST resolve to the Validator's IP address. The Validator acts as a transparent proxy, stripping the prepended data before forwarding the message.

4.1. Trust Tokens as Cookies

In the case of HTTP/HTTPS, the TT described above is sent by the destination server to a trusted client as a web cookie [RFC6265] with the following fields:

Set-Cookie: type= TrustAssertionToken; Version= 1; ProxyValidator= [proxy.example.com]; Domain= [example.com]; ClientSource=[clientIpAddr]; Max-Age= [ttl]; ExpireTime= [timeoutAbsolute]; Signature= [signedHash]

The value of ttl is set by the server to indicate when the trust relationship will end if not renewed. The value timeoutAbsolute SHOULD be current time in seconds since epoch plus ttl plus a server defined grace period in case the cookie expires while in transmission or processing. Note that the timeoutAbsolute value is presented back to the server to determine if a renewal is warranted. The value of signedHash is generated at the destination server by:

1. appending the numeric timeoutAbsolute value to the end of the clientIpAddr value

2. creating a signature using a private key with the EdDSA algorithm and Ed25519 per [RFC8032].

3. converting the signature to an ascii string

4. applying a base64 hash algorithm to reduce the signature to 63 bytes of ascii

In the web implementation, this is accomplished through the following example:

```
const crypto = require('crypto');

const ed25519 = require('ed25519');

class SignatureService extends CommonService {
```



```
generateSignature(timeoutAbsolute, clientIpAddress) {

const seed =crypto.randomBytes(32);

const pair = ed25519.MakeKeypair(seed);

const message = clientIpAddress + timeoutAbsolute;
```

```
const signature = ed25519.Sign(new Buffer(message, 'utf8'),
pair.privateKey);

return crypto.createHash('sha256').update(signature.toString('ascii')
).digest('base64');

}

}
```

The signedHash value MUST be encoded to be compatible with the restrictions of the Hostname [RFC1123]. As the ED25519 signed message is `base64` encoded. The character set potentially produced is [A-Za-z0-9+/]

Since Hostnames are case insensitive and the characters `+` and `/` are not valid, each capital letter MUST BE prepended with a `-` and transformed to lower case; each `+` MUST BE transformed to `--` and each `/` MUST BE transformed into `_`

Valid Hostnames MUST be broken into 63 byte fields, separated by `.` characters. Furthermore, Hostname's starting with `-` or hostnames with `-` before and after `.` are considered invalid. A prefix `a` and a suffix `a` MUST be added to the encoded signature, and if a 63 byte hostname field has a `-` that follows a `.` the `.` is pulled to the left and is set between the first pair of small characters.

e.g. for hostname "asfdsdfasdf--asd-.s"

the '.' would be pulled to the left to in between s and d, so asfdsdfasdf--asd-.s becomes asfdsdfasdf--as.d-s

4.2. Assertion Tokens

Assertion Tokens are a subset of the full browser cookie. The use of HTTPS results in cookies that are encrypted during transmission, which hides them from a stateless Validator. Instead, an Assertion Token MUST contain the clientIpAddr as an ascii representation of the hex values , followed by a ".", followed by the ascii representation

of the value in timeoutAbsolute, followed by a ".", followed by the

signedHash value.

For clarity, clientIpAddr MAY NOT be a dotted decimal string and timeoutAbsolute MUST be the number of seconds since epoch, in GMT, after which the trust relationship will expire, with the digits expressed as an ASCII string plus a grace period.

4.3. Assertion Token use within a Browser

Within the browser environment, the a participating client MUST run code that prepends the AT, followed by a ".", to the ProxyValidator value and places this in the hostname or SNI field. Note that the ProxyValidator value is a superset of the Domain. This rewrite will send the session via the transparent proxy server at the ProxyValidator address.

4.4. Validator Function

When an HTTP/HTTPS session is initiated to the Validator, it strips the prepended AT and proxy from the hostname/SNI. It then MUST use a public key, which MAY be locally stored, associated with the remaining hostname value to decrypt and test the validity of the signedHash. Should no local key exist, or the hash validation fail, the message MUST be dropped. The Validator MUST test the timeoutAbsolute against the current time, and MUST drop the message should the current time exceed the value specified in the AT.

