

Internet Engineering Task Force	P. M. Hallam-Baker
Internet-Draft	Comodo Group Inc.
Intended status: Standards Track	April 04, 2013
Expires: October 06, 2013	

HTTP Session Management

draft-hallambaker-httpsession-00

Abstract

The HTTP Session Management Mechanism provides a mean of securely establishing a persistent authentication session between a HTTP client and server that does not rely on the presentation of a confidential bearer token. The Session Management Mechanism is intended to provide a replacement for the existing HTTP State Management Mechanism (Cookies) for this purpose.

This document defines the HTTP Accept-Session, Set-Session and Session headers and specifies their use to establish symmetric authentication keys and their use to authenticate and verify specific parts of an HTTP message. Other means by which keys used to authenticate the messages are established are outside the scope of this document.

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 October 06, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

- 1. Introduction**
 - 1.1. Relationship to Other Authentication Technologies**
 - 1.2. Example: Web Browser User Authentication**
 - 1.3. Use in Web Services**
 - 2. Session Context**
 - 2.1. Fixed Session Context**
 - 2.1.1. Id: Identifier**
 - 2.1.2. Key: Authentication Key**
 - 2.1.3. MAC: Message Authentication Code Algorithm**
 - 2.1.4. Scope Attributes**
 - 2.1.5. Replay Attack Prevention**
 - 2.1.6. Direction**
 - 2.1.7. TLS Binding (Fixed)**
 - 2.1.8. Domain: String**
 - 2.2. Session Context State Attributes**
 - 2.2.1. Expiry time: Max-Age**
 - 2.2.2. Now: Time Offset (Time)**
 - 2.2.2.1. Now: Last Now (Time)**
 - 2.2.3. Count: Last Count (Count)**
 - 2.2.4. Nonce: Last Nonce (Nonce)**
 - 3. Syntax**
 - 3.1. Accept-Session Header**
 - 3.2. Set-Session Header**
 - 3.3. Session Header**
 - 3.3.1. Required Attributes**
 - 3.3.1.1. Attribute Value=[base64(value)] (required)**
 - 3.3.1.2. Replay Attack Prevention Attributes**
 - 3.3.2. TLS Channel Binding Attributes**
 - 3.3.2.1. Attribute tlsu=[value]**
 - 3.3.2.2. Attribute tlss=[value]**
 - 3.3.3. Preparing the Input to the Authentication Algorithm**
 - 4. Processing**
 - 4.1. Calculating the Authentication Value**
 - 4.1.1. Start line**
 - 4.1.2. Canonical Headers**
 - 4.1.3. Message Content**
 - 4.2. Generating a Session Header**
 - 4.3. Authenticating a HTTP Message under a Session Context**
 - 5. Security Considerations**
 - 5.1. Data outside the specified scope is not authenticated**
 - 5.2. Truncated Hash Algorithms**
 - 5.3. Randomness of Secret Keys and nonces**
 - 5.4. Weak Ciphers**
 - 6. IANA Considerations**
 - 7. References**
 - 7.1. Normative References**
 - 7.2. Non Normative References**
- Author's Address**

1. Introduction

The HTTP State Management Mechanism 'Cookies'[\[RFC6265\]](#) was intended to allow HTTP [\[RFC2616\]](#) servers to let servers maintain a stateful session over the mostly stateless HTTP

protocol. While the exchange of static tokens is an acceptable mechanism for maintaining state, use of static tokens as bearer tokens for an authentication mechanism is not. Such tokens are not bound to any part of the message they purport to authenticate and may be disclosed to intermediaries.

While use of TLS transport provides a confidentiality enhancement for HTTP content, recent research [CRIME], [BEAST] demonstrates that relying on a transport or network layer to protect the confidentiality of a bearer authentication token is fundamentally unsound. The interaction of HTTP header compression mechanisms and a Turing complete active code mechanism under control of the attacker produces a threat model in which the capabilities afforded the attacker far exceed the capabilities that it is sensible to expect a protocol design to resist.

