Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 5, 2014

M. Cavage
Joyent
M. Sporny
Digital Bazaar

February 1, 2014

HTTP Signatures
draft-cavage-http-signatures-01

Abstract

When communicating over the Internet using the HTTP protocol, it is often desirable to be able to securely verify the sender of a message as well as ensure that the message was not tampered with during transit. This document describes a way to add origin authentication and message integrity to HTTP messages.

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 August 5, 2014.

Table of Contents

1. Introduction

This protocol is intended to provide a standard way for clients to sign HTTP requests. RFC 2617 [RFC2617] (HTTP Authentication) defines Basic and Digest authentication mechanisms, and RFC 5246 [RFC5246] (TLS 1.2) defines client-auth, both of which are widely employed on the Internet today. However, it is common place that the burdens of PKI prevent web service operators from deploying that methodoloy, and so many fall back to Basic authentication, which has poor security characteristics.

Additionally, OAuth provides a fully-specified alternative for authorization of web service requests, but is not (always) ideal for machine to machine communication, as the key acquisition steps (generally) imply a fixed infrastructure that may not make sense to a service provider (e.g., symmetric keys).

Several web service providers have invented their own schemes for signing HTTP requests, but to date, none have been placed in the public domain as a standard. This document serves that purpose. There are no techniques in this proposal that are novel beyond previous art, however, this aims to be a simple mechanism for signing these requests.

2. Signature Authentication Scheme

The "signature" authentication scheme is based on the model that the client must authenticate itself with a digital signature produced by either a private asymmetric key (e.g., RSA) or a shared symmetric key (e.g., HMAC). The scheme is parameterized enough such that it is not bound to any particular key type or signing algorithm. However, it does explicitly assume that clients can send an HTTP `Date` header.

2.1. Authorization Header

The client is expected to send an Authorization header (as defined in RFC 2617) with the following parameterization:

```
credentials := "Signature" SP params
params := keyId "," algorithm [", " headers] [", " ext] ", " signature

keyId := "keyId" "=" plain-string
algorithm := "algorithm" "=" DQUOTE ( rsa-sha1 / rsa-sha256 / rsa-sha512 /
                                      dsa-sha1 / hmac-sha1 / hmac-sha256 /
                                      hmac-sha512 )
                             DQUOTE
headers := "headers" "=" plain-string
ext := "ext" "=" plain-string
signature := "signature" "=" plain-string

plain-string   = DQUOTE *( %x20-21 / %x23-5B / %x5D-7E ) DQUOTE
```

2.1.1. Signature Parameters

The following section details the signature parameters of the Authorization Header.

2.1.1.1. keyId

REQUIRED. The `keyId` field is an opaque string that the server can use to look up the component they need to validate the signature. It could be an SSH key fingerprint, an LDAP DN, etc. Management of keys and assignment of `keyId` is out of scope for this document.

2.1.1.2. algorithm

REQUIRED. The `algorithm` parameter is used if the client and server agree on a non-standard digital signature algorithm. The full list of supported signature mechanisms is listed below.

2.1.1.3. headers

OPTIONAL. The `headers` parameter is used to specify the list of HTTP headers used to sign the request. If specified, it should be a quoted list of HTTP header names, separated by a single space character. By default, only one HTTP header is signed, which is the `Date` header. Note that the list MUST be specified in the order the values are concatenated together during signing. To include the HTTP request line in the signature calculation, use the special `request-line` value. While this is overloading the definition of `headers` in HTTP linguism, the request-line is defined in RFC 2616 [RFC2616], and as the outlier from headers in useful signature calculation, it is deemed simpler to use `request-line` than to add a separate parameter for it.

2.1.1.4. extensions

OPTIONAL. The `extensions` parameter is used to include additional information which is covered by the request. The content and format of the string is out of scope for this document, and expected to be specified by implementors.

2.1.1.5. signature

REQUIRED. The `signature` parameter is a `Base64` encoded digital signature generated by the client. The client uses the `algorithm` and `headers` request parameters to form a canonicalized `signing string`. This `signing string` is then signed with the key associated with `keyId` and the algorithm corresponding to `algorithm`. The `signature` parameter is then set to the `Base64` encoding of the signature.

2.1.2. Signature String Construction

In order to generate the string that is signed with a key, the client MUST take the values of each HTTP header specified by `headers` in the order they appear. It is out of scope for this document to dictate what headers a service provider will want to enforce, but service providers SHOULD at minimum include the request line, Host, and Date headers.

