Security Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 26, 2011

B. Burke
Red Hat
February 22, 2011

HTTP Header for digital signatures
draft-burke-content-signature-00

Abstract

This document describes the Content-Signature HTTP entity header. It is used to transmit one or more digital signatures comprised of metadata and the HTTP message body.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes the Content-Signature HTTP entity header. This header is used to transmit one more more digital signatures which are composed by signing both the HTTP message body and an optional set of transmitted metadata about the signature. While the formats like multipart/signed already allow you to transmit digital signatures, it requires HTTP clients and servers to be able to understand and parse the format to get at the enclosed message body even if they have no interest in the digital signature. The multipart/signed format also does not really specify how the signature is composed and leaves it up to the sender. While this specification does not mandate the digital signature algorithm to use, it does describe what is signed, particularly how metadata and headers are combined with the message body before they are signed.

In summary the goals and features of the Content-Signature header are:

- o Ability to transmit multiple digital signatures via an HTTP request or response entity header
- o Ability to transmit a set of standard and user-defined metadata about each signature and to include it as part of the signature signing and verification process
- o The receiver of the HTTP request or response should be able to ignore signature information.
- o Ability to include zero or more other HTTP headers within the calculation of the signature
- o Ability to include other attached signatures within the calculation of a particular signature

This specification is not meant to replace OAuth or the HTTP Digest protocol. While those protocols are more interested in guaranteeing the integrity of a message between a specific interaction between a client and server, Content-Header is merely a way to attach one or more signatures to a representation. In other words, Content-Signature can be completely orthogonal to the authentication and authorization protocol of the HTTP request.

2. Notational Conventions

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 RFC 2119 [RFC2119].

This document uses the Augmented Backus-Naur Form (ABNF) notation of RFC2616 [RFC2616], and explicitly includes the following rules from it: quoted-string, token, SP (space), and HEX.

3. The Content-Signature Header Field

The Content-Signature entity-header field provides a means for serializing one or more digital signatures and metadata associated with those signatures. The value of this header is a set of attributes which are name-value pairs delimited by the ";" character. Multiple values of Content-Signature are allowed if they are delimited by the "," character.

```
   Content-Signature        = "Content-Signature" ":" #content-signature-value
   content-signature-value = signature *( ";" metadata)
   signature                = "signature" "=" 1*HEX
   metadata                 = ( ("id" "=" (ptoken | quoted-string))
                            | ( "algorithm" "=" (ptoken | quoted-string))
                            | ( "values" "=" name-list)
                            | ( "headers" "=" name-list)
                            | ( "signature-refs" "=" name-list )
                            | ( "signer" "=" ( ptoken | quoted-string ) )
                            | ( "timestamp" "=" HTTP-date )
                            | ( "expiration" "=" HTTP-date )
                            | ( "nonce" "=" 1*DIGIT )
                            | ( other-metadata) )
   other-metadata           = ( paramname "=" ( ptoken | quoted-string ) )
   name-list                = paramname *( ":" paramname)
   paramname                  = 1*paramchar
   paramchar                = "!" | "#" | "$" | "%" | "&" | "'" | "("
                            | ")" | "*" | "+" | "-" | "." | "/" | DIGIT
                            | "<" | "=" | ">" | "?" | "@" | ALPHA
                            | "[" | "]" | "^" | "_" | "`" | "{" | "|"
                            | "}" | "~"
   ptoken                   = 1*ptokenchar
   ptokenchar               = "!" | "#" | "$" | "%" | "&" | "'" | "("
                            | ")" | "*" | "+" | "-" | "." | "/" | DIGIT
                            | ":" | "<" | "=" | ">" | "?" | "@" | ALPHA
                            | "[" | "]" | "^" | "_" | "`" | "{" | "|"
                            | "}" | "~"
```

3.1. Attribute Descriptions

Name: signature
Required: yes
Description:

Hex encoded string of the binary value of the signature

Name: id
Required: no
Description:

Identifies an included signature. This is useful when there are more than one signature value included within the Content-Signature header. Applications can use this "id" attribute to find the signature they are interested in verifying.

Name: algorithm
Required: no
Description:

Specifies the algorithm used to calculate the signature. The algorithm registry (Section 6.1) specifies possible but not exhaustive list of values.

