| | | |
|---|---|---|
| Network Working Group | T. Broyer | |
| Internet-Draft | January 05, 2009 | |
| Intended status: Standards Track | | |
| Expires: July 9, 2009 | | |

TOC

**Cookie-based HTTP Authentication**
**draft-broyer-http-cookie-auth-00**

## Status of this Memo



## Copyright Notice



## Abstract

This document specifies an HTTP authentication scheme for use when credentials are validated by an out-of-band mechanism (not defined here) and later communicated to the server through the use of a cookie. Which out-of-band mechanism should be used, and how, is described by the 401 (Unauthorized) response body. It is common practice that this mechanism is an HTML form, sending the user's credentials with the use of an HTTP

POST request to a tier URL which will set a cookie in response; though this document doesn't preclude the use of other mechanisms.

**Editorial Note (To be removed by RFC Editor before publication)**

Distribution of this document is unlimited. Please send comments to the ietf-http-auth mailing list at ietf-http-auth@osafoundation.org, which may be joined by sending a message with subject "subscribe" to ietf-http-auth-request@osafoundation.org.
Discussions of the ietf-http-auth mailing list are archived at http://lists.osafoundation.org/pipermail/ietf-http-auth/.
XML versions, latest edits and the issues list for this document are available from http://broyer.info/hg/http-cookie-auth/.

---

**Table of Contents**



---

## 1. Introduction

TOC

Authentication on the web can be done either at the Hypertext Transfer Protocol (HTTP) (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” June 1999.) [RFC2616] level with a 401 (Unauthorized) status code, or using SSL certificates. Among other issues already listed in User Agent Authentication Forms (Lawrence, S. and P. Leach, “User Agent Authentication Forms,” February 1999.) [W3C.NOTE-authentform-19990203], the former suffers from a poor user experience while the latter can quickly become expensive. That's why the most common authentication mechanism is based on HyperText Markup Language (HTML) forms (Jacobs, I., Hors, A., and D. Raggett, “HTML 4.01

Specification," December 1999.) [W3C.REC-html401-19991224] and cookies (Kristol, D. and L. Montulli, "HTTP State Management Mechanism," October 2000.) [RFC2965].
However, form-based authentication is almost always implemented with an HTTP redirect to the login form, making it impossible for non-browser user agents to detect a protected resource (this leads to people downloading and saving login forms instead of the protected resource they wanted, web service clients failing with unrecoverable errors, etc.).
User Agent Authentication Forms (Lawrence, S. and P. Leach, "User Agent Authentication Forms," February 1999.) [W3C.NOTE-authentform-19990203] tried to overcome this with an amendment to HTML forms making them "HTTP-authentication aware".
This document solves the problem the other way around, keeping the mechanism backwards compatible with browsers while making it independant of HTML.

---

## 2. Notational Conventions

TOC

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 Appendix of [RFC2119] (Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," March 1997.).
The terminology used here follows and extends that in the HTTP specification Appendix of [RFC2616] (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," June 1999.).

---

## 3. Cookie Authentication Scheme

TOC

The "cookie" authentication scheme tries to reconcile the current practice of many web sites and web development frameworks of using HTML forms and cookies to authenticate users, and the Access Authentication Framework described in Section 1.2 of [RFC2617] (Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication," June 1999.). The user credentials being passed through cookies, the Authorization and Proxy-Authorization request headers are therefore not used.
The "cookie" authentication scheme cannot be used for proxy authentication (within the value of a Proxy-Authenticate response header) because, as defined in Section 3.5 of [RFC2965] (Kristol, D. and L. Montulli, "HTTP State Management Mechanism," October 2000.): "Proxies

MUST NOT introduce Set-Cookie2 (Cookie) headers of their own in proxy responses (requests)."

When the origin server sends a 401 (Unauthorized) response containing a WWW-Authenticate header with a "cookie" authentication scheme, the response body gives instructions on how to create the appropriate cookies, generally by issuing another HTTP request (preferably a POST request) to a distinct URL.

In most current web sites and web applications, the response body would be an HTML document containing a form; when the form is submitted, the server checks the user-provided form-data and upon validation sends the appropriate Set-Cookie2 response header fields within a 303 (See Other) response redirecting back to the protected resource.

The "cookie" authentication scheme is however not limited to such scenarios: the response body could be for example an SVG image with an embedded XForms, or an HTML document with an embedded script that will compute a hash of user-provided data and set the cookie by script before reloading the resource, or some specific entity recognized by the UA, which will authenticate using an out-of-band mechanism and set the appropriate cookie before re-requesting the protected resource. This last scenario might be better solved using another authentication scheme, though this scenario would allow server-side negotiation of the authentication mechanism using content negotiation; instead of the client-side negotiation traditionally used when sending multiple WWW-Authenticate response headers.