1. If the header name is not `request-line` then append the lowercased header name followed with an ASCII colon `:` and an ASCII space ` `.

2. If the header name is `request-line` then appened the HTTP request line, otherwise append the header value.

3. If value is not the last value then append an ASCII newline `\n`. The string MUST NOT include a trailing ASCII newline.

The rest of this section uses the following HTTP request as an example.

```
POST /foo HTTP/1.1
Host: example.org
Date: Tue, 07 Jun 2014 20:51:35 GMT
Content-Type: application/json
Digest: SHA-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=
Content-Length: 18

{"hello": "world"}
```

The following sections also assume that the "rsa-key-1" keyId refers to a private key known to the client and a public key known to the server. The "hmac-key-1" keyId refers to key known to the client and server.

2.1.2.1. RSA Example

The authorization header and signature would be generated as:

```
Authorization: Signature keyId="rsa-key-1",algorithm="rsa-sha256",
headers="request-line host date digest content-length",
signature="Base64(RSA-SHA256(signing string))"
```

The client would compose the signing string as:

```
POST /foo HTTP/1.1\n
host: example.org\n
date: Tue, 07 Jun 2014 20:51:35 GMT\n
digest: SHA-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=\n
content-length: 18
```

Note that the '\n' symbols above are included to demonstrate where the new line character should be inserted. There is no new line on the final line of the signing string.

For an RSA-based signature, the authorization header and signature would then be generated as:

```
Authorization: Signature keyId="rsa-key-1",algorithm="rsa-sha256",
headers="request-line host date digest content-length",
signature="Base64(RSA-SHA256(signing string))"
```

2.1.2.2. HMAC Example

For an HMAC-based signature without a list of headers specified, the authorization header and signature would be generated as:

```
Authorization: Signature keyId="hmac-key-1",algorithm="hmac-sha1",
headers="request-line host date digest content-length",
signature="Base64(HMAC-SHA1(signing string))"
```

The only difference between the RSA Example and the HMAC Example is the signature algorithm that is used. The client would compose the signing string in the same way as the RSA Example above:

```
POST /foo HTTP/1.1\n
host: example.org\n
date: Tue, 07 Jun 2014 20:51:35 GMT\n
digest: SHA-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=\n
content-length: 18
```

3. Appendix A: Security Considerations

There are a number of security considerations to take into account when implementing or utilizing this specification. A thorough security analysis of this protocol, including its strengths and weaknesses, can be found in Security Considerations for HTTP Signatures [1].

4. Appendix B: Extensions

This specification was designed to be simple, modular, and extensible. There are a number of other specifications that build on this one. For example, the HTTP Signature Nonces [2] specification details how to use HTTP Signatures over a non-secured channel like HTTP and the HTTP Signature Trailers [3] specification explains how to apply HTTP Signatures to streaming content. Developers that desire more functionality than this specification provides are urged to ensure that an extension specification doesn't already exist before implementing a proprietary extension.

5. Appendix C: Test Values

The following test data uses the following RSA 2048-bit keys, which we will refer to as `keyId=Test` in the following samples:

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDCFENGw33yGihy92pDjZQhl0C3
6rPJj+CvfSC8+q28hxA161QFNUd13wuCTUcq0Qd2qsBe/2hFyc2DCJJg0h1L78+6
Z4UMR7EOcpfdUE9Hf3m/hs+FUR45uBJeDK1HSFHD8bHKD6kv8FPGfJTotc+2xjJw
oYi+1hqp1fIekaxsyQIDAQAB
-----END PUBLIC KEY-----
```

