                            Captive Portal API
                        draft-ietf-capport-api-03

Abstract

   This document describes an HTTP API that allows clients to interact
   with a Captive Portal system.

Status of This Memo

   This Internet-Draft is submitted in full conformance with the
   provisions of BCP 78 and BCP 79.

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF). Note that other groups may also distribute
   working documents as Internet-Drafts. The list of current Internet-
   Drafts is at https://datatracker.ietf.org/drafts/current/.

   Internet-Drafts are draft documents valid for a maximum of six months
   and may be updated, replaced, or obsoleted by other documents at any
   time. It is inappropriate to use Internet-Drafts as reference
   material or to cite them other than as "work in progress."

   This Internet-Draft will expire on January 7, 2020.

Table of Contents

1. Introduction

   This document describes a HyperText Transfer Protocol (HTTP)
   Application Program Interface (API) that allows clients to interact
   with a Captive Portal system. The API defined in this document has
   been designed to meet the requirements in the Captive Portal
   Architecture [I-D.ietf-capport-architecture]. Specifically, the API
   provides:

   o The state of captivity (whether or not the client has access to
      the Internet)

   o A URI that a client browser can present to a user to get out of
      captivity

   o An encrypted connection (TLS for both the API and portal URI)

2. Terminology

   This document leverages the terminology and components described in
   [I-D.ietf-capport-architecture] and additionally uses the following
   association:

   o Captive Portal Client: The client that interacts with the Captive
      Portal API is typically some application running on the User
      Equipment that is connected to the Captive Network. This is also
      referred to as the "client" in this document.

o Captive Portal API Server: The server exposing the API's defined
   in this document to the client. This is also referred to as the
   "API server" in this document.

3. Workflow

The Captive Portal Architecture defines several categories of
interaction between clients and Captive Portal systems:

1. Provisioning, in which a client discovers that a network has a
    captive portal, and learns the URI of the API server.

2. API Server interaction, in which a client queries the state of
    the captive portal and retrieves the necessary information to get
    out of captivity.

3. Enforcement, in which the enforcement device in the network
    blocks disallowed traffic.

This document defines the mechanisms used in the second category. It
is assumed that the location of the Captive Portal API server has
been discovered by the client as part of Provisioning. The mechanism
for discovering the API Server endpoint is not covered by this
document.

4. API Details

4.1. URI of Captive Portal API endpoint

The URI of the API endpoint MUST be accessed using HTTP over TLS
(HTTPS) and SHOULD be served on port 443 [RFC2818]. The client
SHOULD NOT assume that the URI for a given network attachment will
stay the same, and SHOULD rely on the discovery or provisioning
process each time it joins the network. Depending on how the Captive
Portal system is configured, the URI might be unique for each client
host and between sessions for the same client host.

For example, if the Captive Portal API server is hosted at
example.org, the URI's of the API could be:

o "https://example.org/captive-portal/api"

o "https://example.org/captive-portal/api/X54PD"

4.1.1. Server Authentication

The purpose of accessing the Captive Portal API over an HTTPS
connection is twofold: first, the encrypted connection protects the
integrity and confidentiality of the API exchange from other parties
on the local network; and second, it provides the client of the API
an opportunity to authenticate the server that is hosting the API.
This authentication is aimed at allowing a user to be reasonably
confident that the entity providing the Captive Portal API has a
valid certificate for the hostname in the URI (such as
"example.com"). The hostname of the API SHOULD be displayed to the
user in order to indicate the entity which is providing the API
service.

Clients performing revocation checking will need some means of
accessing revocation information for certificates presented by the
API server. Online Certificate Status Protocol [RFC6960] (OCSP)
stapling, using the TLS Certificate Status Request extension
[RFC6066] SHOULD be used. OCSP stapling allows a client to perform
revocation checks without initiating new connections. To allow for
other forms of revocation checking, a captive network could permit
connections to OCSP responders or Certificate Revocation Lists (CRLs)
that are referenced by certificates provided by the API server. In
addition to connections to OCSP responders and CRLs, a captive
network SHOULD also permit connections to Network Time Protocol (NTP)
[RFC5905] servers or other time-sync mechnisms to allow clients to
accurately validate certificates.

Certificates with missing intermediate certificates that rely on
clients validating the certificate chain using the URI specified in
the Authority Information Access (AIA) extension [RFC5280] SHOULD NOT
be used by the Captive Portal API server. If the certificates do
require the use of AIA, the captive network will need to allow client
access to the host specified in the URI.

