Network Working Group
Internet-Draft
Expires: January 4, 2013

J. Silvera
M. Cox
I. Pashov
O. Mazahir
G. Montenegro
Microsoft

July 3, 2012

Multilegged Authentication for HTTP Multiplexing
draft-montenegro-httpbis-multilegged-auth-01

Abstract

In line with the HTTP compatibility goal for HTTP 2.0, HTTP 2.0 must also be compatible with currently deployed authentication schemes. This draft addresses this goal in the presence of multiplexing (expected to be part of HTTP 2.0), while addressing some of the issues currently encountered when performing multilegged authentication.

Status of this Memo

Copyright Notice

Table of Contents

1. Overview

This document defines multilegged authentication for HTTP multiplexing.

Without reliable support for Kerberos and other multilegged auth schemes, the reach of HTTP2.0 will be greatly diminished in corporations that rely on these authentication schemes to protect their intranet resources.

HTTP's architecture requires that messages be stateless; that is, that they are able to be interpreted in isolation, without relying upon state from "lower" layers, such as association between requests using the same TCP connection. To enable better support of multilegged authentication, we propose that the shared state for which some authentication schemes rely upon the TCP connection (thereby making messages stateful) be moved into the HTTP/2.0 session layer.

The following table summarizes widely deployed authentication schemes, their authentication types and the authentication level they provide:

Table 1:

Scheme	Type of Authentication	Authentication Level
Basic	Per Request	Request
Digest	Per Request	Request
NTLM	Multilegged	Connection
Kerberos	Multilegged	Connection or Request
Negotiate	Multilegged	Connection or Request

RFC 2616 defines HTTP as a stateless protocol and dictates that authentication schemes MUST be stateless. However, multilegged authentication support for multiplexing requires state to associate separate request/response pairs that are part of the same multilegged authentication process.

1.1. Requirements Language

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].

2. Multilegged Authentication in HTTP 1.X

As implied by its name, multilegged authentication requires multiple roundtrips to establish an authenticated communication channel between client and server. If the resource requested by a client requires authentication, the server initiates the authentication process as follows:

Figure 1: Multilegged authentication example:

```
  Client                                      Server
    |                                           |
    | -------- (0) HTTP GET Request ---->       |
    |                                           |
    | <------- (1) HTTP 401 -------------       |
    |                                           |
    | -------- (2) HTTP Get Request ---->       |
    |              w Auth header                |
    |                                           |
    | <------- (3) HTTP 401 -------------       |
    |                                           |
    | -------- (4) HTTP Get Request ---->       |
    |              w Auth header                |
    |                                           |
    | <------- (5) HTTP 200 OK-----------       |
    |                                           |
    |                                           |
    v                                           v
```

1. Server sends HTTP 401 response because the resource requested requires authentication.

2. Client re-issues the HTTP GET request for the resource, including authentication headers.

3. Server responds with an HTTP 401 and authentication headers requesting additional information.

4. Client re-issues the HTTP GET Request for the resource, including authentication headers with the additional information required to complete authentication.

5. If authentication succeeds, the server responds with an HTTP 200 OK message including the requested resource.

Multilegged authentication requires state to be communicated between multiple streams. Network flows 3 and 4 in Figure 1 need to share

state in order for authentication to succeed. Some multilegged authentication schemes (i.e. Kerberos and Negotiate) can authenticate either a connection or individual requests, which has historically caused a lot if issues with multilegged authentication in HTTP 1.1.

