Captive Portal WG
Internet-Draft
Intended status: Informational
Expires: September 14, 2017

M. Donnelly
M. Cullen
Painless Security

March 13, 2017

Captive Portal (CAPPORT) API
draft-donnelly-capport-detection-01

Abstract

This document describes an HTTP API that allows User Equipment to detect the existence of a Captive Portal on the local network, determine the properties of the Captive Portal, and satisfy requirements for network access.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document describes a HyperText Transfer Protocol (HTTP) Application Program Interface (API) that allows User Equipment to detect the existence of a Captive Portal (CAPPORT) on the local network, determine the properties of the Captive Portal, and satisfy requirements for network access. The API defined in this document has been designed to meet the requirements of the CAPPORT API, as discussed in the CAPPORT Architecture [I-D.larose-capport-architecture].

2. Requirements Notation

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 [RFC2119].

3. Workflow

The CAPPORT protocol consists of three phases. In the first phase User Equipment acquires an IP address and determines the URL of the local CAPPORT API Server, if any. The second phase consists of the User Equipment querying the CAPPORT API Server for the requirements for accessing its protected networks, and submitting proofs of meeting those requirements. In the third phase, the User Equipment is granted access to the protected network and can query the CAPPORT API Server for status.

During the first phase, User Equipment uses the Dynamic Host Configuration Protocol (DHCP) or IPv6 Router Advertisements (RAs) to acquire an IP address and to determine the URL for the local CAPPORT API Server. This details for the first phase are described in RFC 7710 [RFC7710], and the rest of this document assumes that the User Equipments already has a URL to reach the CAPPORT API Server.

The second phase begins with the User Equipment accessing the URL provided in the first phase. The CAPPORT API Server responds with the current status of the User Equipment’s access to the protected networks and any conditions requirements to gain access to the protected networks. The User Equipment then submits proofs of satisfying the access requirements to the CAPPORT API Server. The CAPPORT API Server again responds with the current status of the User Equipment and any additional requirements necessary to gain access to the protected network. The second phase continues until all of the requirements are met; the CAPPORT API Server grants access to the protected network and responds with a status indicating the access.

At any point in the second phase, the User Equipment MAY stop communicating over the CAPPORT protocol and instead direct a web browser to access the URL. The web browser then becomes the agent for proving that the User Equipment meets the requirements for access to the protected networks.

During the third phase, the User Equipment has access to the protected network. The User Equipment may access the URL provided in the first phase to query the current status. The CAPPORT API Server responds with the current status of the User Equipment. The CAPPORT API Server SHOULD respond with the current status of the User Equipment regardless of whether the User Equipment used the automated CAPPORT protocol or a web browser to complete the second phase.

4. Use of the DHCP Captive-Portal Option

As decribed above, to use the CAPPORT API, User Equipment needs a URL that can be used to reach the CAPPORT API Server. DHCP Servers and IPv6 Routers should provide, and User Equipment SHOULD obtain, the required URL using the DCHP Captive-Portal Option or the IPv6 RA Captive-Portal Option, as described in [RFC7710].

To provide backwards compatibility with the original use of the DHCP and RA options described in RFC7710, the CAPPORT API defined in this document is exclusively accessed using HTTP Methods with an Accept header value of "application/json". Captive Portals that implement the CAPPORT API SHOULD respond to an HTTP GET that has an Accept header of "text/html" with HTML content that, when displayed in a web browser, will allow the user to interactively meet the Captive Portal requirements for network access.

5. CAPPORT API

This section defines the CAPPORT API.

5.1. URLs and HTTP Methods

This section describes the URLs that can be used to access the CAPPORT API.

5.1.1. Associating User Equipment with its URL

The CAPPORT API Server SHOULD associate an incoming request with a particular User Equipment consistently. [TODO: specify how this would happen.]

5.1.2. Fallback URL

The CAPPORT API Server SHOULD respond to HTTP GET requests to the provided URL that specify an Accept header value of "text/html" with HTML content instead of this protocol. If the User Equipment determines that it is unable to satisfy the conditions for network access, it SHOULD display this fallback URL in a web browser to allow the user to complete the network access outside of this protocol.

5.1.3. CAPPORT API POST URL

The CAPPORT API Server SHOULD respond to HTTP POST requests to the provided URL that specify an Accept header value of "application/json" with the CAPPORT API protocol.

5.1.4. CAPPORT REST API DELETE URL

The CAPPORT API Server SHOULD respond to HTTP DELETE requests to the provided URL that specify an Accept header value of "application/json" by revoking any network access to protected networks immediately. The CAPPORT API Server MUST NOT allow any device other than the User Equipment to DELETE the network access of the User Equipment via the CAPPORT API.

The CAPPORT API Server MAY delete the session token (Section 5.2.1.5) for this User Equipment as part of the DELETE request.

5.2. JSON Data Structures

The CAPPORT API data structures are specified in JavaScript Object Notation (JSON) [RFC7159]. This document specifies the structure of the JSON structures and message using the JSON Content Rules (JCR) defined in draft-newton-json-content-rules [I-D.newton-json-content-rules].