The HTTP Accept-Session, Set-Session and Session headers provide a simple and effective means of maintaining a HTTP authentication session without passing static authentication data in either direction after the authentication session has been established. The design of the Set-Session and Session headers permit them to be used as a replacement for the Set-Cookie and Cookie headers in situations where they are supported by both the client and the server and ensure correct behavior by intermediaries conformant to the HTTP specification.

A HTTP authentication session MAY be established inband by means of the Set-Session header. The Set-Session header specifies a unique identifier for the session and determines the session parameters including the cryptographic algorithm and shared key.

Applications SHOULD make use of cryptographic enhancements to protect the confidentiality of a session context established using the Set-Session header.

Clients and Servers MAY support other means of establishing a HTTP authentication session. For example in a federated authentication scheme such as SAML, Kerberos or OpenID, the authentication session might be provided by a third party.

Once the HTTP authentication session is established, a Session header is added to HTTP requests and/or responses as directed by the session context. The session header specifies the session identifier and an authentication value calculated over portions of the HTTP message and other attributes to which it is bound as directed by the corresponding session context. The bound attributes and portions of the HTTP message cannot then be changed without invalidating the authentication value.

The use of bound attributes permits protection against TLS channel rebinding and/or HTTP message replay attacks.

The portions of a HTTP message to which it is desirable to bind an authentication session depend on the situation. Binding the authentication session to the message content prevents modification of the content but imposes more constraints on implementations than binding to the message start line. Interactions with intermediaries and in particular intermediaries that are not fully compliant with the HTTP specification also raise concerns Web browsers are typically coded to be tolerant of such implementation defects and operate despite unauthorized modification of content by caches and other intermediaries. The preferred behavior of a Web Service client in such situations is likely to be to abort the transaction rather than risk continuing with corrupted data.

1.1. Relationship to Other Authentication Technologies

The term 'user authentication' is applied to three separate concerns; credential management, credential presentation and session continuation.

Credential Management describes the means by which credentials are created, issued and revoked.

Credential Presentation describes the means by which a party demonstrates holdership of a credential to establish an authentication session.

Session Continuation describes the means by which a party demonstrates that a particular transaction is taking place within the context of a particular authentication session.

The HTTP Session Management Mechanism is designed to support only Session Continuation and to compliment existing and future mechanisms for Credential management and Credential Presentation. While a session continuation mechanism is not in itself a solution to the problem of user authentication, the provision of a robust session continuation mechanism that does not depend on a bearer token addresses the most challenging problem facing the designers of SAML, OpenID and OAUTH.

1.2. Example: Web Browser User Authentication

The principal mechanism for user authentication in use today is to present a HTML form in which the user enters their username and password.

This approach has many known defects. These include the risk of impersonation of the Web site causing the user to enter their username and password into a form controlled by the attack and the strong likelihood that the user will use the same password across multiple sites. But since they are related to the functions of Credential Management and Presentation, these defects are out of scope of this document.

The client indicates that it supports the session header by including one or more Accept-Session headers in the request transferring the username and password values. The Accept-Session header specifies the scope and replay binding options that the client offers to support.

[If a public key exchange mechanism such as Diffie-Hellman was to be supported in the protocol, the client would specify its public Diffie-Hellman key in the request. Note however that this raises a tradeoff between privacy and performance concerns as using the same DH public key for each request provides the server with an easily linkable identifier.]

```
POST /login.php HTTP/1.1
Host: example.com
Cache-Control: no-store
Content-Type: application/x-www-form-urlencoded
Content-Length: 29
Accept-Session: Start=required Request=required Content=optional Time=required

username=skroob&password=1234
```

If the browser does not specify a Accept-Session header the server MAY reject the connection request entirely or fallback to the traditional Cookie mechanism as determined by site policy.