Syntax (using the augmented Backus-Naur Form (BNF) defined in Section 2.1 of [RFC2616] (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” June 1999.)):

```
challenge           =  "Cookie" cookie-challenge

cookie-challenge    = 1#( realm | [ form-action ] | cookie-name |
                      [ secure-cookie-name ] | [auth-param] )

form-action         = "form-action" "=" <"> URI <">
URI                 = absolute-URI | ( path-absolute [ "?" query ] )
cookie-name         = "cookie-name" "=" token
secure-cookie-name  = "secure-cookie-name" "=" token

path-absolute  = <defined in [RFC3986], Section 3.3>
quoted-string  = <defined in [RFC2616], Section 2.2>
query          = <defined in [RFC3986], Section 3.4>
token          = <defined in [RFC2616], Section 2.2>
```

The meanings of the values of the directives used above are as follows:

> **form-action** OPTIONAL. The value of the "form-action" attribute is the URI reference of the resource that will set the cookies used for authenticating the user in subsequent requests. The value

must resolve to an URI reference where the "scheme" part MUST be "http" or "https", the "authority" part contains no "userinfo", the "host" and "abs_path" parts have the same contraints as the "Domain" and "Path" attributes of a Set-Cookie2 response header respectively.

**cookie-name** REQUIRED. The value of the "cookie-name" attribute is the name of the cookie that is checked by the server to authenticate the user; an UA thus could then inform the user this cookie is necessary to gain access to the protected resource, and eventually use a different, more secure, storage than for other cookies.

**secure-cookie-name** OPTIONAL. In case the application uses a mix of secured and unsecured channels, the value of the "secure-cookie-name" attribute is the name of the cookie that is checked by the server to authenticate the user when the communication uses a secured channel, while the cookie named by the "cookie-name" attribute will be used for unsecured channel.

The applicability of the cookie(s) (its Domain, Port and Path attributes) defines the protection space.

---

## 4. Acknowledgements

TOC

---

## 5. IANA Considerations

TOC

This memo includes no request to IANA.

---

## 6. Security Considerations

TOC

As with any use of cookies, care should be taken by servers to avoid cookie spoofing, and clients to prevent unexpected cookie sharing (see Section 6 and Section 7 of [RFC2965] (Kristol, D. and L. Montulli, “HTTP State Management Mechanism,” October 2000.)).
However, using cookies for account information requires that some additional measures be taken. Using HTTP Over TLS (Rescorla, E., “HTTP Over TLS,” May 2000.) [RFC2818] or other means of encrypting the conversation is sufficient to mitigate most threats, though it requires that some additional measures be taken, as described in this section.

To mitigate replay attacks (re-use of a sniffed cookie), the value of the cookie used for authentication SHOULD NOT contain the users credentials but rather a key associated with the authentication session, and this key SHOULD be renewed (and expired) frequently.
Sensitive information (such as the user's IBAN on an online store) and sensitive actions (such as confirming an order) SHOULD only happen on a secure channel such as HTTP Over TLS (Rescorla, E., “HTTP Over TLS,” May 2000.) [RFC2818], and protected with a secure cookie (a cookie with the "Secure" bit set) so that it cannot be stolen on a unsecured channel.
This document does not specify how credentials are sent to the "form-action" URL, though care should be taken that those credentials cannot be sniffed. In the case of an HTML form, the "form-action" SHOULD use a secure channel such as HTTP Over TLS (Rescorla, E., “HTTP Over TLS,” May 2000.) [RFC2818].
[anchor2] (TODO: document how secure-cookie-name helps with security by preventing replay-attacks. The cookie must obviously have the Secure attribute set.)
[anchor3] (TODO: add some words about CSRF (and find a normative reference). Mention "logout" as a mean to mitigate CSRF.)

---

## 7. References

TOC

---

### 7.1. Normative References

TOC


| | |
|---|---|
| [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 (TXT, HTML, XML). |
| [RFC2818] | Rescorla, E., “HTTP Over TLS,” RFC 2818, May 2000 (TXT). |
| [RFC2965] | Kristol, D. and L. Montulli, “HTTP State Management Mechanism,” RFC 2965, October 2000 (TXT, HTML, XML). |
| [RFC3986] | Berners-Lee, T., Fielding, R., and L. Masinter, “Uniform Resource Identifier (URI): Generic Syntax,” RFC 3986, January 2005. |


---

### 7.2. Informative References

TOC

| | |
|---|---|
| [W3C.NOTE-authentform-19990203] | Lawrence, S. and P. Leach, “User Agent Authentication Forms,” W3C NOTE NOTE-authentform-19990203, February 1999. |
| [W3C.REC-html401-19991224] | Jacobs, I., Hors, A., and D. Raggett, “HTML 4.01 Specification,” World Wide Web Consortium Recommendation REC-html401-19991224, December 1999 (HTML). |

---

## Appendix A. Examples

TOC

Most detail of request and response headers has been omitted. Assume that the user agent has no stored cookies.

