Internet Engineering Task Force	P. Hallam-Baker
Internet-Draft	Comodo Group Inc.
Intended status: Standards Track	November 14, 2012
Expires: May 18, 2013	

TOC

HTTP Integrity Header
draft-hallambaker-httpintegrity-02

Abstract

The HTTP Integrity header provides a means of specifying authentication data in HTTP requests and responses. This document defines the HTTP integrity header and specifies its use to authenticate and verify specific parts of an HTTP message. The means by which the symmetric or asymmetric keys used to authenticate the messages is 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 May 18, 2013.

Table of Contents

---

1. Introduction

TOC

The HTTP Integrity header provides a simple and effective means of authenticating components of a HTTP message **[RFC2616]** inband without disclosure of the secret(s) used as the basis for authentication.

This approach has considerable advantages over the traditional approaches of using HTTP Cookies **[RFC2965]** containing an authentication secret, embedding authentication data within the HTTP message content or relying on integrity checks in other protocol layers.

Used in conjunction with an appropriate means of key exchange, the Integrity header provides an equivalent security functionality to the use of authentication cookies without the vulnerabilities intrinsic to the use of static authentication secrets that are disclosed to the verifying party en-clair to effect verification.

Use of a HTTP header to provide authentication information offers a simpler and more flexible approach than including the authentication information in the message content using schemes such as PKCS#7/CMS **[RFC5652]**, XML Signature **[RFC3275]** or JSON Signature [TBS]. The chief technical challenge in such specifications being to reliably identify the exact scope of the data being signed and the form of encoding. Moving the integrity check to a HTTP header permits the scope of the signature to be defined as the scope of the HTTP content body and the encoding to be the HTTP transport encoding.

Use of the HTTP Intergity header permits the same mutual authentication guarantees provided by TLS client authentication **[RFC5246]** without the need to provision client certificates and with considerably less complexity.

For simplicity, the HTTP Integrity header is strictly limited to identifying a authentication context, the HTTP transaction item(s) to be authenticated and the resulting authentication value. The means of establishing the keys and algorithms that make up the authentication context are outside the scope of this document.

In the typical case the authentication context identifier is a ticket (c.f. Kerberos **[RFC4120]**) that contains the account identifier, shared secret(s) and algorithm identifier required a Web Service protocol or Web Browser Authentication protocol. This approach permits a stateless server design in which the server does not store per-account keys.

1.1. Use in conjunction with TLS.

TOC

Transport Layer Security (TLS) **[RFC5246]** provides transport layer enhancements to protect the confidentiality and integrity of messages. Use of the HTTP Integrity header

compliments the use of TLS security rather than replacing TLS.

While TLS provides for server and client authentication, these controls are implemented at the transport layer and access to these features requires the traversal of a protocol layer boundary that is frequently undesirable or inappropriate. In particular the API available to the client may not expose the necessary functionality and many server deployments make use of external cryptographic accelerator devices that cause the TLS session to be terminated on an entirely different machine. Access control requires authorization in addition to authentication. Since authorization is fundamentally an application layer concern, attempts to carry authorization data at the transport layer tend to be rather unsatisfactory requiring both the sender and receiver to cross the protocol layer.

TOC

1.2. Use in Web Services

The HTTP Integrity header was originally developed to simplify implemenatation 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 achieves nothing more than attaching an authentication value. Carrying the authentication value as a HTTP header eliminates the need for the SOAP and WS-Security layers entirely. For example, the following example is a HTTP request in the Omnibroker connection protocol [TBS].

```
Post / HTTP/1.1
Host: example.com
Cache-Control: no-store
Content-Type: Application/json;charset=UTF-8
Content-Length: 78
Integrity: content=true;
    value=cjkMkfnnYP8JYWZAbRLvtpqImmOK3rsrOT1XcvAgHDk=;
    id=TUMnorO0SjHHS7D2uFcGlRYJ0Hd3eibwe0ogptoNMQuCYmCHfHAJcJlyvi
      j8WoXDglTSOkctnmoBzl8W0NLSlcgSyZcmsAyoWs8y1Rn2ZlO2WBgoWrFIOqPa4
      oB29dgs/ei6ieINZtmvXNCm2NUkWA==

{
"TicketRequest": {
"ChallengeResponse": "TctLOG74cwpm26YNpEibcQ==}}
```

A single HTTP message MAY have multiple Integrity 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 chek that she has proof of agreement by both A and B.