3. Multilegged Authentication in the Presence of HTTP 2.0 Multiplexing

Figure 2 below provides a detailed breakdown of proposed network flows to implement multilegged authentication for HTTP 2.0 multiplexing:

Figure 2 - Proposed multilegged authentication:

```
                                             Multiplexing
    Client                              HTTP Server
       |                                       |
       | --------   HTTP GET Request ------>   |
       |                                       |
       | <------- (0) HTTP 401 -------------   |
       |                                       |
       | -------- (1) HTTP Get Request ---->   |
       |              w Auth header            |[Auth-ID header
       |                                       | generated (1.5)]
       | <------- (2) HTTP 401 -------------   |
       |              w Auth-ID header         |
       |                                       |
       | -------- (3) HTTP Get Request ---->   |[Persisted-auth header
       |              w Auth header and        | generated (3.5)]
       |              Auth-ID header           |
       |                                       |
       | <------- (4) HTTP 200 OK-----------   |
       |              w optional               |
       |              PersistedAuth header     |
       |                                       |
       |                                       |
       v                                       v
```

1. Server sends HTTP 401 response because the resource requested requires authentication.

2. Client re-issues the HTTP GET request for the resource, including authentication headers.

   Multilegged authentication schemes could authenticate individual requests or the HTTP 2.0 session. Clients SHOULD NOT authenticate individual streams belonging to an authenticated HTTP 2.0 session. The following describes client behavior when attempting to authenticate streams, for which it does not know if the negotiation will result in request based or HTTP 2.0 session based authentication:

If an HTTP 2.0 session has a stream in process of authenticating using a multilegged authentication scheme, the client SHOULD queue all subsequent requests (regardless of whether they require authentication) on the session until the multilegged authentication completes.

If a server receives multiple authenticated requests from the same client, it SHOULD NOT block responses. It is the client's responsibility to queue requests when a multilegged stream authentication process has been initiated in the session. If the client does not queue the requests, then it might unnecessarily authenticate streams in a session that has already been authenticated.

If connection-based multilegged authentication succeeds on a previously authenticated session, the server SHOULD discard the previous authentication context and authenticate the session with the newly negotiated authentication context.

3. Server responds with an HTTP 401 and authentication headers requesting additional information.

   A session's lifetime is not tied to the duration of a request/ response pair. If the authentication scheme used to validate the client's identity is a multilegged scheme, servers MUST generate a new "Auth-ID" (1.5 in Figure 2) header. The Auth-ID value is an opaque blob that SHOULD NOT be interpreted. It MUST be sent in its complete form to continue an authentication process. The server uses this to look up the correct security context to process this authentication request.

   The HTTP 401 response from the server MUST contain the "Auth-ID" header to enable sharing of authentication context across streams, as required for multilegged authentication.

4. Client re-issues the HTTP GET request for the resource and MUST include the required authentication headers and the "Auth-ID" header, to inform the server that the request is part of a previously initiated multilegged authentication process.

5. Authentication succeeds and the server returns the requested resource, along with level of multilegged authentication scheme:

   Some multilegged authentication schemes can result in per-request or per-connection (i.e., Kerberos or Negotiate) authentication. When a session is authenticated, servers MUST generate a Persistent-auth header 3.5 in Figure 2) and send it along with the HTTP 200 OK response. The client MUST use the presence or

absence) of the Persistent-auth header to determine what action to take with previously queued requests due to multilegged authentication being in progress:

1. If the session was authenticated, as indicated by the presence of the Persistent-auth header, the client does not need to authenticate new streams it creates to service the queued requests on the authenticated session.

   Clients SHOULD assume that successful authentication with schemes that only support connection-based authentication (i.e. NTLM) always result in an authenticated session, even if the Persisted-auth header is not present.

2. If the session was not authenticated, as indicated by the absence of the Persistent-auth header, the client SHOULD remember the negotiated authentication scheme used for authentication. The client SHOULD NOT block streams on the session when processing requests using the multilegged authentication scheme that previously resulted in per-request authentication.

A server MAY generate and add Auth-ID header as soon as it knows that the requested authentication scheme is multilegged. The client MUST add the Auth-ID header to all subsequent requests required to complete the authentication process.

A server MAY generate a Persistent-auth request as soon as it knows that the requested authentication scheme will authenticate the session. The client CANNOT make any assumptions by the absence of the Persistent-auth header, until the authentication process is complete and it receives the final server response containing the requested resource.