```
-----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQDCFENGw33yGihy92pDjZQhl0C36rPJj+CvfSC8+q28hxA161QF
NUd13wuCTUcq0Qd2qsBe/2hFyc2DCJJg0h1L78+6Z4UMR7EOcpfdUE9Hf3m/hs+F
UR45uBJeDK1HSFHD8bHKD6kv8FPGfJTotc+2xjJwoYi+1hqp1fIekaxsyQIDAQAB
AoGBAJR8ZkCUvx5kzv+utdl7T5MnordT1TvoXXJGXK7ZZ+UuvMNUCdN2QPc4sBiA
QWvLw1cSKt5DsKZ8UETpYPy8pPYnnDEz2dDYiaew9+xEpubyeW2oH4Zx71wqBtOK
kqwrXa/pzdpiucRRjk6vE6YY7EBBs/g7uanVpGibOVAEsqH1AkEA7DkjVH28WDUg
f1nqvfn2Kj6CT7nIcE3jGJsZZ7zlZmBmHFDONMLUrXR/Zm3pR5m0tCmBqa5RK95u
412jt1dPIwJBANJT3v8pnkth48bQo/fKel6uEYyboRtA5/uHuHkZ6FQF7OUkGogc
mSJluOdc5t6hI1VsLn0QZEjQZMEOWr+wKSMCQQCC4kXJEsHAve77oP6HtG/IiEn7
kpyUXRNvFsDE0czpJJBvL/aRFUJxuRK91jhjC68sA7NsKMGg5OXb5I5Jj36xAkEA
gIT7aFOYBFwGgQAQkWNKLvySgKbAZRTeLBacpHMuQdl1DfdntvAyqpAZ0lY0RKmW
G6aFKaqQfOXKCyWoUiVknQJAXrlgySFci/2ueKlIE1QqIiLSZ8V8OlpFLRnb1pzI
7U1yQXnTAEFYM560yJlzUpOb1V4cScGd365tiSMvxLOvTA==
-----END RSA PRIVATE KEY-----
```

All examples use this request:

```
POST /foo?param=value&pet=dog HTTP/1.1
Host: example.com
Date: Thu, 05 Jan 2014 21:31:40 GMT
Content-Type: application/json
Digest: SHA-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=
Content-Length: 18

{"hello": "world"}
```

5.1. Default Test

If a list of headers is not included, the date is the only header that is signed by default. The string to sign would be:

```
date: Thu, 05 Jan 2014 21:31:40 GMT
```

The Authorization header would be:

```
Authorization: Signature keyId="Test",algorithm="rsa-sha256",
signature="ATp0r26dbMIxOopqw0OfABDT7CKMIoENumuruOtarj8n/97Q3htH
FYpH8yOSQk3Z5zh8UxUym6FYTb5+A0Nz3NRsXJibnYi7brE/4tx5But9kkFGzG+
xpUmimN4c3TMN7OFH//+r8hBf7BT9/GmHDUVZT2JzWGLZES2xDOUuMtA="
```

5.2. Basic Test

The minimum recommended data to sign is the request-line, host, and date. In this case, the string to sign would be:

```
POST /foo?param=value&pet=dog HTTP/1.1
host: example.com
```

```
date: Thu, 05 Jan 2014 21:31:40 GMT
```

The Authorization header would be:

```
Authorization: Signature keyId="Test",algorithm="rsa-sha256",
headers="request-line host date", signature="KcLSABBj/m3v2Dhxi
CKJmzYJvnx74tDO1SaURD8Dr8XpugN5wpy8iBVJtpkHUIp4qBYpzx2QvD16t8X
0BUMiKc53Age+baQFWwb2iYYJzvuUL+krrl/Q7H6fPBADBsHqEZ7IE8rR0Ys3l
b7J5A6VB9J/4yVTRiBcxTypW/mpr5w="
```

5.3. All Headers Test

A strong signature including all of the headers and a digest of the body of the HTTP request would result in the following signing string:

```
POST /foo?param=value&pet=dog HTTP/1.1
host: example.com
date: Thu, 05 Jan 2014 21:31:40 GMT
content-type: application/json
digest: SHA-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=
content-length: 18
```

The Authorization header would be:

```
Authorization: Signature keyId="Test",algorithm="rsa-sha256",
headers="request-line host date content-type digest content-length",
signature="jgSqYK0yKclIHfF9zdApVEbDp5eqj8C4i4X76pE+XHoxugXv7q
nVrGR+30bmBgtpR39I4utq17s9ghz/2QFVxlnToYAvbSVZJ9ulLd1HQBugO0j
Oyn9sXOtcN7uNHBjqNCqUsnt0sw/cJA6B6nJZpyNqNyAXKdxZZItOuhIs78w="
```

6. 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.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[RFC3339] Klyne, G., Ed. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.

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

[1] <https://web-payments.org/specs/source/http-signatures-audit/>

[2] <https://web-payments.org/specs/source/http-signature-nonces/>

[3] <https://web-payments.org/specs/source/http-signature-trailers/>

Authors' Addresses

Mark Cavage
Joyent
One Embarcadero Center
9th Floor
San Francisco, CA 94111
US

Phone: +1 415 400 0626
Email: mark.cavage@joyent.com
URI: http://www.joyent.com/

Manu Sporny
Digital Bazaar
1700 Kraft Drive
Suite 2408
Blacksburg, VA 24060
US

Phone: +1 540 961 4469
Email: msporny@digitalbazaar.com
URI: http://manu.sporny.org/