5.2.1. CAPPORT Common Elements

This section describes structures that are shared between requests and responses.

5.2.1.1. Toplevel Object

The CAPPORT API will contain JSON-formatted data. The toplevel object contains a networks object whose value is an array of zero or more network objects.

```
    $toplevel = {
      $networks ,
      $session_token ?
    }
```

The toplevel object MUST contain a networks object.

The CAPPORT API Server responses MUST contain a session_token object. The session-token object contains a session token which will be used in ICMP requests as discussed in RFC 7710.

QUESTION: Should the session token just be provided by the server, or should it be negotiated between the client and server using something like a DH exchange?

5.2.1.2. Networks Object

The networks object represents the list of networks being acted on in this CAPPORT session.

```
$networks = {
  ( "DEFAULT" || // ) = $network +
}
```

The networks object is a JSON object whose keys are network names and whose values are network objects. Thus a single response could be used in gaining access to multiple protected networks at once. The first request to the CAPPORT API Server will contain no networks, and acts as a discovery request.

The CAPPORT API Server SHOULD use the special name DEFAULT for one network that provides access to the greater Internet.

5.2.1.3. Network Object

The network object represents a network protected by the Captive Portal.

```
$network = {
  "conditions" : [ $condition + ] ,
  "state" : $network_state ? ,
  "details" : $network_details ?
}
```

The network object MUST contain a ’conditions’ key whose value is an array of one or more $condition objects, which represent the unmet conditions for gaining access to this network. The conditions object SHOULD NOT contain conditions that have already been met.

CAPPORT API Server responses MUST contain the ’state’ key, whose value is the $network_state object, which represents the state of access that the User Equipment has to the network.

CAPPORT API Server responses SHOULD contain the ’details’ key, whose value is the $network_details object, which provides relevant information about the network.

5.2.1.4. Condition Object

The condition object describes one of the conditions necessary for access to the protected network. The CAPPORT API Server uses this object to express the requirements for User Equipment to access the protected network. The User Equipment uses this object as proof that it has satisfied the corresponding requirement for access to the protected network.

```
    $condition = {
      "id" : $uuid,
      "type" : string ? ,
      "requirement_details" : $requirement_details ? ,
      "satisfaction_details" : $satisfaction_details ?
    }
```

The condition object MUST include an ’id’ key whose value is a UUID that uniquely identifies this condition. This ID will be used to match the client condition satisfactions with the server condition requirements.

CAPPORT API Server responses MUST contain the ’type’ key, whose value is a string that represents the type of condition that permits access to the network.

CAPPORT API Server responses MUST contain the ’requirement_details’ key, whose value is the $requirement_details object. The $requirement_details object details the requirements that the User Equipment must pass to gain access to the protected network.

User Equipment requests MUST contain the ’satisfaction_details’ key, whose value is the $satisfaction_details object. The $satisfaction _details object details the proof that the User Equipment has satisfied the conditions of access to the protected network.

5.2.1.5. Session Token Object

The session_token object describes the CAPPORT session token.

```
    $session_token = "session_token" : base64
```

The session_token object MUST include a "session_token" key whose value is a base64-encoded string of a 32-bit session token. This token will be used as proposed in [I-D.larose-capport-architecture]. The CAPPORT API Server SHOULD send the same session token to a given User Equipment in every response, until the User Equipment DELETEs its network access (Section 5.1.4). After a DELETE, the CAPPORT API

Server MAY generate a new session token if the User Equipment makes a new request.

5.2.2. User Equipment Request

For the initial CAPPORT request from the User Equipment, the JSON object will consist of the toplevel object (Section 5.2.1.1) with its required networks (Section 5.2.1.2) and session_token (Section 5.2.1.5) objects. The networks object will contain no networks, and the session_token object will be empty. This acts as a discovery request.

```
{
  "networks" : {}
  "session-token" : ""
}
```

Figure 1

Subsequent CAPPORT requests will contain data to satisfy conditions to access protected networks.

5.2.2.1. Satisfaction Details Object

The satisfaction_details object details proof that the User Equipment has satisfied one of the conditions of access to a protected network.

```
$satisfaction_details = { // : any + }
```

Like the requirement details (Section 5.2.3.1) in the CAPPORT API Server Response, the list of keys and values for this object will depend on the value of the ’type’ key in the enclosing condition (Section 5.2.1.4). Section 6 contains conditions and their Satisfaction Details Objects.

5.2.3. CAPPORT API Server Response

5.2.3.1. Requirement Details Object

The requirement_details object details the requirements of the Captive Portal Enforcement for access to a protected network.

```
$requirement_details = { // : any + }
```

Like the satisfaction details (Section 5.2.2.1), of the User Equipment Request, the list of keys and values for this object will depend on the value of the ’type’ key in the enclosing condition

(Section 5.2.1.4). Section 6 contains conditions and their Requirements Details Objects.

5.2.3.2. Network State Object

The network_state object details the current state of the User Equipment access to the protected network.

```
    $network_state = {
      "permitted" : boolean ,
      "expires" : datetime ? ,
      "bytes_remaining" : integer ?
    }
```

The network_state object MUST contain the "permitted" key, whose boolean value indicates whether the User Equipment is permitted to access the protected network.

