Hypertext Transport Protocol (HTTP) Session Continuation Protocol

draft-williams-websec-session-continue-proto-00trust200902stdHTTP Session ProblemSecurity AreaInternet-Drafttocindent="no"comments="yes"inline="yes"

One of the most often talked about problems in web security is “cookies”. Web cookies are a method of associating requests with “sessions” that may have been authenticated somehow. Cookies are a form of bearer token that leave much to be desired. This document proposes a session “continuation” protocol for HyperText Transport Protocol (HTTP).

1. Introduction

The motivation for this protocol is described in I-D.williams-websec-session-continue-prob.

We define a protocol for cryptographic “session continuation” for HyperText Transport Protocol (HTTP) RFC2616. Session continuation is the act of binding an HTTP request to a “session”. A “session” consists of all the HTTP requests by a given user (possibly an authenticated user, or possibly an anonymous user). This protocol is a cryptographic protocol that aims to meet all the requirements given in I-D.williams-websec-session-continue-prob.

The protocol consists of:

- a request header carrying a keyed Message Authentication Code (MAC) that proves possession of a shared session key (shared between the user and the server);
- a response header advertising a default session scope to clients;
- a session identification in the form of a URI;
- an optional facility for server-side statelessness by storing state on the client-side, encrypted in a secret key known to the server;
- a request header for requesting the establishment of a session;
- a response header for indicating the establishment of a session, and including a session URI and any optional state to be repeated by the client.

1.1. Conventions used in this document

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. Session Keying

There are two methods for keying an HTTP session:

- session keys are output by HTTP authentication;
- or session keys are asserted by the client or the server;

For the key assertion method TLS with confidentiality protection is clearly REQUIRED for security. We've considered the possibility of using Diffie-Hellman key agreement or RSA key transport, but as that would duplicate functionality that is in TLS we consider that out of scope for the time being.

In either case a single session key is produced, and that is the only key utilized. Having a single key helps reduce session state size and protocol complexity, but we must (and do) distinguish key usage by prefixing a purpose indicator to the MAC input.

When keys are output by HTTP authentication there may be a key length mismatch. In this case a Key Derivation Function (KDF) RFC5869 is applied to generate the session key.

We could also use the TLS extractor to generate keys, but that would be an unnecessary complication and would provide very little additional value. Channel binding is achieved per-RFCs 5056 and 5929.Should we have distinct session keys for request and response MACs? Probably, but it increases state size. Better to add a direction indicator to the MAC plaintext.

2.1. Mixing HTTP and HTTPS

We expect that many sites will continue to mix HTTP and HTTPS for various reasons. To make this possible the MAC input will include a marker indicating HTTP or HTTPS.

2.2. Authenticated Session Keying

When an HTTP client uses HTTP authentication, and the authentication mechanism used can establish a session key, then the client SHOULD request session initiation using a shared session key output by the HTTP authentication mechanism. The client MUST send the session initiation header concurrently with the last HTTP authentication message.

2.2.1. HTTP/Negotiate Session Keying

Write text explaining how to use the GSS PRF to exchange keys when using HTTP/Negotiate

2.2.2. Digest

Digest could output a session key. Do we want to bother? (Basic certainly can't, or we'd not want it to anyways.)

2.3. Unauthenticated Session Keying

Sessions for unauthenticated users may appear to make little sense at first. This is useful, for example, and just as web cookies are, for tracking "shopping carts" when a user is window shopping, so to speak.

For unauthenticated session initiation the client merely requests the creation of a session with an asserted session key, for lack of a better choice.

3. Session Initiation

Sessions are always initiated by the client by including a Session-Init header in the client's request carrying the client's proposal for a session.

Servers that support sessions will respond by creating a session and returning a session ID URI.

The Session-Init proposal header's value consists of a comma-separated list of proposal parameters:

session-param = token "=" (token | quoted-string) Session-Init = 1#session-param

Session-Init request header

The following session parameters are defined:

Key-Method

The type of keying: "auth" (key will be output by HTTP authentication), "c-assert" (key is asserted in this Session-Init) or "s-assert" (the server is expected to assert a key). In the "auth" case the Session-Init MUST also carry a nonce and a MAC. This session-param MUST be present.