If the service accepts the offer of session management support, it includes a Set-session Header in the response specifying the session context:

```
HTTP/1.1 201 OK
Content-Length: 35
Set-Session:
  Id=TUMnor00SjHHS7D2uFcG1RYJ0Hd3eibwe0ogptoNMQuCYmCHFHAJcJlyvi
  j8WoXDg1TS0kctnmoBz18W0NLS1cgSyZcmsAyoWs8y1Rn2Z102WBgoWrFI0qPa4
  oB29dgs/ei6ieINZtmvXNCm2NUkWA==
  Key=7eb219188339135ba51e8715f3900bfb974995e145d6e490e4adbbdb26f4bb4
  Alg=HMAC-SHA256 Start=True Request=True Time=True Now=745531 Domain=example.com
  Max-Age=31536000

<html><h1>Authenticated</h1></html>
```

In this case the server avoids the need to track per client state by using a time based mechanism to avoid replay attacks and storing the state associated with the client session as encrypted data within the session identifier. The scope of the content binding is limited to the start line and the timer to be used for replay attack prevention has an offset 745531 seconds in the past.

Once the session has been established, the client MUST include a Session header in subsequent HTTP requests made to the specified DNS domains.

```
GET /status.php HTTP/1.1
Host: example.com
Cache-Control: no-store
Content-Type: application/x-www-form-urlencoded
Content-Length: 29
Session: Id=TUMnor00SjHHS7D2uFcG1RYJ0Hd3eibwe0ogptoNMQuCYmCHFHAJcJlyvi
  j8WoXDg1TS0kctnmoBz18W0NLS1cgSyZcmsAyoWs8y1Rn2Z102WBgoWrFI0qPa4
  oB29dgs/ei6ieINZtmvXNCm2NUkWA==
  Value=cjkMkfnnYP8JYWZAbRLvtpqImmOK3rsr0T1XcvAgHDk=;
  Now=745533
```

In this case the session scope does not specify responses and so the response does not require an Session header but a server MAY provide one so as to specify updated values for the replay attack prevention attributes Now and/or Count. Whenever a Session header is present the Id and Value attributes MUST be specified and correct:

```
HTTP/1.1 201 OK
Content-Length: 35
Session:
  Id=TUMnor00SjHHS7D2uFcG1RYJ0Hd3eibwe0ogptoNMQuCYmCHfHAJcJlyvi
  j8WoXDglTS0kctnmoBz18W0NLS1cgSyZcmsAyoWs8y1Rn2Z102WBgoWrFIOqPa4
  oB29dgs/ei6ieINZtmvXNCm2NUkWA==
  Value=cjkMkfnnYP8JYWZAbRLvtpqImmOK3rsrOT1XcvAgHDk=;
  Now=745532

<html><h1>Shield is Closed</h1></html>
```

In this particular instance the clock at the server is running behind that of the client requiring the timer offset value to be decreased by one second. To ensure that the replay attack protection values only increase or stay the same, the client uses the last value of the old time offset until the new time offset value has superceded it.

The Web Browser MAY terminate the session by simply deleting the session context information from its store preventing reuse. A client MAY inform the server that the session context is about to be deleted by including a Session header with the Deleted attribute:

```
HEAD /status.php HTTP/1.1
Host: example.com
Session: Id=TUMnor00SjHHS7D2uFcG1RYJ0Hd3eibwe0ogptoNMQuCYmCHfHAJcJlyvi
  j8WoXDglTS0kctnmoBz18W0NLS1cgSyZcmsAyoWs8y1Rn2Z102WBgoWrFIOqPa4
  oB29dgs/ei6ieINZtmvXNCm2NUkWA==
  Value=cjkMkfnnYP8JYWZAbRLvtpqImmOK3rsrOT1XcvAgHDk=;
  Deleted
```

A server may inform the client that the session has been terminated by including a Session header with the Deleted attribute in the response.

1.3. Use in Web Services

Use of HTTP Session Management simplifies implementation of Web Services. Using the SOAP [TBS] approach a Web Service message is encoded in XML [TBS], wrapped in a SOAP envelope and a WS-Security [TBS] header with an XML Signature [TBS] attached. The whole package is then attached to a HTTP message as a content payload.

