# HTTP Mutual auth

Yutaka OIWA

HTTPAUTH, IETF 87

# Design Goal

- Strongest-possible HTTP authentication based on a single ID/password pair
  - Replacement for Basic/Digest
  - Simple to use
  - No additional devices
  - No client-stored data (e.g. PKI keys)

# Provided Features (1)

- Strong password authentication
- Mutual server/client authentication
  - Authentication status mutually agreed
  - The client will know whether the server knows him/her account, or just “lying”
- Per-server/per-domain authentication credentials
  - Authentication credentials localized to domains or hosts
  - Mitigation for stolen server DBs and/or malicious administrators

# Provided Features (2)

- Channel bindings
  - To lower layers: HTTPS (TLS certficate) and plain HTTP (hostname)
    - Against man-in-the-middle attacks
  - To higher layers: providing secure shared keys
    - Application interface provided
- Efficient re-authentication
  - Important for HTTP-based protocol uses
  - With protections for replay attack
  - Works good with pipelining, multiple connections, and HTTP/2.0 multiplexing

# How it works?

- Using PAKE (a.k.a. ZKPP) as a tool
- Adopted for HTTP 1.1 (and 2.0)
  1. PAKE key exchange using secrets from the same password
  2. Use a hash to verify its correctness *mutually*
  3. For re-authentication, only hashes are used (like nonce cache in Digest)
  4. Session keys can be discarded at any time

# Design Status

- Completed -- “working status”
  - Initial 4-message authentication
  - 2-message fast re-authentication
  - Mandatory server-to-client authentication using “Authentication-Info” header
  - Cryptographic primitive agility
  - Efficient and secure nonce management
    - Duplicated nonce detection is MUST
    - Implementable in a (small) constant memory per session, in both server/client sides

# Implementations

- Working codes!
  - Server side
    - Apache module
    - Ruby/Webrick reference implementation
  - Client side
    - Ruby library reference implementation
    - Customized Mozilla Firefox (old 3.6)
    - Chromium (recent one) – almost done!

    - Status: published / to appear / now working on

# Security (1)

- Strength against various attacks
  - Traffic eavesdropping (passive)
    - No password information leaked
    - Even off-line dictionary attack impossible
  - Traffic rewriting (network-level)
    - No password information leaked
    - No replay attack possible
      - Thanks to strong shared keys and duplicate nonce checking
    - Request/result will be rewritten:
      for integrity/confidentiality, use HTTPS/TLS

# Security (2)

- Traffic forwarding (URL-level attack)
  - User has input the password of good.example.com to bad.example.net – what will happen?
    - *(Natural) assumption: bad.example.net does not know the exact password*
  - Authentication will *always fail*
    - Forwarding traffics to good site will not work
    - Bad site can't build forged successful result
  - No password information leaked to bad site
    - A valuable property even when HTTPS is used

# Security (3)

- Server database leakage
  - Stored credentials are “hashed”
    - Not the password equivalent (compare with Digest)
    - Salted by fixed data (domain name and user name)
    - Stored credentials are bound to each site/domain
  - Much safer than hash-based “Digest” algorithms
    - If passwords are strong enough against dictionary attacks, security will not be broken
  - One strong password can be safely used with multiple sites (technically)
    - not ethically recommended, of course

# Open Issues (1)

- Standard interfaces to higher layers
  - We provide a key-provision facility
  - How to standardize its use?
    - Session continuation
    - Oauth MAC etc.
    - Content-body signature/authentication
    - Web application-layer key managements
  - How to share it among proposals?
    - Draft-oiwa-httpauth-multihop-template is a straw-man proposal

# Open Issues (2)

- Document Structure
  - Currently, crypto part is a separate draft
    - draft-oiwa-httpauth-mutual-algo (individual submission)
  - Separation was a Bar-BoF request
    - Provision for separate crypto discussion/reviews
  - Now, the situation has changed so much
    - WG has been formed
    - Area changed: Application → Security
    - Intended status changed: Std → Exp
  - What to do?
    - Merge it again? Or Promote the algo draft to WG draft?
      - I need a new consensus to follow

# HTTP Auth Extensions

Yutaka OIWA

HTTPAUTH, IETF 87

# What's this?

- General user-experience extensions of HTTP authentication for interactive clients
  - i.e. Web browsers
  - Not changing low-level behavior of HTTP authentication
    - Thus, not applicable for simple HTTP clients
- Independent from authentication schemas
  - Applicable for all interactive HTTP authentication schemes

# Features

- Browser hints for authentication-related behaviors
  - What to do if authentication does not occur?
    - but do re-authentication if password already known
    - Redirect, instead of asking password
    - Do not ask for new authentication on this URI
  - What to do when user wanted to log out?
    - Redirect to a special “log-out” page
  - How long should authentication retained?
    - Time-out for inactive authentication sessions

# Technical Design

- New header “Authentication-Control”
  - For backward compatibility
  - For scheme independence
  - For simple use
- One carefully-designed point: simple use for simple use cases!
  - Setting the header globally will work
    - in .htaccess or apache.conf etc.
    - No additional modules/CGIs needed

# Status

- Deployable
  - Working code in Firefox-based Mutual-auth implementation
    - For server side, no new code required at all
- Some refinement/polishs may be good
  - Shorten keywords – received a feedback
  - Feature requests/completeness analysis?
  - Detailed semantics to be defined -- especially regarding POST requests

# Back-up slides (for Mutual auth)

# Security (4)

- Phishing: preventing it as much as technologically possible
  - The user's password and other secrets will be safe, even when the user talks with bad site; under some assumptions:
    - Browsers will tell users whether Mutual is used or not
    - Users will not send passwords in other protocols
      - by Basic, Web Forms, Digest, phone, papers etc.
    - Browsers will always tell mutual authentication result
      - Required in the specification
    - Users will not proceed when authentication has failed
    - TLS correctly used for payload body safety
      - But not relying on user's careful checking of URL/subject