Key

The key that the client asserts, if the client asserts a key (Key-Method=c-assert). This may also be included when Key-Method is "auth" in case the server's implementation of HTTP authentication does not output a key, but only when using HTTPS.

Key-Length

The length of the master session key, as a count of key bits, in base-10. The value MUST NOT be less than 96 or larger than 256. If absent the key length SHALL be 128 bits.

MAC-Algs

The MAC algorithms supported by the client. This document defines only "HMAC-SHA-1" (HMAC with SHA-1), "HMAC-SHA-1-96" (HMAC with SHA-1 and truncation to 96 bits), "HMAC-SHA256" (HMAC with SHA256), and "HMAC-SHA256-128" (HMAC with SHA256 and truncation to 128 bits). All of these use HMAC RFC2104. Clients and servers MUST support HMAC-SHA-1-96 and HMAC-SHA256-128. If absent the default value is "HMAC-SHA256-128".

KDF-Algs

A list of KDF algorithms. This is needed only when Key-Method is "auth". The following are specified here: "HKDF-SHA-1" (HKDF RFC5869 with SHA-1) and "HKDF-SHA256" (HKDF with SHA256). Clients and servers MUST support HKDF-SHA256. If absent and Key-Method is "auth" then the default value is HKDF-SHA256.

Channel-Binding-Types

A comma-separated list of channel binding RFC5056 types. Clients and servers MUST support 'tls-server-end-point' RFC5929 when using HTTPS. (Note the need to use quoted-string when the list has more than one item.) If absent the default is 'tls-server-end-point'.

Nonce

A 128-bit nonce, base64-encoded. This session-param MUST be present.

Previous-Session-URI

The URI of a previous session. See Section 5.

Previous-Session-State

The session state for the previous session, if any. See Section 5.

Unprotected-Allowed

If present the value MUST be "true", and indicates that HTTP and HTTPS are both allowed for this session. Otherwise only HTTPS is allowed for this session.

The server responds with a Session-Assign header:

session-params = 1#session-param Session-Init-Value = <the value of the Session-Init header> MAC-input = Session-Init-Value "," session-params MAC = <base64-encoding of MAC taken over the MAC-input> Session-Assign = session-params ["," MAC]

Session-Assign response header

The MAC is OPTIONAL when using HTTPS, REQUIRED otherwise.

The session-params for Session-Assign are:

Key-Method

If the client requested "auth" as the key method but the server's implementation of HTTP authentication could not output a key then this session-param MUST be present with a value of "c-assert" (if the client included a Key in its Session-Init) or "s-assert" (otherwise).

Key

The server-asserted key, if the client requested a server-asserted key.

MAC-Alg

The name of the MAC algorithm selected by the server from the client's proposal (REQUIRED).

KDF-Alg

The selected KDF algorithm (when the client's selected Key-Method is "auth").

URI

The URI of the session (REQUIRED).

State

Server-side state to be stored on the client (OPTIONAL). Note that servers MAY choose to store server-side state in cookies instead.

Previous-Session

Indicates whether the previous session was recognized and accepted (“accepted”), rejected (“rejected”), or unknown (“unknown”). This session-param MUST be present when the client’s Session-Init had a Previous-Session-URI session-param.

Host-Scope

A DNS domainname (in A-label form) that the session can be used with. Multiple Scope parameters are allowed. If the domainname starts with a ’.’ then the session may be used with all server hosts whose domainnames are sub-domains of the given Host-Scope domainname. The server’s fully-qualified hostname is always part of the session’s host scope.

4. Session Scope: Sharing Sessions Across Servers

A service might be composed of multiple related servers, each with a different hostname. As a result the service may require a client to use the same session across the service’s component servers. We provide a mechanism by which the server may indicate a set of such servers to the client: the Host-Scope session-param in the server’s Session-Assign response header.

[XXX We need a way to constrain this for privacy protection reasons. It’s not yet clear how the client can judge which Host-Scope paramters to accept or ignore, only that must be allowed to do so.]