This approach involves a considerable degree of complexity and in most cases does nothing more than attach an authentication data. Carrying the authentication value as a HTTP header eliminates the need for the SOAP and WS-Security layers entirely.

Use of session management in Web Services presents different requirements and constraints. In the case of an entirely new Web Service with no deployment history, there is no need to consider support for legacy code at all, eliminating one of the principal constraints governing use of new HTTP protocol features in Web Browsers.

A Web Service specification MAY require clients and servers to support and/or use scope and/or replay prevention mechanisms that are optional in this document. For example an Web Service exchanging payment transaction requests in XML is likely to require the authentication scope to include the message content and to support counter and nonce based replay attack prevention mechanisms.

A single HTTP message MAY have multiple Session headers. This facilitates support for multi-party transactions in which A submits a transaction to B who countersigns it and passes it to C who is required to check that she has proof of agreement by both A and B.

Use of the Session header permits the developer to isolate integrity and authentication checks to a single point of control, as is advised by best security practice. The security monitor examines a HTTP message, verifies that the required integrity data is present and correct and only passes the payload on for processing by the Web Service itself if and only if the verification checks have been

passed.

2. Session Context

The processing of the Session header is determined by the session context which consists of a set of fixed attributes that remain constant for the lifetime of the session and state attributes that are updated as Session headers are generated and verified.

2.1. Fixed Session Context

The fixed session context elements are set when the session is established and remain constant for the lifetime of the session. The values specified can only be changed by establishing a new session which **MUST** have a different session identifier.

2.1.1. Id: Identifier

The session identifier is a statistically unique sequence of binary data which **SHOULD** be unique, **MUST** be statistically unique, **SHOULD** be less than 512 octets in length and **MUST NOT** be longer than 4096 octets in length.

Servers **MAY** avoid the need to maintain per-session server side state by encoding the some or all of the fixed session context parameters in to the identifier. Servers **MUST** ensure that appropriate cryptographic enhancements are employed to authenticate the session context and protect the confidentiality of the authentication key. The scheme used to construct the session identifiers used in the examples is described in Appendix [TBS]

2.1.2. Key: Authentication Key

The cryptographic key to be used to calculate the authentication value.

2.1.3. MAC: Message Authentication Code Algorithm

The message authentication algorithm to be used to calculate the authentication value.

HMAC construction [RFC 2104]:

HMAC-SHA256-64

HMAC using the SHA-1 algorithm with the output truncated to the first 64 bits.

HMAC-SHA256-128

HMAC using the SHA-1 algorithm with the output truncated to the first 128 bits.

HMAC-SHA512-256

HMAC using the SHA-1 algorithm with the output truncated to the first 256 bits.

HMAC-SHA2-256-128

HMAC using the SHA-2 algorithm with the output truncated to the first 128 bits.

HMAC-SHA2-512-256

HMAC using the SHA-2 algorithm with the output truncated to the first 256 bits.

CMAC Construction [RFC 4493]:

CMAC-AES128-64

The AES algorithm employed in CMAC mode with a 128 bit key and the output truncated to the first 64 bits.

CMAC-AES128

The AES algorithm employed in CMAC mode with a 128 bit key and the entire output.

CMAC-AES256

The AES algorithm employed in CMAC mode with a 256 bit key and the entire output. NB: RFC 4493 only specifies CMAC for 128 bit keys so this would need an extension.

2.1.4. Scope Attributes

The scope attributes specify which parts of the message are authenticated.

[Separating the scope attribute from the authentication context so that it is specified in each Session header would permit the scope of the authentication check to be declared to

intermediaries and allows the same authentication context to be used to authenticate different portions of the HTTP message separately. I do not see the need to do this however and doing this would likely lead to implementations that are vulnerable to downgrade attacks by scope stripping.]

The scope is specified by the start, header and content attributes. The order in which the scope attributes are specified in the HTTP Set-Session header is immaterial. The scope is always constructed in the same order as the elements occur in a HTTP message, i.e. start, dummy headers and content.

Content: Boolean

If set true, the specified scope includes the message body. The content transfer encoding (e.g. chunked) is ignored for the purpose of determining the content.