The network_state object SHOULD contain the "expires" key if the access to the protected network will expire at a known time in the future. The value is a datetime object of the time the access will expire. If there is not a known expiration time, the key SHOULD be omitted.

The network_state object SHOULD contain the "bytes_remaining" key if the access to the protected network will expire after the User Equipment transfers a known number of bytes. The value is an integer of the number of bytes remaining. If there is not a known limit for this User Equipment, the key MAY be omitted or its value MAY be -1.

6. Network Access Conditions

Captive Portal systems will have many conditions for access to their protected networks. The conditions object is open for use in expressing different conditions. Each condition MUST define a "type" string, its requirement_details, and its satisfaction_details.

6.1. Terms and Conditions

One common use of a Captive Portal is for the User to accept some terms and conditions for the network access. This network access condition will communicate the terms and conditions to the User Equipment, and communicate their acceptance back to the CAPPORT API Server.

For this network access condition, the condition object’s ’type’ value MUST be "t&c"

This condition is satisfied by presenting an MD5 sum of the terms and conditions document referenced by the requirements. This has the property that the MD5 sum will not change unless the terms and conditions document itself changes. User Equipment MAY cache values and submit a cached value for the MD5 sum preemptively without retrieving the terms and conditions document.

6.1.1. Requirements

```
$requirement_details = {
  "text" : string ?,
  "html" : string ?
}
```

The requirement_details object for the Terms and Conditions network access condition MUST include the "text" key, whose value is a URL referencing the plaintext terms and conditions which govern the use of the protected network.

The requirement_details object for the Terms and Conditions network access condition MUST include the "html" key, whose value is a URL referencing the HTML-fomatted terms and conditions which govern the use of the protected network.

6.1.2. Satisfaction

```
$satisfaction_details = {
  "text" : string ?,
  "html" : string ?
}
```

The satisfaction_details object for the Terms and Conditions network access condition MUST include one of "text" or "html" as a key. The satisfaction_details MAY include both.

The "text" key of the satisfaction_details object has a string value that is an MD5 sum of the document referred to by the URL provided in the Requirement Details (Section 6.1.1) "text" key’s value.

The "html" key of the satisfaction_details object has a string value that is an MD5 sum of the document referred to by the URL provided in the Requirement Details (Section 6.1.1) "html" key’s value.

6.2. Passcode

Another common use of a captive portal is to have a user enter a passcode to gain access to the protected network. The Passcode network access condition will communicate the requirement for that passcode to the User Equipment and satisfy the Captive Portal Enforcement that the User Equipment has the correct passcode.

For the Passcode network access condition, the condition object’s "type" value must be "passcode".

6.2.1. Requirements

```
    $requirement_details = { }
```

The requirement_details object of the Passcode network access condition has no elements.

6.2.2. Satisfaction

```
    $satisfaction_details = {
      "passcode" : string
    }
```

The satisfaction_details object of the Passcode network access condition MUST include the "passcode" key, whose value is a string of the passcode that grants access to the protected network.

7. IANA Considerations

This document does not require any IANA allocations. Please remove this section before RFC publication.

8. Security Considerations

The CAPPORT API described in this document is intended to automate a process that is currently accomplished by a user filling out a HTML form in a Web Browser. Therefore, this mechanism should meet the requirement of being no less secure than presenting the user with a HTML form for completion in a Web Browser, and submitting that form to a Captive Portal.

TBD: Provide complete security requirements and analysis.

8.1. 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 CAPPORT API Servers do not allow access to the CAPPORT API over unecrypted sessions.

9. Acknowledgements

This document was written using xml2rfc, as described in [RFC7749]

10. References

10.1. Normative References

[I-D.larose-capport-architecture]
Larose, K. and D. Dolson, "CAPPORT Architecture", draft-larose-capport-architecture-00 (work in progress), March 2017.

[I-D.newton-json-content-rules]
Newton, A. and P. Cordell, "A Language for Rules Describing JSON Content", draft-newton-json-content-rules-07 (work in progress), September 2016.

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

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7710] Kumari, W., Gudmundsson, O., Ebersman, P., and S. Sheng, "Captive-Portal Identification Using DHCP or Router Advertisements (RAs)", RFC 7710, DOI 10.17487/RFC7710, December 2015, <http://www.rfc-editor.org/info/rfc7710>.

10.2. Informative References

[RFC7749] Reschke, J., "The "xml2rfc" Version 2 Vocabulary", RFC 7749, DOI 10.17487/RFC7749, February 2016, <http://www.rfc-editor.org/info/rfc7749>.

Authors’ Addresses

Mark Donnelly
Painless Security
14 Summer Street, Suite 202
Malden, MA 02148
USA

Email: mark@painless-security.com
URI: http://www.painless-security.com

Margaret Cullen
Painless Security
14 Summer Street, Suite 202
Malden, MA 02148
USA

Phone: +1 781 405-7464
Email: margaret@painless-security.com
URI: http://www.painless-security.com