To facilitate interoperable session sharing across heterogeneous server implementations we define a session resource -named by its session URI- that can be obtained with a properly-authenticated GET by authorized entities. The session resource’s representation is a application/json document type, containing a JSON-serialized associative array with the following REQUIRED keys:

Master-Key

The session’s master key, base64-encoded.

MAC-Alg

The MAC algorithm for this session.

We probably want each server to see a different master key, in which case we probably want to use a KDF with the server’s hostname as part of the salt.We probably want to define some OPTIONAL keys for this object, such as “User”, “User-URI”, “HTTP-Auth-Scheme-Used”, “HTTP-Auth-Scheme-<param>”, and so on, as well as an application-specific namespace of keys (e.g., “App-<appname>” or “<URN>”).

5. Unauthenticated to Authenticated Session Upgrade

A client might first establish an unauthenticated session then authenticate the user later. When authentication is done the client might wish to preserve any state associated with the preceding

unauthenticated session. The client does this by sending a Session-Init at authentication time with a 'Previous-Session-URI' session-param and, if there was server-assigned session state, a 'Previous-Session-State' session-param.

6. Session Continuation

Once a session is established the client binds requests to sessions as described here.

There are two cases: HTTPS and HTTP. In both cases the client adds a header

For the HTTPS case the client adds a "Session" header to its requests with the following content: the session identifier assigned by the server, a nonce generated by the client, and a MAC of the nonce and the TLS channel bindings.

The value of the Session header consists of a base64-encoded 128-bit nonce and a MAC, using the session's MAC algorithm, of the nonce and the channel binding, each base64-encoded then concatenated in that order:

CB = <base64-encoding of the channel bindings> nonce = <base64-encoded 128-bit nonce> new-state = ... direction = "c2s" | "s2c" prot-state = "protected" | "unprotected" response-status = "" | "Invalid-MAC" | | "Session-expired" "Session-unknown" MAC-input = direction "," prot-state "," nonce "," CB "," status "," [new-state] MAC = <base64encoded MAC taken over MAC-input> Session = nonce "," response-status "," [new-state] "," MAC

Session header

Where the response must carry a Session header, the form of the value is the same as for requests.

The MAC is taken over a direction indicator, an indicator of whether TLS is used, the nonce, the channel bindings, and so on, as shown in Figure 3. Only the server may assert new session state, and only the server indicates a response-status other than "" (empty string).

6.1. Session Validation and Error Handling

The receiver computes the same MAC using the sender's nonce (and new-state, if present, when the receiver is the client) and compares the resulting MAC to the MAC from the Session header.

If MAC validation of a request fails then the server MUST respond with a 403 status code with a non-empty response-status int he Session-header. Error responses MUST include a Session header. If 403 response's Session header indicates "Invalid-MAC" then if the client had used HTTPS then the client SHOULD warn the user, otherwise the client SHOULD retry. If the 403 response's Session header indicates "Session-expired" then the client SHOULD renew the session (see Section 6.2). Otherwise the client must assume that the old session has been destroyed (e.g., because of a logout or server state data loss) and may establish a new session.

If MAC validation of a response fails the the client MUST act as though a 400 (bad request) had been sent instead. If the request was idempotent the client SHOULD retry, otherwise recovery is not specified.

6.2. Session Expiration and Renewal

If the server decides that a session is no longer valid then the server should respond with a 401 status code. The client should then re-authenticate or establish a new unauthenticated session, using the Previous-Session-URI and Previous-Session-State session-params of the new Session-Init to indicate that the old session is being "renewed".

6.3. Alternative: Define Session Scheme for WWW-Authenticate

One possibility that has some appeal would be to define a new HTTP authentication scheme called "Session" (say) and use that instead of the "Session" header defined above. The primary advantage to the WWW-Authenticate approach is that it fits the existing HTTP authentication framework, allowing a server to present to an application the user authentication information embedded in the session state as if the user were re-authenticated in each request. Session continuation can then be seen as a form of fast re-authentication.

7. Logout

To logout the client SHOULD perform a DELETE of the session URI.

8. Inquiring Session Status

The client MAY do a GET of the session URI. The semantics of the response body for this are not specified here. As explained in Section 4, servers also may GET a session URI; see Section 4 for more details.