Use of the Integrity 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.

TOC

1.3. User Authentication and Authorization

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

> Credential management describes the means by which credentials are created, issued and revoked.
>
> Session management describes the means by which a party demonstrates holdership of a credential to establish an authentication session and the means by which that authentication session may be terminated.

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 Integrity header is designed to support only Session Continuation. 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.

TOC

2. Use

The Integrity header has two required attributes, the id attribute which identifies an authentication context and the value attribute which authenticates the HTTP message in which it is presented within the specified authentication context and optional attributes specified.

The authentication context comprises all the information used to authenticate and validate HTTP messages. This includes the choice of cryptographic algorithms, the keys and any protocol options such as the TLS channel binding or protections against replay attack.

Confining the choice of cryptographic algorithm, protocol etc. to the authentication context eliminates the need for identifiers to specify these attributes in the Integrity header. Note that while the Integrity header primarily designed for use with a symmetric key authentication algorithm (MAC), it MAY be employed to support use of an public key based scheme such as a digital signature.

The means by which the authentication context is established are outside the scope of this document. In the simplest approach, the authentication context might be passed as in-band plaintext in a prior HTTP exchange. In a more elaborate scheme the authentication context might be established using an authentication framework such as Kerberos, SAML or OpenID.

The authentication identifier MAY be of any length up to the maximum permitted by the client and server. This permits an authentication mechanism to avoid the need to maintain server side per-session state by encoding the server authentication context as encrypted data in the authentication identifier. Alternatively, the same approach might be used to avoid the need to maintain client side state.

The optional attributes are used to provide protection against replay attacks, specify the scope of the authentication check and protect against TLS man in the middle attacks.

TOC

2.1. Example of Use

[The following example is for illustrative purposes only to aid discussion of the draft and should not be part of a final document.]

The Plaintext Authentication exchange supports message authentication using a MAC and a key exchanged as plaintext in-band. The communication pattern consists of an initial exchange in which the server returns the credential and authentication context to the client followed by an indeterminate number of messages authenticated under that context.

TOC

2.1.1. Client Request

By definition, the protocol exchange is initiated by a client request. The client advises the server that it supports the use of the scheme by means of the Support header (in an actual protocol mechanism, this should probably be some other header but none appears to be a precise match).

```
Get / HTTP/1.1
Host: example.com
```

```
Support: Plaintext
```

TOC

2.1.2. Server Response

The server responds with the Authentication Context identifier, the key and the algorithm to use:

```
HTTP/1.1 201 OK
Plaintext:
    id=TUMnorO0SjHHS7D2uFcGlRYJ0Hd3eibwe0ogptoNMQuCYmCHfHAJcJlyvi
      j8WoXDglTSOkctnmoBzl8W0NLSlcgSyZcmsAyoWs8y1Rn2ZlO2WBgoWrFIOqPa4
      oB29dgs/ei6ieINZtmvXNCm2NUkWA==
    key=7eb219188339135ba51e8715f3900bfb974995e145d6e490e4addbbdb26f4bb4
    alg=HMAC-SHA256
```

From this point on the client and server protect their messages using the Integrity header as shown in the previous example.

TOC

2.1.3. Design Notes for Key Exchange

While the Plaintext mechanism described above offers better security than the use of HTTP cookes as bearer tokens, the construction is very much sub optimal.

In particular a full key exchange binding should describe the precise manner in which the scope is and the replay attack protection attributes are fed into the digest algorithm. It is also desirable for an authentication mechanism to use separate keys for different purposes, in particular use of separate keys for requests and responses.

As specified, the mechanism only provides protection against passive eavesdropping attacks if the Plaintext exchange is protected by an adequate confidentiality protection. For example the use of TLS. A stronger protection MAY be established by passing the key value through a Diffie Hellman key exchange.