Name: values
Required: no
Description:

A ":" delimited list of parameter names. These parameter names correspond to attribute metadata values within the Content-Signature value. The values of these parameters are used when creating and verifying the signature. See Creating and Verifying Signature (Section 4) section for more details.>

Name: headers
Required: no
Description:

A ":" delimited list of header names. These names correspond to HTTP header values included in the HTTP request or response. The values of these headers are used when creating and verifying the signature. See Creating and Verifying Signatures (Section 4) section for more details.>

Name: signature-refs
Required: no
Description:

A ":" delimited list of signature ids. These names correspond to other signatures included in the Content-Signature header. The names reference the "id" attribute of each of these included signatures. The "signature" attribute value of each of these identified signatures are used when creating and verifying the signature. See Creating and Verifying Signatures (Section 4) section for more details.>

Name: signer
Required: no
Description:

This is the identity of the signer of the message. It allows the receiver to look up verification keys within an internal registry. It also allows applications to know who sent the message if identity cannot be determined by authentication protocols.

Name: timestamp
Required: no
Description:

The time and date the message was signed. This gives the receiver the option to refuse old signed messages. The format of this timestamp is the Date format described in RFC 2616 [RFC2616].

Name: expiration
Required: no
Description:

The time and date the message should be expired. This gives the sender the option to set an expiration date on the message. The format of this timestamp is the Date format described in RFC 2616 [RFC2616].

Name: nonce
Required: no
Description:

Random number to ensure old communications cannot be replayed.

4. Creating and Verifying Signatures

Signatures are created by applying a digital signature algorithm to the contatenation a set of metadata with the body of the HTTP message. The order in which data is concatenated is very important

as verifiers must know this in order to verify the signature when they receive it. This is the order that will be applied:

```
values + headers + signature-refs + message-body
```

The values and headers pertain to the Content-Signature metadata of a specific Content-Signature value.For example, if the signer decides to include the signer and expiration attributes, the Content-Type header, and a text/plain message of "hello world", the base for the signature would look like this:

```
billSunday, 06-Nov-11 08:49:37 GMTtext/plainhello world
```

The Content-Signature header transmitted would look like:

```
Content-Signature: values=signer:expiration;
                   headers=Content-Type;
                   signer=bill;
                   expiration="Sunday, 06-Nov-11 08:49:37 GMT";
                   signature=0f341265ffa32211333f6ab2d1
```

To verify a signature, the verifier would recreate the signature by concatenating the attributes specified in the "values" attribute, HTTP headers defined in "headers" attribute, the hex signature values pointed to by "signature-ref" attribute, and finally the message body. The array of bytes produced should by verified with the decoded value of the "signature" attribute.

If there is an attribute declared within the "values" attribute that isn't specified in the Content-Signature header, it is assumed it is a secret held between the signer and verifier. If there is a header declared within the "headers" attribute that doesn't exist, the server may choose to abort if it cannot figure out how to reproduce this value.

Here's an example of multiple signatures. Let's say the Content-Signature header is initially set up like this with a message body of "hello":

```
Content-Signature: id=husband;
                   signature=0A01,
                   id=wife;
                   signature=0F02
```

Here, we have two initial signatures signed by two different entities, husband and wife (found by their id attribute). We want to define a third signature, marriage, that includes those signatures.

```
      Content-Signature: id=husband;
                         signature=0A01,
                         id=wife;
                         signature=0F02,
                         id=marriage;
                         signature-refs=husband:wife
                         signature=0D0330
```

The marriage signature would be calculated by the signing of this array of bytes:

```
      0A010F02hello
```

Which is the husband's signature + wife's signature + message-body.

If there is a signature reference declared within the signature-refs attribute that doesn't exist, the server may choose to abort if it cannot figure out how to reproduce this value.

5. OAuth Considerations

While Content-Signature is only a means to attach signatures to a transmitted representation, it is possible to use it as an authentication and authorization mechanism on top of OAuth2 [I-D.ietf-oauth-v2]. An authorization token can be obtained using OAuth mechanisms and attached as metadata to transmitted Content-Signature values. This is different than the MAC [I-D.hammer-oauth-v2-mac-token] protocol in that since Content-Signature is an entity-header, it can travel through and be forwarded by transparent gateways. The representation, along with the Content-Signature can make multiple hops until it reaches the protected resource. The protected resource can trust the forwarded representation because it has the desired auth-token all signed by the original sender.

An example of this is an application that listens to Twitter feed of a particular person. The application sends SMS messages on behalf of the individual to the individual's friends. The mobile provider bills the individual rather than the application. The signer is the individual. The transparent gateway is Twitter. The protected resource is the mobile provider.