9. Analysis

Quite clearly this protocol meets requirements 1, 2, 3, 5, and 11 from I-D.williams-websec-session-continue-prob.

The security requirements are also met:

requirement 4

> The active cookie recovery attacks on TLS we consider are adaptive chosen plaintext attacks. These attacks depend on the cookies sent by the client being the same in every request. This protocol uses MAC of at least channel bindings data (which doesn't change for any one connection) salted (so to speak) with a nonce. This use of nonces causes the MAC sent to be different for each request, which defeats the known cookie recovery attacks on TLS. Note that we assume confidentiality protection from TLS; clients MUST NOT negotiate cipher suites that provide no confidentiality protection.

requirement 6

This is clearly met by the use of a MAC keyed with a session key not available to attackers. This clearly depends on implementations having decent entropy sources, but this is no different than for TLS. Note, however, that insecure session initiation with key assertion is clearly insecure relative to passive attackers, as well as active attackers that can redirect packet flows so they can observe session initiation.

requirement_7

This is clearly met by prefixing an indicator of whether TLS is used or not to the MAC input.

requirement_8

The use of channel bindings as an input to the MAC meets this requirement.

requirement_9

This requirement is clearly met by having DELETE of a session URI terminate a session. It is important that clients promptly destroy any remnant of deleted sessions' state so that servers get no benefit from not deleting sessions when the clients demand it.

requirement_10

This is clearly met by using headers that proxies should pass unmodified.

10. IANA Considerations

This document creates a number of new HTTP request and response headers. These headers will need to be added to the HTTP header registry: <TBD>.

11. Security Considerations

This session continuation protocol appears to meet the requirements outlined in I-D.williams-websec-session-continue-prob. [XXX Add analysis. In particular explain how MAC(CB + nonce) is sufficient to defeat BEAST and CRIME.]

This proposal meets security requirements from the problem statement I-D.williams-websec-session-continue-prob. See Section 9 for details.

[...]

12. Normative References

rfc2119 (http://xml.resource.org/public/rfc/bibxml/reference.RFC.2119.xml)

<reference anchor='I-D.williams-websec-session-continue-prob'> <front> <title>Hypertext Transport Protocol (HTTP) Session Continuation: Problem Statement</title> <author initials='N.' surname='Williams' fullname='Nicolas Williams'> <organization /> </author> <date month='January'

day='1' year='2013' /> <abstract><t>Abstract One of the most often talked about problems in web security is âcookiesâ. Web cookies are a method of associating requests with âsessionsâ that may have been authenticated somehow. Cookies are a form of bearer token that leave much to be desired. This document describes the session âcontinuationâ problem for the HyperText Transport Protocol (HTTP).</t></abstract> </front> <seriesInfo name='Internet-Draft' value='draft-williams-websec-session-continue-prob-00' /> <format type='TXT' target='http://www.ietf.org/internet-drafts/draft-draft-williams-websec-session-continue-prob-00.txt' /> </reference>

rfc2104 (http://xml.resource.org/public/rfc/bibxml/reference.RFC.2104.xml)

rfc2616 (http://xml.resource.org/public/rfc/bibxml/reference.RFC.2616.xml)

rfc5246 (http://xml.resource.org/public/rfc/bibxml/reference.RFC.5246.xml)

rfc5056 (http://xml.resource.org/public/rfc/bibxml/reference.RFC.5056.xml)

rfc5929 (http://xml.resource.org/public/rfc/bibxml/reference.RFC.5929.xml)

rfc5869 (http://xml.resource.org/public/rfc/bibxml/reference.RFC.5869.xml)

13. Informative References

rfc2617 (http://xml.resource.org/public/rfc/bibxml/reference.RFC.2617.xml)

rfc5849 (http://xml.resource.org/public/rfc/bibxml/reference.RFC.5849.xml)

I-D.ietf-oauth-v2 (http://xml.resource.org/public/rfc/bibxml3/reference.I-D.ietf-oauth-v2.xml)

I-D.hallambaker-httpintegrity (http://xml.resource.org/public/rfc/bibxml3/reference.I-D.hallambaker-httpintegrity.xml)