---

### A.1. Simple example (everything goes through TLS)

TOC

1. User Agent -> Server

```
GET https://www.example.com/acme/ HTTP/1.1
```

2. Server -> User Agent

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Cookie realm="Acme"
        form-action="/acme/login"
        cookie-name=ACME_TICKET
Content-Type: text/html

<title>Unauthorized</title>
<form action="/acme/login" method=POST>
<input type=hidden name=referer value="/acme/">
<p><label>Username: <input name=user></label>
<p><label>Password: <input name=pwd type=password></label>
<p><button type=submit>Sign in</button>
<p><a href="/acme/register">Register for an account</a>
</form>
```

3. User Agent -> Server

```
        POST https://www.example.com/acme/login HTTP/1.1
        Content-Type: application/x-www-form-urlencoded

        referer=%2Facme%2F&user=Aladdin&password=open%20sesame
```

4. Server -> User Agent

```
        HTTP/1.1 303 See Other
        Location: https://www.example.com/acme/
        Set-Cookie2: ACME_TICKET="sdf354s5c1s8e1s"; Version="1";
                Path="/acme"; Secure
```

5. User Agent -> Server

```
        GET https://www.example.com/acme/ HTTP/1.1
        Cookie: $Version="1"; ACME_TICKET="sdf354s5c1s8e1s"; $Path="/acme"
```

6. Server -> User Agent

```
        HTTP/1.1 200 OK
```

---

### A.2. Mixed HTTP/HTTPS example

TOC

1. User Agent -> Server

```
        GET http://www.example.com/acme/ HTTP/1.1
```

2. Server -> User Agent

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Cookie realm="Acme"
        form-action="https://secure.example.com/acme/login"
        cookie-name=ACME_TICKET
        secure-cookie-name=ACME_SECURE_TICKET
Content-Type: text/html

<title>Unauthorized</title>
<form action="https://secure.example.com/acme/login" method=POST>
<input type=hidden name=referer
       value="http://www.example.com/acme/">
<p><label>Username: <input name=user></label>
<p><label>Password: <input name=pwd type=password></label>
<p><button type=submit>Sign in</button>
<p><a href="/acme/register">Register for an account</a>
</form>
```

3. User Agent -> Server

```
POST https://secure.example.com/acme/login HTTP/1.1
Content-Type: application/x-www-form-urlencoded

referer=http%3A%2F%2Fwww.example.com%2Facme%2F&user=Aladdin&password=open%20sesa
```

4. Server -> User Agent

```
HTTP/1.1 303 See Other
Location: http://www.example.com/acme/
Set-Cookie2: ACME_TICKET="sdf354s5c1s8e1s"; Version="1";
        Path="/acme"; Domain=".example.com"
Set-Cookie2: ACME_SECURE_TICKET="drg53d51fd535rg"; Version="1";
        Path="/acme"; Domain=".example.com"; Secure
```

5. User Agent -> Server

```
GET http://www.example.com/acme/ HTTP/1.1
Cookie: $Version="1"; ACME_TICKET="sdf354s5c1s8e1s";
        $Path="/acme"; $Domain=".example.com"
```

6. Server -> User Agent

```
HTTP/1.1 200 OK
```

7. User Agent -> Server

```
GET https://secure.example.com/acme/ HTTP/1.1
Cookie: $Version="1"; ACME_SECURE_TICKET="drg53d51fd535rg";
        $Path="/acme"; $Domain=".example.com"
```

8. Server -> User Agent

```
HTTP/1.1 200 OK
```

---

### A.3. Cross-domain example

TOC

[anchor9] (TODO: using CSRF and server-to-server communication to achieve cross-domain single sign-on between sso.some-co.com and www.some-tm.net.)

At some-tm.net, the 401 response body loads a javascript from sso.some-co.com that sets a "temporary" cookie if already authenticated or redirects to sso.some-co.com otherwise. In the former case, the server validates the "temporary" cookie by calling sso.some-co.com and then sets the appropriate cookie to authenticate the user at some-tm.net. On the latter case, the server then redirects the browser back to some-tm.net with some token in the URL; this token is validated the same way as with the "temporary" cookie and the browser is then redirected back to the protected resource.

Fallback in case javascript is not available is a <meta refresh> (in a <noscript>) to redirect the browser to sso.some-co.com. The process is then similar to JA-SIG Central Authentication Service (CAS).

Or maybe these should be two distinct examples?

And do not forget the "single-logout" issue.

---

### Author's Address

TOC


| | |
|---|---|
| | Thomas Broyer |
| Email: | t.broyer@ltgt.net |