Firstly, the individual obtains an authorization token using OAuth from the mobile provider which gives the individual permission to forward messages. The application obtains an authorization token from the mobile provider which gives it permission to send SMS messages on behalf of authenticated and authorized individuals. The

individual then posts a message to Twitter signing it including its authorization token.

```
   Content-Signature: signature=0A01;
                      values=mobile-auth-code:signer:nonce
                      mobile-auth-code=342a2f11;
                      signer=bill;
                      nonce=1
```

Twitter receives the messages and sends it to all the interested feeds. The application picks up the message from twitter. It adds an additional signature to the message that is comprised of its auth token, the individual's signature, and the message.

```
   Content-Signature: id=invidual;
                      signature=0A01;
                      values=mobile-auth-code:signer:nonce
                      mobile-auth-code=342a2f11;
                      signer=bill;
                      nonce=1,
                      id=forwarder;
                      signature=0B02;
                      values=mobile-forward-auth-code:signer;
                      signature-refs=individual;
                      mobile-forward-auth-code=4020FACE;
                      signer=application
```

The application then sends an SMS to each of the individual's friends attaching the signatures to each SMS message. An interesting thing to note here is that the application added additional metadata to the individual's signature, specifically the "id" attribute. Labeling each signature allows the mobile provider to sort out which signatures are which. Finally, the mobile provider looks at each message to the individual's and application's signatures. If approved it bills the individual, if not it bills the provider.

Granted this example is a bit contrived, but hopefully you get the picture.

6. IANA Considerations

6.1. Algorithm Registry

This specification establishes the digital signature algorithm registry which is a list of algorithm names that can be used as values for the "algorithm" attribute of Content-Signature.

6.1.1. Registering New Algorithms

(FYI: I copied this section from Link header draft) Algorithms are registered on the advice of (TBD)...

Registration requests consist of the completed registration template below, typically published in an RFC or Open Standard (in the sense described by [RFC2026], Section 7). However, to allow for the allocation of values prior to publication, the Designated Expert may approve registration once they are satisfied that a specification will be published.

The registration template is:

- Algorithm Name:
- Description:
- Reference:
- Notes: [optional]
- Application Data: [optional]

Registration requests should be sent to the [TBD]@ietf.org mailing list, marked clearly in the subject line (e.g,. "NEW DIGITAL SIGNATURE ALGORITHM REQUEST").

6.1.2. Initial Registry Contents

TBD

6.2. Attribute Registry

This specification establishes the Content-Signature attribute registry which is a list of attribute names that can be included as metadata within a Content-Signature.

6.2.1. Registering New Attributes

(FYI: I copied this section from Link header draft) Algorithms are registered on the advice of (TBD)...

Registration requests consist of the completed registration template below, typically published in an RFC or Open Standard (in the sense described by [RFC2026], Section 7). However, to allow for the allocation of values prior to publication, the Designated Expert may approve registration once they are satisfied that a specification

will be published.

The registration template is:

- o Attribute Name:
- o Description:
- o Reference:
- o Notes: [optional]
- o Application Data: [optional]

Registration requests should be sent to the [TBD]@ietf.org mailing list, marked clearly in the subject line (e.g,. "NEW DIGITAL SIGNATURE ALGORITHM REQUEST").

6.3. Registration Approval Process

(FYI: Stole this from Link header draft) Within at most 14 days of registration request, the Designated Expert(s) will either approve or deny the registration request, communicating this decision to the review list. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful.

Decisions (or lack thereof) made by the Designated Expert can be first appealed to Application Area Directors (contactable using app-ads@tools.ietf.org email address or directly by looking up their email addresses on http://www.iesg.org/ website) and, if the appellant is not satisfied with the response, to the full IESG (using the iesg@iesg.org mailing list).

7. Security Considerations

TBD

8. Acknowledgements

9. References

9.1. Normative References

[I-D.hammer-oauth-v2-mac-token]
Hammer-Lahav, E., "HTTP Authentication: MAC

Authentication", draft-hammer-oauth-v2-mac-token-02 (work in progress), January 2011.

[I-D.ietf-oauth-v2]
Hammer-Lahav, E., Recordon, D., and D. Hardt, "The OAuth 2.0 Authorization Protocol", draft-ietf-oauth-v2-12 (work in progress), January 2011.

[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.

9.2. Informative References

[InfRef] "", 2004.

Appendix A. An Appendix

Author's Address

Bill Burke
Red Hat

Email: bburke@redhat.com
URI: http://bill.burkecentral.com