If the client is unable to validate the certificate presented by the
API server, it MUST NOT proceed with any of the behavior for API
interaction described in this document. The client will proceed to
interact with the captive network as if the API capabilities were not
present. It may still be possible for the user to access the network
by being redirected to a web portal.

4.2. JSON Keys

The Captive Portal API data structures are specified in JavaScript
Object Notation (JSON) [RFC8259]. Requests and responses for the
Captive Portal API use the "application/captive+json" media type.
Clients SHOULD include this media type as an Accept header in their

GET requests, and servers MUST mark this media type as their Content-
Type header in responses.

The following keys are defined at the top-level of the JSON structure
returned by the API server:

o "captive" (required, boolean): indicates whether the client is in
   a state of captivity, i.e it has not satisfied the conditions to
   access the external network. If the client is captive (i.e.
   captive=true), it can still be allowed enough access for it to
   perform server authentication Section 4.1.1.

o "user-portal-url" (optional, string): provides the URL of a web
   portal with which a user can interact.

o "venue-info-url" (optional, string): provides the URL of a webpage
   or site on which the operator of the network has information that
   it wishes to share with the user (e.g., store info, maps, flight
   status, or entertainment).

o "expire-date" (optional, string formatted as [RFC3339] datetime):
   indicates the date and time after which the client will be in a
   captive state. The API server SHOULD include this value if the
   client is not captive (i.e. captive=false) and SHOULD omit this
   value for captive clients.

o "bytes-remaining" (optional, integer): indicates the number of
   bytes remaining, after which the client will be in placed into a
   captive state. The byte count represents the total number of IP
   packet (layer 3) bytes sent and received by the client. Captive
   portal systems might not count traffic to whitelisted servers,
   such as the API server, but clients cannot rely on such behavior.

The valid JSON keys can be extended by adding entries to the Captive
Portal API Keys Registry Section 6. If a client receives a key that
it does not recognize, it MUST ignore the key and any associated
values. All keys other than the ones defined in this document as
"required" will be considered optional.

4.3. Example Interaction

A client connected to a captive network upon discovering the URI of
the API server will query the API server to retrieve information
about its captive state and conditions to escape captivity. To
request the Captive Portal JSON content, a client sends an HTTP GET
request:

```
GET /captive-portal/api/X54PD
Host: example.org
Accept: application/captive+json
```

The server then responds with the JSON content for that client:

```
HTTP/1.1 200 OK
Cache-Control: private
Date: Mon, 04 Dec 2013 05:07:35 GMT
Content-Type: application/captive+json

{
   "captive": true,
   "user-portal-url": "https://example.org/portal.html",
   "venue-info-url": "https://flight.example.com/entertainment",
   "expire-date": "2014-01-01T23:28:56.782Z"
}
```

Upon receiving this information the client will provide this
information to the user so that they may navigate the web portal (as
specified by the user-portal-url value) to enable access to the
external network. Once the user satisfies the requirements for
extenal network access, the client SHOULD query the API server again
to verify that it is no longer captive.

5. Security Considerations

One of the goals of this protocol is to improve the security of the
communication between client hosts and Captive Portal systems.
Client traffic is protected from passive listeners on the local
network by requiring TLS-encrypted connections between the client and
the Captive Portal API server, as described in Section 4. All
communication between the clients and the API server MUST be
encrypted.

In addition to encrypting communications between clients and Captive
Portal systems, this protocol requires a basic level of
authentication from the API server, as described in Section 4.1.1.
Specifically, the API server MUST present a valid certificate on
which the client can perform revocation checks. This allows the
client to ensure that the API server has authority for a hostname
that can be presented to a user.

It is important to note that while the server authentication checks
can validate a specific hostname, it is certainly possible for the
API server to present a valid certificate for a hostname that uses
non-standard characters or is otherwise designed to trick the user

into believing that its hostname is some other, more trustworthy,
name. This is a danger of any scenario in which a hostname is not
typed in by a user.

5.1. Privacy Considerations

Information passed in this protocol may include a user's personal
information, such as a full name and credit card details. Therefore,
it is important that Captive Portal API Servers do not allow access
to the Captive Portal API over unencrypted sessions.

6. IANA Considerations

IANA is requested to create a registration for an "application/
captive+json" media type (Section 6.1) and a registry for fields in
that format (Section 6.2).

6.1. Captive Portal API JSON Media Type Registration

This document registers the media type for Captive Portal API JSON
text, "application/captive+json".