In the Diffie Hellman approach, the key establishment would have the following pattern:

1. Client->Server
   Initiate conversation, signal support for DH key exchange option.
2. Server->Client
   Pass initial Authentication Context identifier and server Diffie Hellman key parameters.
3. Client->Server
   Pass client Diffie Hellman parameters, authenticate message under established key.
4. Server->Client
   Optionally specify a new Authentication Context identifier to reflect the fact that the initial aunthentication context has been updated to include the client information.

Note that even though the key exchange is only completed in the third message, all messages following and including the third message are protected.

A separate draft describing a lightweight exchange to replace the use of bearer tokens is planned. Such a draft should probably support both the plaintext and the Diffie Hellman approaches to ensure that there is no remaining excuse for http authentication cookie attrocities.

TOC

3. Syntax and options

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

[Insert ABNF here]

Four classes of attribute are currently specified:

Required
: Attributes that MUST always be specified. These are the Authentication Context Identifier attribute 'id' and the Authentication Value attribute 'value'.

Replay Attack Prevention
: Attributes used to implement replay prevention mechanisms.

Scope Attributes
: Attributes that specify the scope over which the authentication value is calculated

TLS Channel Binding Attributes
: Attributes used to protect against TLS channel rebinding and/or TLS channel stripping attacks.

3.1. Required Attributes

3.1.1. Attribute id=[base64(value)] (required)

The ticket attribute identifies the authentication context under which the authentication value has been generated. The attribute is an opaque sequence of octets in base64 encoding.

3.1.2. 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.

3.1.3. Replay Attack Prevention Attributes

Three means of protection against replay attack are supported:

Challenge-Response
: Challenge response mechanisms are supported by the nnonce and cnnonce 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 reponder to return the value of the challenge nonce using the rnonce 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
: 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
: 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 tollerably close time synchronization over the duration of the transaction and for the latency introduced by the communication path being tollerably small.

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 retreiving the state information needed to validate a Counter or Challenge Response based mechanism.

TOC

3.1.3.1. 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.

TOC

3.1.3.2. 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.

TOC

3.1.3.2.1. 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.

TOC

3.1.4. Scope Attributes

Scope attributes specify which parts of the message are authenticated.

Separating the scope attribute from the authentication context permits 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.

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

TOC

3.1.4.1. Attribute content=[true|false]

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.

TOC

3.1.4.2. Attribute start=[true|false]

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.

3.1.4.3. Attribute header=[escaped(headers)]

Specifies HTTP headers to be included in the specified scope following the escaped encoding defined in DKIM.

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.

3.2.1. Attribute tlsu=[value]

Specifies the TLS unique channel binding as specified in **[RFC5929]**.

3.2.2. Attribute tlss=[value]

Specifies the TLS server end point channel binding as specified in **[RFC5929]**.

3.3. Preparing the Input to the Authentication Algorithm

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

---

4. Security Considerations

4.1. 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.

4.2. 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.

4.3. 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]**.

4.4. 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.

---

5. IANA Considerations

Add the 'Integrity' header to the list of provisional HTTP headers.

[Upgrade if/when this becomes an RFC]

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

[Stuff from rest of document.]

---

6. References

6.1. Normative References

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

[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 (TXT, PS, PDF, HTML, XML).

[RFC2965] Kristol, D. and L. Montulli, "HTTP State Management Mechanism," RFC 2965, October 2000 (TXT, HTML, XML).

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security," BCP 106, RFC 4086, June 2005 (TXT).

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

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS," RFC 5929, July 2010 (TXT).

6.2. Non Normative References

[RFC3275] Eastlake, D., Reagle, J., and D. Solo, "(Extensible Markup Language) XML-Signature Syntax and Processing," RFC 3275, March 2002 (TXT).

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)," RFC 4120, July 2005 (TXT).

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)," STD 70, RFC 5652, September 2009 (TXT).

Author's Address

Phillip Hallam-Baker
Comodo Group Inc.
Email: philliph@comodo.com