Workgroup: httpapi
Internet-Draft:
draft-schwartz-httpapi-popup-authentication-00
Published: 17 October 2022
Intended Status: Standards Track
Expires: 20 April 2023
Authors: B. M. Schwartz
Google LLC

Interactive Authentication of Non-Interactive HTTP Requests

Abstract

On the World Wide Web, a rich ecosystem of authentication options has been developed to support access control for HTTP resources. However, non-interactive usage of HTTP remains limited to the simple authentication mechanisms defined in the HTTP standards. This specification allows non-interactive HTTP contexts to open a browser-like authentication context when necessary, and close it when authentication is complete.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-schwartz-httpapi-popup-authentication/.

Source for this draft and an issue tracker can be found at https://github.com/bemasc/access-services.

Status of This Memo

Copyright Notice

Table of Contents

1. Background

In technical systems today, we can divide usage of HTTP into two categories. The first category is represented by the World Wide Web, where browsers load HTML files and their subresources for display to the user, in response to user actions. We call this category of usage "interactive".

The second category of usage consists of requests whose results are not presented interactively to the user ("non-interactive"). Instead these HTTP requests are used to perform operations needed by a software system such as a browser, application, or operating system. These requests are generally not for HTML content, and are often entirely invisible to the user. Even if the request is user-initiated, it does not normally present the user with a browser window.

In interactive usage, HTTP offers a variety of authentication options. A simple option is to use HTTP's built-in password challenge capabilities (carried in Basic or Digest authentication headers), but this pattern is generally regarded as obsolete on the web today. Instead, user authentication relies on account information entered via HTML forms, session cookies to retain login state, and new device attestation systems like WebAuthn. Third-party account providers and server-to-server OAuth2 are also widely used to simplify account management.

In non-interactive usage, the only available generic HTTP authentication mechanism is the built-in password challenge. In this mode, the HTTP server responds with a WWW-Authenticate header requesting Basic or Digest authentication. If the client already knows a username and password, it can provide those; otherwise, it might display a login prompt, with an explanation of what subsystem is requesting these credentials and why.

Client-to-server OAuth2 is commonly used for authentication of non-interactive HTTP clients, but it is concerned exclusively with client software that is already registered with a specific service. This specification aims to define an authentication pattern that allows interactive authentication of non-interactive HTTP requests between any participating client and server, without any private arrangement.

2. Overview

This specification enables a new mode of authentication for non-interactive HTTP requests. In this mode, the non-interactive request temporarily becomes interactive, enabling web-like authentication patterns. The process is as follows:

1. The user enters a URL into a configuration field in their system. This could be a field for specifying the URL of a proxy configuration file, software update server, or any other remote resource that is understood by the system.

2. Either immediately or at a later time, the system attempts to access this resource.

3. The server sends a response that means "interactive authentication required".

4. The system opens a browser-like window, showing HTML content provided by the server.

5. The user interacts with the HTML content in that window, potentially navigating between origins.

6. Eventually, a response from the server indicates that interactive authentication is complete.

7. The system closes the browser window and repeats the initial request with additional authentication headers. The request is authorized and succeeds.

8. Subsequent requests retain the authentication state, and succeed as non-interactive requests.

9. Eventually, the authentication state may expire, in which case the server requests interactive authentication again.

3. Conventions and Definitions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

4. Specification

We presume the existence of a desired resource, identified by the "initial URL". This resource **MAY** require client authentication. If client authentication is not required, this specification is not relevant, and access to the resource proceeds as usual.

4.1. Server requirements

If the request requires authentication, the server **SHALL** return HTTP 401 "Unauthorized" with a "WWW-Authenticate" header whose auth-scheme is "interactive" (registered in Section 8). This header field value **MUST** also contain a parameter named "location" whose value is a URL path (the "authentication path"). The server **MAY** also include other "WWW-Authenticate" headers indicating other supported authentication schemes.

Any GET request to the authentication path (on the same origin as the initial URL) **MUST** be subject to the same authentication requirements as the rejected request to the initial URL. Note: the rejected request may have used a different method, such as POST, that might have different authentication requirements than a GET request to the initial URL.

A GET request to the authentication path that fails authentication **MUST** return a webpage that guides the user through the authentication process. This process **MUST** conclude by causing the client to repeat the GET request to the authentication path,

returning a 2XX response code (as the client is now including the necessary authentication headers).

4.2. Client requirements

If the client receives an HTTP 401 "Unauthorized" error with a "WWW-Authenticate" header whose auth-scheme is "interactive", it **SHOULD** notify the user that the initial URL's origin is requesting interactive authentication, including a reminder of the role for which this origin is being used noninteractively. With the user's approval, it **SHOULD** load the authentication path from the "location" parameter as a webpage in a browser context. This context **SHOULD** have access to the user's credential assistance functions (e.g. password manager) but **MAY** otherwise be a blank context.

This browser **MUST** behave similarly to a normal browser, including support for navigation between origins. It **SHOULD** display the current origin to the user, to reduce the risk of impersonation attacks.

The client **MUST** monitor any requests made by the browser to the authentication path (whether as navigation, subresource, or javascript-initiated fetch). If any such request succeeds (i.e. receives a 2XX status code), the client **MUST** (1) store any "Authorization" and "Cookie" headers used in this request and (2) close this browser instance. The client **SHOULD** also display a notification that interactive authentication has concluded.