ContentDigest: Label

If a message digest algorithm is specified the authentication scope MAY be calculated indirectly by first calculating a Message Digest value over the content and using the resulting value in place of the actual content value to calculate the Message Authentication Code value.

Start: Boolean

If set true, the specified scope includes the message start line. This being the request Line in the case of a request and the status line in the case of a response.

[The inclusion of HTTP headers in the authentication scope was considered but rejected as unnecessarily complex. At the very least headers would have to be duplicated. This does however have the unfortunate side effect of leaving the content meta-data outside the authentication scope. Perhaps there is a way that this could be finessed. For example a content-metadata option?]

2.1.5. Replay Attack Prevention

Three mechanisms for preventing message replay attacks are defined.

Nonce: Boolean

Challenge response mechanisms are supported by the nnonce and cnonce attributes. The challenger specifies a new nonce using the nnonce attribute which the responder MUST use to calculate the authentication value. In the case that the nonce value to be used cannot be determined by the context, an authentication protocol MAY require the responder to return the value of the challenge nonce using the nnonce attribute.

This approach provides a very high degree of protection but is limited to sequential protocols in which there is only one exchange in progress at the same time.

Counter: Boolean

Counter based mechanisms are supported by the count attribute. The value of a counter MUST increase for successive transactions within the same transaction stream. Concurrency MAY be supported by specifying multiple streams but this requires a separate counter state to be maintained for each transaction stream.

Time: Boolean

Time based approaches are supported by the time attribute. If the value of the time attribute falls within the permitted acceptance window, the message MAY be accepted. Otherwise the message MUST be rejected.

Using a time based approach avoids the need to maintain state at either the client or server. The principal disadvantage of this approach being that the mechanism only protects against a replay attack within a specific time.

Another disadvantage to the time based approach is that it relies on the sender and receiver maintaining a tolerably close time synchronization over the duration of the transaction and for the latency introduced by the communication path being tolerably small.

2.1.6. Direction

A session MAY be defined to apply to requests only, responses only or to both requests and responses.

Request: Boolean

This session context applies to requests.

Response: Boolean

This session context applies to responses.

2.1.7. TLS Binding (Fixed)

The TLS binding attribute specifies whether TLS channel binding is to be used.

2.1.8. Domain: String

The DNS Domain(s) to which the session context applies. The syntax and semantics of the Domain attribute are identical to those of the Domain attribute of the Cookie header defined in [1].

2.2. Session Context State Attributes

2.2.1. Expiry time: Max-Age

The time at which the session expires. To avoid the need for the client or server to have access to a realtime clock, Set-Session and Session headers specify the expiry time as the remaining lifetime of the session from the instant the header is generated in seconds.

A server MAY update the value Max-Age value to extend the lifetime of the session before expiry by specifying a new value for Max-Age in the Session header.

2.2.2. Now: Time Offset (Time)

The Time Offset value is used to calculate the value of the Now attribute in the session header and is only required when the Time replay protection mechanism is in use.

To avoid the need for clients or servers to have access to a reference time source, time values used to protect against replay attack are specified relative to an arbitrary epoch start time specified by the server. The Time Offset value is the difference between the time epoch specified by the server and the local time according to the machine. A server MAY use the same epoch start time for all clients or use a different epoch for each one.

2.2.2.1. Now: Last Now (Time)

If the local clock at the client runs faster or slower than that of the server, a timing discrepancy emerges over time. A client SHOULD and a server MAY correct for such inaccuracies by noting the value of the now attribute specified by the other party and adjusting the local time offset value accordingly provided that the mechanism employed to do so ensures that the values of the now attribute in a HTTP message is never less than the value specified in a previous header.

Recording the value of the last value of Now specified in a header permits this condition to be met.

2.2.3. Count: Last Count (Count)

If counter based replay attack prevention is in use the client and server MUST maintain a record of the last value of the counter for each concurrent stream active within the session.

2.2.4. Nonce: Last Nonce (Nonce)

If nonce based replay attack prevention is in use, the parties MUST maintain a record of the last nonce value so as to be able to return it when necessary.