The validator MAY further check that the tcp connection has a source IP address that matches the clientIPAddress and drop the message should the test fail. The validator MAY choose to ignore this test to accommodate longer lived TTs where the public IP address might change (e.g. multiple public interfaces or dual IP stack clients),

4.5. Implementation Considerations

The role of the Validator is to both check that browser sessions are unexpired and legitimate, and to act as a transparent proxy server. If all tests are successful at the Validator, messages are forwarded to the destination server as specified in the hostname/SNI field without the prepended AT or proxy values.

For performance reasons, the Validator MAY choose to cache signedHash values once they have been tested, until the expiration of the ATs.

Each participating destination SHOULD have a GRE tunnel to each validator, and SHOULD announce routes to the Destination Server via that tunnel. The Validator MUST NOT announce these learned routes

When the Validator decides to forward sessions to the destination, the closest next hop from a routing perspective is the GRE, essentially bypassing any traffic filtering and transformations that untrusted traffic is routed through include diagram

Timeout values are in both the cookie and the AT, as the timeout needs to be visible to both the destination server and the Validator . The destination server MAY use the timeoutAbsolute to refresh the TT as it nears expiration

5. IANA Considerations

No IANA considerations are defined at this time.

6. Internationalization Considerations

No IANA considerations are defined at this time.

7. Security Considerations

No Additional Security Considerations are made in this document.

8. Acknowledgements

The authors would like to thank the following, in alphabetical order, for their contributions:

John Curtis of Curtis Digital and Thomas M Jacob of Comcast contributed to a POC implementation and testing of this system. Their efforts and design feedback are greatly appreciated.

9. References

9.1. Normative References

[RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <https://www.rfc-editor.org/info/rfc1123>.

[RFC1546] Partridge, C., Mendez, T., and W. Milliken, "Host Anycasting Service", RFC 1546, DOI 10.17487/RFC1546, November 1993, <https://www.rfc-editor.org/info/rfc1546>.

[RFC2784] Farinacci, D., Li, T., Hanks, S., Meyer, D., and P. Traina, "Generic Routing Encapsulation (GRE)", RFC 2784,

DOI 10.17487/RFC2784, March 2000,
<https://www.rfc-editor.org/info/rfc2784>.

[RFC3315] Droms, R., Ed., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, DOI 10.17487/RFC3315, July 2003, <https://www.rfc-editor.org/info/rfc3315>.

[RFC3514] Bellovin, S., "The Security Flag in the IPv4 Header", RFC 3514, DOI 10.17487/RFC3514, April 2003, <https://www.rfc-editor.org/info/rfc3514>.

[RFC4032] Camarillo, G. and P. Kyzivat, "Update to the Session Initiation Protocol (SIP) Preconditions Framework", RFC 4032, DOI 10.17487/RFC4032, March 2005, <https://www.rfc-editor.org/info/rfc4032>.

[RFC4302] Kent, S., "IP Authentication Header", RFC 4302, DOI 10.17487/RFC4302, December 2005, <https://www.rfc-editor.org/info/rfc4302>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

9.2. Informative References

[I-D.ietf-v6ops-v4v6tran-framework]
Carpenter, B., Jiang, S., and V. Kuarsingh, "Framework for IP Version Transition Scenarios", draft-ietf-v6ops-v4v6tran-framework-02 (work in progress), July 2011.

[RFC6343] Carpenter, B., "Advisory Guidelines for 6to4 Deployment", RFC 6343, DOI 10.17487/RFC6343, August 2011, <https://www.rfc-editor.org/info/rfc6343>.

Authors' Addresses

John Jason Brzozowski
Comcast Cable
1701 John F. Kennedy Blvd.
Philadelphia, PA
USA

Email: john_brzozowski@cable.comcast.com

Kris Beevers
NS1
40 Exchange Place, Suite 410
New York, NY
USA

Email: kbeevers@ns1.com

James Cariello
Google
111 8th Ave
New York, NY
USA

John Colton
Squarespace
8 Clarkson St
New York, NY
USA

Lutz Jacob
Bloomberg
731 Lexington Avenue
New York, NY
USA

Email: ljacob@bloomberg.net

John Leddy
Comcast Cable
1701 John F. Kennedy Blvd.
Philadelphia, PA
USA

Email: john_leddy@cable.comcast.com

Josh Shaul
Akamai
150 Broadway
Cambridge, MA
USA

Email: jshaul@akamai.com

Lou Steinberg
CTM Insights
1988 Beekman Ct
Yorktown Heights, NY
USA

Email: lou@ctminsights.com