After learning the authorization headers, the client **SHOULD** retry the failed request if it is still relevant. For this and all subsequent requests to the initial URL, the client **MUST** add the stored "Authorization" and "Cookie" headers.

If the user closes the browser instance without successfully retrieving the resource at the authentication path, the system **SHOULD** warn the user that authentication has failed. The system **SHOULD** avoid spamming the user with repeated authentication requests, but **SHOULD NOT** permanently abandon authentication.

Web browsers **MUST NOT** implement support for the "interactive" auth-scheme in ordinary usage. This auth-scheme is not meaningful in an interactive context.

4.3. Use with proxy servers

If the "initial URL" indicates a proxy server, this procedure applies with the following modifications:

- When authenticating requests to the proxy:
  - The "Proxy-Authorization" header field is used instead of "Authorization".
  - The "Cookie" header field is not added.
- In replies from the proxy:
  - The HTTP 407 "Proxy Authentication Required" status code is used instead of HTTP 401.
  - The "Proxy-Authenticate" header field is used instead of "WWW-Authenticate".

5. Example

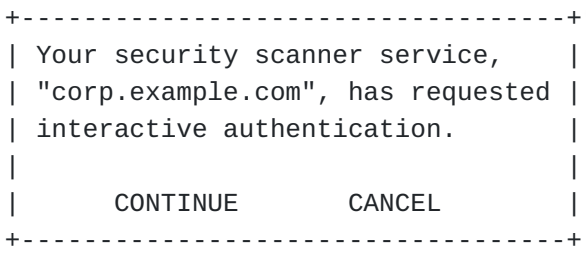
Suppose that the user has entered an initial URL of "https://corp.example.com/scan" into a settings panel on their system labeled "Executable Security Scanner URL". Later, when the user is installing a new executable, the system attempts to upload it to the security scanner service:

```
POST /scan HTTP/1.1
Host: corp.example.com
Accept: application/json
Content-Type: application/x-msdownload
Content-Length: 123456
...
```

The security scanner is access-controlled by interactive authentication, so it sends the following reply:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: interactive location=/scanner-login
...
```

The client displays a notification to the user:



The user approves, and the client loads "https://corp.example.com/scanner-login" in a browser context:

```
GET /scanner-login HTTP/1.1
Host: corp.example.com
Accept: text/html,...
Accept-Language: en-US,...
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: none
Sec-Fetch-User: ?1
...
```

This request is still unauthorized, so the server replies with HTTP 401 again:

```
HTTP/1.1 401 Unauthorized
Content-Type: text/html
...
```

The content of the HTTP 401 response is a login page. The user logs in, perhaps via third-party OAuth or using WebAuthn. Once login is complete, the final step navigates back to the authorization path. This time, the request includes an additional Cookie header:

```
GET /scanner-login HTTP/1.1
Host: corp.example.com
Accept: text/html,...
Accept-Language: en-US,...
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Cookie: login=6bb0e2c8-874e-44c8-b8e0-25e12f339b46
...
```

```
HTTP/1.1 200 OK
Content-Type: text/html
...
```

   The client detects this response and closes the browser context.
   Instead, it displays a notification:

```
+---------------------------------+
| You have successfully logged in |
| to "corp.example.com".          |
|                                 |
|               OK                |
+---------------------------------+
```

   The client then retries the initial request, with the additional
   Cookie header:

```
POST /scan HTTP/1.1
Host: corp.example.com
Accept: application/json
Content-Type: application/x-msdownload
Content-Length: 123456
Cookie: login=6bb0e2c8-874e-44c8-b8e0-25e12f339b46
...
```

   The server accepts the cookie as authorization and replies with its
   scan results:

```
HTTP/1.1 200 OK
Content-Type: application/json
...

{"scan_result": "safe"}
```

6. Security Considerations

   This specification grants noninteractive HTTP origins the ability to
   become interactive, surfacing arbitrary content to the user. This
   raises a number of security concerns.

   One important concern is "domain impersonation", in which the
   initial URL's origin poses as a different origin, in order to trick
   the user into revealing their password or taking some other harmful
   action. This is mitigated by displaying the current origin's

hostname to the user in the browser context (as normally done by browsers and recommended in Section 4.2).

Another concern is related to "clickjacking" attacks, in which a hostile origin causes a user to interact with the wrong user interface. For example, if the hostile origin places an "OK" button at the expected location of a system security setting, the origin might be able to close the browser window just before the user clicks, causing them to change the security setting instead. Clickjacking is prevented by the interstitial notifications when entering and exiting interactive mode (recommended in Section 4.2).

Web browsers also offer an expanded attack surface related to software vulnerabilities. If the "initial URL" has significant potential to be malicious, and an up-to-date web browser is not available, this specification may not be appropriate to implement.

7. Privacy Considerations

Authenticating noninteractive requests also makes them more identifiable and linkable. Standards developers should consider whether authentication is necessary and appropriate before incorporating this procedure into their standard.

> TODO: Language on clearing cookies. If the authentication is allowed to use an ephemeral browser context, what does it mean to clear cookies?

8. IANA Considerations

IF APPROVED, IANA is requested to add the following entry to the "HTTP Authentication Schemes" registry:

*Authentication Scheme Name: "interactive"

*Reference: (This document)

9. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

Acknowledgments

TODO acknowledge.

Author's Address

Benjamin M. Schwartz
Google LLC

Email: bemasc@google.com