In most circumstances the nonce value is used immediately and need not be stored.

3. Syntax

3.1. Accept-Session Header

The Accept-Session header is used to negotiate the establishment of an authentication context. When used in a request the Accept-Session header specifies a set of acceptable parameters for the session context.

MAC=[Label(,Label)*]

The message authentication algorithms the client is willing to support.

Content=[Optional | Required]
Offers or requires the inclusion of the message content in the authentication scope.

ContentDigest=[Optional | Required]
Offers or requires the inclusion of the message content by means of a content digest in the authentication scope.

Start=[Optional | Required]
Offers or requires the inclusion of the message start line in the authentication scope.

Request=[Optional | Required]
Offers or requires the use of a Session header in a request message.

Response=[Optional | Required]
Offers or requires the use of a Session header in a response message.

TLSBinging=[Optional | Required]
Offers or requires the use of TLS Binding [Need to think this through further]

Nonce=[Optional | Required]
Offers or requires the use of the nonce replay attack prevention mechanism.

Counter=[Optional | Required]
Offers or requires the use of the counter replay attack prevention mechanism.

Time=[Optional | Required]
Offers or requires the use of the time replay attack prevention mechanism.

When used by the client to offer the use of an authentication session, all header attributes are optional. Note however that even though it is permissible for a client to offer an empty Accept-Session header, doing so does not allow a valid session context to be established as the server is required to specify at least an authentication scope and MAC algorithm from amongst those offered by the client.

3.2. Set-Session Header

Id=[base64(value)]
The session context identifier as in base64 encoding.

Key=[base64(value)]
The cryptographic key to be used to calculate the authentication value in base64 encoding.

MAC=[Label]
The message authentication algorithm to be used to calculate the authentication value as defined in [IANA TBS].

Content
Specifies the inclusion of the message content in the authentication scope.

ContentDigest
Specifies the inclusion of the message content by means of a content digest in the authentication scope.

Start
Specifies the inclusion of the message start line in the authentication scope.

Request
Specifies the use of a Session header in a request message.

Response
Specifies the use of a Session header in a response message.

TLSBinging
Specifies the use of TLS Binding [Need to think this through further]

Nonce
Specifies the use of the nonce replay attack prevention mechanism.

Counter
Specifies the use of the counter replay attack prevention mechanism.

Time
Specifies the use of the time replay attack prevention mechanism.

Max-Age

Now

A Set-Session header MUST contain the following elements:

Id

Key
MAC
At least one Scope attribute offered by the client
At least one direction attribute
A Max-Age value

3.3. Session Header

The Session header has the tag 'Session' and takes a sequence of attribute values as follows:

[Insert ABNF here]

3.3.1. Required Attributes

3.3.1.1. Attribute Value=[base64(value)] (required)

The value attribute specifies the value resulting from applying the authentication context and nonce (if present) to the specified scope.

3.3.1.2. Replay Attack Prevention Attributes

Three means of protection against replay attack are supported:

An authentication protocol MAY employ multiple replay attack protection schemes within the same exchange. For example a time based approach MAY be employed to perform an initial check before retrieving the state information needed to validate a Counter or Challenge Response based mechanism.

3.3.1.2.1. Attribute nnonce=[base64(value)], rnonce=[base64(value)]

The nnonce and rnonce attributes specify a nonce value to be used in combination with a challenge-response mechanism defined by the specified authentication context. The nnonce attribute is used to specify a new nonce value, the rnonce attribute is used to specify a returned nonce value.

3.3.1.2.2. Attribute count=[hex(stream),hex(count)]

Specifies a stream identifier and a count value that MUST increase monotonically for successive messages with the same identifier. The stream and count values are specified as hexadecimal encoded positive integers.

3.3.1.2.2.1. Attribute time=[value]

Specifies a time value to be used in combination with the specified authentication context. The format of the time value is determined by the authentication context.

