intarea
Internet-Draft
Intended status: Informational
Expires: September 14, 2017

B. Bruneau
Ecole polytechnique
P. Pfister
Cisco
D. Schinazi
T. Pauly
Apple
E. Vyncke, Ed.
Cisco
March 13, 2017

Proposals to discover Provisioning Domains
draft-bruneau-intarea-provisioning-domains-00

Abstract

This document describes one possible way for hosts to retrieve additional information about their Internet access configuration. The set of configuration items required to access the Internet is called a Provisioning Domain (PvD) and is identified by a Fully Qualified Domain Name.

This document separates the way of getting the Provisioning Domain identifier, the way of getting the Provisioning Domain information and the potential information contained in the Provisioning Domain.

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 http://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 September 14, 2017.

Table of Contents

1. Introduction

It has become very common in modern networks that hosts have Internet or more specific access through different networking interfaces, tunnels, or next-hop routers. The concept of Provisioning Domain (PvD) was defined in RFC7556 [RFC7556] as a set of network configuration information which can be used by hosts in order to access the network. In this document, PvDs are associated with a Fully Qualified Domain Name (called PvD ID) which is used within the host to identify correlated sets of configuration data and also used to retrieve additional information about the services that the network provides.

Devices connected to the Internet through multiple interfaces would typically be provisioned with one PvD per interface, but it is worth noting that multiple PvDs with different PvD IDs could be provisioned on any host interface, as well as noting that the same PvD ID could be used on different interfaces in order to inform the host that both PvDs, on different interfaces, ultimately provide equivalent services.

This document proposes multiple methods allowing the host to to retrieve the PvD ID associated with a set of networking discover the PvD and retrieve the PvD information. It also explains configuration as well as the methods and format in order to retrieve some of the parameters that can describe a PvD.

2. Terminology

PvD — A provisioning domain, usually with a set of provisioning domain information; for more information, see [RFC7556].

2.1. Requirements Language

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 RFC 2119 [RFC2119].

3. Retrieving the PvD ID

In this document, each provisioning domain is identified by a PvD ID. The PvD ID is a Fully Qualified Domain Name which belongs to the network operator to avoid conflicts among network operators. The same PvD ID can exist in several access networks if the set of configuration information is identical in all those networks (such as in all home networks of a residential subscriber). Within a host, the PvD ID SHOULD be associated to all the configuration information associated to this PvD ID; this allows for easy update and removal of information while keeping a consistent state.

This section assumes that IPv6 Router Advertisements are used to discover the PvD ID and explains why this technique was selected.

3.1. Using One Router Advertisement per PvD

Hosts receive implicit PvDs by the means of Router Advertisements (RA).

A router MAY add a single PvD ID Option in its RAs. The PvD ID specified in this option is then associated with all the Prefix Information Options (PIO) included in the RA (albeit it is expected that only one PIO will be included in the RA). All other information contained in the RA (notably the RDNSS and Route Information Option) are to be associated with the PvD ID. The set of information contained in the RA forms the bootstrap (or hint) PvD. A new RA option will be required to convey the PvD ID.

When a host receives an RA which does not include a PvD ID Option, the set of information included in the RA (such as Recursive DNS server, IPv6 prefix) is attached to an implicit PvD identified by the local interface ID on which the RA is received, and by the link-local address of the router sending the RA.

In the cases where a router should provide multiple independent PvDs to all hosts, including non-PvD aware hosts, it should send multiple RAs, as proposed in [I-D.bowbakova-rtgwg-enterprise-pa-multihoming] using different source link-local addresses (LLA); the datalink layer (MAC) address could be the same for all the different RA. If the router is actually a VRRP instance, then the procedure is identical

except that the virtual link-layer address is used as well as virtual link-layer addresses.

Using RA allows for an early discovery of the PvD ID as it is early in the interface start-up. As RA is usually processed in the kernel, this requires a host OS upgrade. The RA SHOULD contain other PvD information as explained in section Section 4.1.

3.2. Rationale for not selecting other techniques

There are other techniques to discover the PvD ID that were not selected by the authors and reviewers, this section explains why. The design goal was to be as reliable as possible (do not depend on Internet connectivity) and as fast as possible.

3.2.1. Using DNS-SD

For each received RA including a RDNSS option as well as a DNS search list option, the host MAY retrieve the PvD ID by querying the configured DNS server for records of type PTR associated with _pvd.<DNS search name>. If a PvD ID is configured, the DNS recursive resolver MUST reply with the PvD ID as a PTR record. NXDOMAIN is returned otherwise.

When the RDNSS address is link-local, the host MAY retrieve the PvD ID before configuring its global scope address(es).

Relying on a valid DNS service at the interface bootstrap can lead into delay to start the interface or starting without enough information: for example when the RDNSS is a non local address and there is no Internet connectivity.

3.2.2. Using Reverse DNS lookup

[I-D.stenberg-mif-mpvd-dns] proposes a solution to get the name of the PvD using a reverse DNS lookup based on the host global address(es). It merely relies on prepending a well-known prefix '_pvd' to the reverse lookup, for example ' _pvd....ip6.arpa.'.

However, the PvD information is typically provided by the network operator, whereas the reverse DNS zone could be delegated from the operator to the network user, in which case it would not work.

It also requires a fully functional global address to retrieve the information which may be too late for a correct host configuration. One advantage is that it does not require any change in the IPv6 protocol and no change in the host kernel or even in the CPE.

3.3. IoT Considerations

TBD: should state that when end-host (IoT) cannot impletement completely this RFC it MAY select any of the PvD or the router SHOULD send a single unicast RA (hence a single PvD) in response to the RS or none if it detects that it cannot offer the right set of network services.