Type name: application

Subtype name: captive+json

Required parameters: None

Optional parameters: None

Encoding considerations: Encoding considerations are identical to
   those specified for the "application/json" media type.

Security considerations: See Section 5

Interoperability considerations: This document specifies format of
   conforming messages and the interpretation thereof.

Published specification: This document

Applications that use this media type: This media type is intended
   to be used by servers presenting the Captive Portal API, and
   clients connecting to such captive networks.

Additional information: None

Person & email address to contact for further information: See
   Authors' Addresses section.

      Intended usage: COMMON

      Restrictions on usage: None

      Author: CAPPORT IETF WG

      Change controller: IETF

[6.2](#). **Captive Portal API Keys Registry**

   IANA is asked to create and maintain a new registry called "Captive
   Portal API Keys", which will reserve JSON keys for use in Captive
   Portal API data structures. The initial contents of this registry
   are provided in [Section 4.2](#).

   Each entry in the registry contains the following fields:

   Key: The JSON key being registered, in string format.

   Type: The type of the JSON value to be stored, as one of the value
      types defined in [[RFC8259](#)].

   Description: A brief description explaining the meaning of the
      value, how it might be used, and/or how it should be interpreted
      by clients.

   New assignments for Captive Portal API Keys Registry will be
   administered by IANA through Expert Review [[RFC8126](#)]. The Designated
   Expert is expected to validate the existence of documentation
   describing new keys in a permanent publicly available specification.
   The expert is expected to validate that new keys have a clear meaning
   and do not create unnecessary confusion or overlap with existing
   keys. Keys that are specific to non-generic use cases, particularly
   ones that are not specified as part of an IETF document, are
   encouraged to use a domain-specific prefix.

[7](#). **Acknowledgments**

   This work in this document was started by Mark Donnelly and Margaret
   Cullen. Thanks to everyone in the CAPPORT Working Group who has
   given input.

[8](#). **References**

8.1. Normative References

[RFC2818]   Rescorla, E., "HTTP Over TLS", RFC 2818,
            DOI 10.17487/RFC2818, May 2000,
            <https://www.rfc-editor.org/info/rfc2818>.

[RFC3339]   Klyne, G. and C. Newman, "Date and Time on the Internet:
            Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002,
            <https://www.rfc-editor.org/info/rfc3339>.

[RFC5280]   Cooper, D., Santesson, S., Farrell, S., Boeyen, S.,
            Housley, R., and W. Polk, "Internet X.509 Public Key
            Infrastructure Certificate and Certificate Revocation List
            (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008,
            <https://www.rfc-editor.org/info/rfc5280>.

[RFC5905]   Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch,
            "Network Time Protocol Version 4: Protocol and Algorithms
            Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010,
            <https://www.rfc-editor.org/info/rfc5905>.

[RFC6066]   Eastlake 3rd, D., "Transport Layer Security (TLS)
            Extensions: Extension Definitions", RFC 6066,
            DOI 10.17487/RFC6066, January 2011,
            <https://www.rfc-editor.org/info/rfc6066>.

[RFC6960]   Santesson, S., Myers, M., Ankney, R., Malpani, A.,
            Galperin, S., and C. Adams, "X.509 Internet Public Key
            Infrastructure Online Certificate Status Protocol - OCSP",
            RFC 6960, DOI 10.17487/RFC6960, June 2013,
            <https://www.rfc-editor.org/info/rfc6960>.

[RFC8126]   Cotton, M., Leiba, B., and T. Narten, "Guidelines for
            Writing an IANA Considerations Section in RFCs", BCP 26,
            RFC 8126, DOI 10.17487/RFC8126, June 2017,
            <https://www.rfc-editor.org/info/rfc8126>.

[RFC8259]   Bray, T., Ed., "The JavaScript Object Notation (JSON) Data
            Interchange Format", STD 90, RFC 8259,
            DOI 10.17487/RFC8259, December 2017,
            <https://www.rfc-editor.org/info/rfc8259>.

8.2. Informative References

[I-D.ietf-capport-architecture]
            Larose, K. and D. Dolson, "CAPPORT Architecture", draft-
            ietf-capport-architecture-04 (work in progress), June
            2019.

Authors' Addresses

    Tommy Pauly (editor)
    Apple Inc.
    One Apple Park Way
    Cupertino, California 95014
    United States of America

    Email: tpauly@apple.com

    Darshak Thakore (editor)
    CableLabs
    858 Coal Creek Circle
    Louisville, CO 80027
    United States of America

    Email: d.thakore@cablelabs.com