Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 5, 2013

M. Cavage
Joyent
M. Sporny
Digital Bazaar

May 4, 2013

HTTP Signatures
draft-cavage-http-signatures-00

Abstract

This document describes a way to add origin authentication, message integrity, and replay resistance to HTTP requests. It is intended to be used over the HTTPS protocol.

Status of this Memo

Copyright Notice

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 simply 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.

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 2011 20:51:35 GMT
Content-Type: application/json
Content-MD5: lCMsW4/JJy9vc6HjbraPzw==
Content-Length: 15

{"bar": "baz"}
```

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",signature="Base64(RSA-SHA256(signing string))"
```

The client would compose the signing string as:

```
date: Tue, 07 Jun 2011 20:51:35 GMT
```

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

```
Authorization: Signature keyId="rsa-key-1",algorithm="rsa-
sha256",headers="request-line date content-type content-
md5",signature="Base64(RSA-SHA256(signing string))"
```

The client would compose the signing string as (`+ "\n"` inserted for readability):

```
POST /foo HTTP/1.1 + "\n"
date: Tue, 07 Jun 2011 20:51:35 GMT + "\n"
content-type: application/json + "\n"
content-md5: lCMsW4/JJy9vc6HjbraPzw==
```

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",signature="Base64(HMAC-SHA1(signing string))"
```

The client would compose the signing string as:

```
date: Tue, 07 Jun 2011 20:51:35 GMT
```

3. Appendix A: Security Considerations

There are a number of security considerations to take into account when deploying HTTP Signatures.

3.1. Default Parameters

Note the default parameterization of the `Signature` scheme is only safe if all requests are carried over a secure transport (i.e., TLS).

Sending the default scheme over a non-secure transport will leave the request vulnerable to spoofing, tampering, replay/repudiaton, and integrity violations (if using the STRIDE threat-modeling methodology).

3.2. Insecure Transports

If sending the request over plain HTTP, service providers SHOULD require clients to sign ALL HTTP headers, and the `request-line`. Additionally, service providers SHOULD require `Content-MD5` calculations to be performed to ensure against any tampering from clients.

3.3. Nonces

Nonces are out of scope for this document simply because many service providers fail to implement them correctly, or do not adopt security specfiications because of the infrastructure complexity. Given the `header` parameterization, a service provider is fully enabled to add nonce semantics into this scheme by using something like an `x-request-nonce` header, and ensuring it is signed with the `Date` header.

ISSUE: This specification should probably explain exactly how to implement nonces for implementers that would like a fully vetted solution that protects against replay. This would be useful for implementers implementing HTTP signatures in a clear channel environment. Another consideration for nonces is the probability that multiple clients may share the same public key. In this instance, due to clock skew issues, it is possible that some clients may accidentally trigger replay protection by sending a date in the past. The balance that this spec attempts to achieve is a simple per-client, time-based counter. Thus, the nonce would need to include something like a UUID-based client identifier, plus an incredibly accurate UTC datetime-based nonce as described in RFC 3339 [RFC3339]. For example: "598ef3e8-98b0-435d-8ca3-fecefdd87568 2013-05-04 20:00:35.808785840+00:00"

3.4. Clock Skew

As the default scheme is to sign the `Date` header, service providers SHOULD protect against logged replay attacks by enforcing a clock skew. The server SHOULD be synchronized with NTP, and the recommendation in this specification is to allow 300s of clock skew (in either direction).

3.5. Required Headers to Sign

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 `Date` header.

4. Appendix B: 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-----
```


And all examples use this request:

```
POST /foo?param=value&pet=dog HTTP/1.1
Host: example.com
Date: Thu, 05 Jan 2012 21:31:40 GMT
Content-Type: application/json
Content-MD5: Sd/dVLAcvNLSq16eXua5uQ==
Content-Length: 18

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

4.1. Default Test

The string to sign would be:

```
date: Thu, 05 Jan 2012 21:31:40 GMT
```

The Authorization header would be:

```
Authorization: Signature keyId="Test",algorithm="rsa-
sha256",signature="JldXnt8W9t643M2Sce10gqCh/
+E7QIYLiI+bSjnFBGCti7s+mPPvOjVb72sbd1FjeOUwPTDpKbrQQORrm+xBYfAwCxF3LBSSzORvyJ5nRFCFxfJ3nlQD6Kdxh
W3C8qH5uhFTRwF4ruRjh+ENHWuovPgO/HGQ="
```

4.2. All Headers Test

Parameterized to include all headers, the string to sign would be (`+ "\n"` inserted for readability):

```
POST /foo?param=value&pet=dog HTTP/1.1 + "\n"
host: example.com + "\n"
date: Thu, 05 Jan 2012 21:31:40 GMT + "\n"
content-type: application/json + "\n"
content-md5: Sd/dVLAcvNLSq16eXua5uQ== + "\n"
content-length: 18
```

The Authorization header would be:

```
Authorization: Signature keyId="Test",algorithm="rsa-sha256",headers="request-
line host date content-type content-md5 content-length",signature="Gm7W/
r+e90REDpWytALMrft4MqZxCmslOTOvwJX17ViEBA5E65QqvWI0vIH3l/
vSsGiaMVmuUgzYsJLYMLcm5dGrv1+a+0fCoUdVKPZWHyImQEqpLkopVwqEH67LVECFBqFTAKlQgBn676zrfXQbb+b/
VebAsNUtvQMe6cTjnDY="
```

5. 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.

Authors' Addresses

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

Phone: +1 ??? ??? ????
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/