HTTP Digest Update

Rifaat Shekh-Yusef

IETF 87, HTTPAuth WG, Berlin

July 31, 2013

Overview

- **Algorithms Agility**
  - draft-ietf-httpauth-digest-update
  - Standards Track draft
- **I18N**
  - draft-ietf-httpauth-digest-encoding
  - Experimental draft

Algorithms Agility

Browsers Experiments

- **Experiment**
  - Multiple WWW-Authenticate headers in a response with the same scheme but different algorithms.
- **Chrome version 23**
  - Able to handle multiple Authenticate headers with the same scheme.
  - Gives preference to the header that appears first.
  - Ignores algorithms it does not understand, and picks the first algorithm it does understand.
- **IE version 9**
  - Able to handle multiple Authenticate headers with the same scheme.
  - Gives preference to the header that appears first.
  - Reverts back to use Basic scheme if it does not understand the algorithm in first Digest scheme.
- **For more info:**
  - http://www.ietf.org/mail-archive/web/http-auth/current/msg01171.html

MD5

- **MD5** is the only algorithm specified in RFC2617 to be used with the Digest Access Authentication scheme.
- In 2008 the US-CERT issued a note that MD5 "**should be considered cryptographically broken and unsuitable for further use**“.

Algorithm Parameter

- RFC2617 defines the following parameter to be used with the Authenticate header:
  - algorithm = "algorithm" "="

    ("MD5" | "MD5-sess" | **token**)

- The **token** defined above allows new documents the ability to extend the Digest scheme with new algorithms.

New Algorithms

- The Algorithm Agility document adds support for two new algorithms:
  - SHA2-256
  - SHA2-512/256

- The SHA2-512/256 is expected to be replaced by SHA3 when it is ready.

Algorithms Preference

- The draft defines the following preference list, starting with the most preferred algorithm:
  - **SHA2-256** as the **default** algorithm.
  - **SHA2-512/256** as a **backup** algorithm.
  - **MD5** for backward compatibility.

Multiple Authenticate Headers

- RFC2617 is not clear on the number of WWW-Authenticate or Proxy-Authenticate headers using the same scheme that are allowed in a response.
- This draft explicitly allows more that one WWW-Authenticate or Proxy-Authenticate headers using the same scheme but different algorithms to be included in a response.

WWW-Authenticate Example

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Digest
  realm = "testrealm@host.com",
  qop="auth,
  auth-int",
  algorithm="SHA2-256",
  nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
  opaque="5ccc069c403ebaf9f0171e9517f40e41"
WWW-Authenticate: Digest
  realm="testrealm@host.com",
  qop="auth,
  auth-int",
  algorithm="MD5",
  nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
  opaque="5ccc069c403ebaf9f0171e9517f40ef41"
```

Authorization Example

```
Authorization: Digest
   username="Mufasa",
   realm="testrealm@host.com",
   nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
   uri="/dir/index.html",
   qop="auth",
   algorithm="SHA2-256",
   nc=00000001,
   cnonce="0a4f113b",
   response="5abdd07184ba512a22c53f41470e5
         eea7dcaa3a93a59b630c13dfe0a5dc6e38b",
   opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

Open Issue

- There is some concern around the level of support for the SHA2-512/256 algorithm in the common implementation of SHA2.

- Should we keep SHA2-512/256 and replace it with SHA3 later on?
- Should we choose a different algorithm as backup algorithm?
- Should we not specify any backup algorithm?

I18N

ASCII Encoding

- RFC2617 defines a way to concatenate **username-value**, **realm-value**, and **password** as part of the **A1** calculations. (see section 3.2.2.2).
- That concatenation assumes that **ASCII** is used and does not define how to indicate the desire to use **Unicode** characters outside the **ASCII** range.

The "auth-param"

- RFC2617 defines the following parameter to be used with the WWW-Authenticate and Authorization headers:
  - **auth-param**

    This directive allows for future extensions. Any unrecognized directive MUST be ignored.

- The above **auth-param** allows new parameters to be defined and added to the header.

The "charset" Parameter

- This document defines the **"charset"** parameter to be used to indicate the encoding used by the side that adds it to the header.

- The only allowed value is **“UTF-8"**.

Server Behavior

- Send "charset" parameter in a challenge
- Look for "charset" parameter in a subsequent request:
  - "charset" present
    - If it has the same value, continue normal operation; otherwise immediately decline the request.
  - "charset" absent
    - This is an indication that the browser does not support this specification; continue with the current normal operation.

Client Behavior

- Browser adds the "charset" parameter to the subsequent request:
  - Using value it received from the server, if it supports the encoding.
  - Using the value it received from the server but preceded by !, if it does not support the encoding.
- Browsers that do not support this specification will ignore the "charset" parameter.

Open Issue

- We would like to get more feedback from the community around this approach.
- We would like to understand what the various browser vendors are doing, and if this approach is aligned with their implementation.

Questions?