3.1. Auth-ID Header and Security Context Lifetime

Servers create and store security context information when they create the Auth-ID header. An Auth-ID header CANNOT be reused across sessions.

The Auth-ID mapping is destroyed when the authentication process completes. Completion of the authentication process can be a successful authentication or failure to authenticate. TBD: Should Auth-IDs be random numbers? The security vs look-up perf implications should be weighed.

Servers SHOULD limit the number of incomplete security contexts per session, to protect against misbehaving clients that cause the server to create multiple authentication contexts but never complete the authentication process. Servers SHOULD define a maximum number of incomplete security contexts and ignore SYN streams from misbehaving clients.

Per-session security contexts are transient and servers SHOULD discard them when request processing completes. There SHOULD only be one complete security context open per session.

4. Stateful Authentication to Proxies

HTTP proxies MUST add a new Remote-http-version header to inform the client of the HTTP version of the remote host. HTTP 2.0 clients can make authentication decisions based on the HTTP version of the target server. Proxy and gateway applications should take the consideration outlined by RFC 2616 when forwarding messages between client and servers with different protocol version capabilities.

4.1. HTTP 2.0 Client Authenticating to an HTTP 2.0 Server via Proxy

Authentication from an HTTP 2.0 (or greater) client to an HTTP 2.0 (or greater) server will work without additional changes, other than those described in the Multilegged Authentication for HTTP multiplexing proposal section of this document. Proxies MUST bind the client-to-proxy and proxy-to-server connections.

4.2. HTTP 2.0 Client Authenticating to an HTTP 1.1 Server via Proxy

HTTP 2.0 (or greater) clients that establish an authenticated connection to an HTTP 1.1 server (via a proxy) SHOULD downgrade HTTP version to HTTP 1.1, to avoid serialization.

HTTP 2.0 clients can multiplex streams within the authenticated HTTP 2.0 client-proxy session, but the proxy MUST serialize requests through the authenticated HTTP 1.1 proxy-server connection.

HTTP 2.0 clients could decide to downgrade all requests requiring authentication (via a proxy) to HTTP 1.1 servers or only downgrade authenticated sessions, as indicated by the presence of the Persisted-auth header.

4.3. HTTP 1.1 Client Connecting to an HTTP 2.0 Server via Proxy

No changes required to support this scenario as HTTP 1.1 clients ignore HTTP 2.0 headers.

5. Authentication and multi-host sessions

The proposed authentication mechanism works only if there is a limit of one unique host per HTTP 2.0 session.

6. Security Considerations

Implementers should be aware of the security considerations defined by the individual authentication schemes supported. The following are some general security considerations that are independent of the proposed authentication mechanism.

The proposed authentication mechanism is only used to provide authentication of a user to a server. It provides no facilities for protecting the HTTP headers or data including the Authorization and WWW-Authenticate headers that are used to implement this mechanism.

Alternate mechanisms such as TLS can be used to provide confidentiality. Hashes of the TLS certificates can be used as channel bindings to secure the channel. In this case clients would need to enforce that the channel binding information is valid.

If an HTTP proxy is used between the client and server, it MUST take care to not share authenticated connections between different authenticated clients to the same server. If this is not honored, then the server can easily lose track of security context associations.

A proxy that correctly honors client to server authentication integrity will supply the "Proxy-support: Session-Based-Authentication" HTTP header to the client in HTTP responses from the proxy.

7. Acknowledgements

Thanks to the following individuals who provided helpful feedback and contributed to discussions on this document: Paul Leach, Nathan Ide and Rob Trace.

Authors' Addresses

Jonathan Silvera
Microsoft

Email: JSilvera@microsoft.com

Matthew Cox
Microsoft

Email: MaCox@microsoft.com

Ivan Pashov
Microsoft

Email: IvanPash@microsoft.com

Osama Mazahir
Microsoft

Email: OsamaM@microsoft.com

Gabriel Montenegro
Microsoft

Email: Gabriel.Montenegro@microsoft.com