3.3.2. TLS Channel Binding Attributes

TLS channel binding is used to ensure that the HTTP session is protected by TLS and to prevent man in the middle attacks against TLS.

3.3.2.1. Attribute tlsu=[value]

Specifies the TLS unique channel binding as specified in [\[RFC5929\]](#).

3.3.2.2. Attribute tlss=[value]

Specifies the TLS server end point channel binding as specified in [\[RFC5929\]](#).

3.3.3. Preparing the Input to the Authentication Algorithm

[Should specify how the content scope is assembled and how the replay attack attributes are included within it.]

4. Processing

4.1. Calculating the Authentication Value

The input to the MAC algorithm is the concatenation of the following values.

The Start Line

Is included if and only if the value of the start attribute of the session context is true.

The Canonical HTTP Headers

Are always included.

The Message Content

Is included if and only if the value of the content attribute of the session context is true.

4.1.1. Start line

The Start line is the HTTP start line including the final CRLF.

Example:

4.1.2. Canonical Headers

The canonical form of the header(s) specified for inclusion in the authentication scope by the session context sorted into alphabetical order. At present only the Session header is specified and MUST always be included.

The canonical Session header contains all the attributes of the Session header to be added to the HTTP message with the exception of the Value attribute. Attributes MUST be specified in alphabetical order.

Example:

4.1.3. Message Content

If the Content-Digest parameter of the session context is empty the Message content value is the actual value of the message content ignoring any transfer encoding but after any content-encoding has taken place.

If the Content-Digest parameter of the session context specifies at least one Message Digest algorithm, the sender MAY choose to calculate the authentication value over the actual value of the content as specified above or first apply one of the specified message digest algorithms to the actual value of the message content as specified above and then calculate the authentication value over the resulting digest value.

Example:

4.2. Generating a Session Header

Generating a Session Header requires the following steps to be performed:

The Session header parameters are calculated according to the session context.

If necessary, the session context is updated to reflect new values of relevant replay attack prevention attributes.

The authentication value is calculated over the specified scope.

The Session header is added to the HTTP headers.

4.3. Authenticating a HTTP Message under a Session Context

This is the opposite of generating.

5. Security Considerations

5.1. Data outside the specified scope is not authenticated

The integrity check only extends to the portions of the message that are within the specified scope.

5.2. Truncated Hash Algorithms

If the authentication context permits the use of a truncated MAC, it MUST specify the minimum length of the MAC after truncation and verifiers MUST reject MAC values shorter than that length as invalid.

5.3. Randomness of Secret Keys and nonces

The security of any cryptographic protocol relies on the difficulty of guessing secret keys. Secret keys and nonces SHOULD be generated using a mechanism that ensures that the range of possible values is sufficiently large to prevent 'brute force' guessing attacks. For more information see [\[RFC4086\]](#).

5.4. Weak Ciphers

Specification of the cryptographic algorithms used to construct the Integrity header value is implicit in the authentication context identifier and thus outside the scope of this specification.

6. IANA Considerations

Add the 'Session' and 'Set-Session' headers to the list of provisional HTTP headers.

[Upgrade if/when this becomes an RFC]

Create a registry for Session Header attributes. The initial contents of the registry to be:

[Stuff from rest of document.]

7. References

7.1. Normative References

- [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.
- [RFC2965]** Kristol, D. M. and L. Montulli, "HTTP State Management Mechanism", RFC 2965, October 2000.
- [RFC5246]** Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5929]** Altman, J., Williams, N. and L. Zhu, "Channel Bindings for TLS", RFC 5929, July 2010.
- [RFC4086]** Eastlake, D., Schiller, J. and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

7.2. Non Normative References

- [RFC5652]** Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.
- [RFC3275]** Eastlake, D., Reagle, J. and D. Solo, "(Extensible Markup Language) XML-Signature Syntax and Processing", RFC 3275, March 2002.
- [RFC4120]** Neuman, C., Yu, T., Hartman, S. and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.
- [RFC6265]** Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.

Author's Address

Phillip Hallam-Baker

Comodo Group Inc.

E-Mail: philliph@comodo.com