3.4. Linking IPv4 Information to an IPv6 PvD

The document describes IPv6-only PvD but there are multiple ways to link the set of IPv4 configuration information received by DHCPv4:

- correlation based on the data-link layer address of the source, if the IPv6 RA and the DHCPv4 response have the same data-link layer address, then the information contained in the IPv4 DHCP can be linked to the IPv6 PvD;

- correlation based on the interface when there is no data-link address on the link (such as a 3GPP link), then the information contained in the IPv4 PDP context can be linked to the IPv6 PvD (*** TO BE VERIFIED before going -01);

- correlation based on the DNS search list, if the DNS search lists are identical between the IPv6 RDNSS and the DHCPV4 response, then the information contained in the IPv4 DHCP response can be linked to the IPv6 PvD.

The correlation could be useful for some PvD information such as Internet reachability, use of captive portal, display name of the PvD, ...

In cases where the IPv4 configuration information could not be associated with a PvD, hosts MUST consider it as attached to an independent implicit PvD containing no other information than what is provided through DHCPv4.

4. Getting the full set of PvD information

Once the PvD ID is known, it MAY be used to retrieve additional information. PvD Information is modeled as a key-value dictionary which keys are ASCII strings of arbitrary length, and values are either strings (encoding can vary), ordered list of values (recursively), or a dictionary (recursively).

The PvD Information may be retrieved from multiple sources (from the bootstrap PvD contained in the RA to the secondary/extended PvD described in this section); the PvD ID is then used to correlate the

information from different sources. The way a host should operate when receiving conflicting information is TBD but it SHOULD at least override information from less authenticated sources (RA) by more authenticated sources (via TLS).

4.1. Using the PvD Bootstrap Information Option

Routers MAY transmit, in addition to the PvD ID option, a PvD Bootstrap Information option, containing a first subset of PvD information. The additional pieces of bootstrap PvD information data set are transmitted using the short-hand notation proposed in Section 5. This requires another RA option.

As there is a size limit on the amount of information a single RA can convey, it is likely that the PvD Bootstrap Information option may not contain the whole set of PvD Information. The set of PvD information included in the RA is called PvD Bootstrap Information.

4.2. Downloading a JSON file over HTTPS

The host SHOULD try to download a JSON formatted file over HTTPS in order to get more PvD information.

The host MUST perform an HTTP query to https://<PvD-ID>/v1.json. If the HTTP status of the answer is greater than 400 the host MUST abandon and consider that there is no additional PvD information. If the HTTP status of the answer is between 300 and 400 it MUST follow the redirection(s). If the HTTP status of the answer is between 200 and 300 the host MAY get a file containing a single JSON object.

The host MUST respect the cache information in the HTTP header, if any, and at expiration of the downloaded object, it must fetch a fresher version if any.

4.2.1. Advantages

The JSON format allows advanced structures.

It can be secured using HTTPS (and DNSSEC).

It is easier to update a file on a web server than to edit DNS records. It can be especially important if we want providers to be able to often update the remaining phone plan of the user.

4.2.2. Disadvantages

It is slower than using DNS because HTTPS uses TCP and TLS and needs more packets to be exchanged to get the file.

An additional HTTPS server must be deployed and configured.

4.3. Using DNS TXT ressource records (not selected)

This approach was not selected during the design team meeting but has kept here for reference, it will be removed after global consensus is reached.

The host could perform a DNS query for TXT resource records (RR) for the FQDN used as PvD ID (alternatively for _pvd.<PvD-ID>). For each retrieved PvD ID, the DNS query MUST be sent to the DNS server configured from the same router advertisement as the PvD ID. Syntax of the TXT response is defined in Section 5 (Section 5).

4.3.1. Advantages

It requires a single round-time trip in order to retrieve the PvD Information.

It can be secured using DNSSEC.

4.3.2. Disadvantages

A TXT record is limited to 65535 characters in theory but large size of TXT records could require either DNS over TCP (so loosing the 1-RTT advantage) or fragmented UDP packets (which could be dropped by a bad choice of security policy). Large TXT records could also be used to mount an amplification attack.

4.3.3. Using DNS SRV ressource records

It is expected that the DNS TXT records will be sufficient for the host to configure itself with basic networking and policy configuration. Nevertheless, if further information is required, or when a different security model shall be used to access the PvD Information, a SRV Resource Record including a full URL MAY be included as a response, expecting the host to query this URL in order to retrieve additional PvD information.

5. PvD Information

PvD information is a set of key-value pairs. Keys are ASCII character strings. Values are either a character string, an ordered list of values, or an embedded dictionary. Value types and default behavior with respect to some specific keys MAY be further specified (recursively). Some keys have a default value as described in the following sections. When there is an expiration time in a PvD, then the information MUST be refreshed before the expiration time. The behavior of a host when the refresh operation is not successful is TBD.

Nodes using the PvD MUST support the two encodings:

JSON syntax for the complete set of PvD information;

short-hand notation for the bootstrap PvD.

When the PvD information is transferred as a JSON file, then the key used is the second column of the following table. The syntax of the JSON file is obvioulsy JSON and is richer than the short-hand notation specified in the next paragraph.

When transmitting more information than the PvD ID in the RA (or when DNS TXT resource records are used), the shorthand notataion for PvD information is used and consists of a string containing several "key=value;" substrings. The "key" is the first column of the following tables, the value is encoded as:

Shorthand notation for values:

integer: expressed in decimal format with a '.' (dot) used for decimals;

string: expressed as UTF-8 encoded string, delimited by single quote character, the single quote character can be expressed by two consecutive single quote character;

boolean: expressed as '0' for false and '1' for true;

IPv6 address: printed as RFC5952 [RFC5952].

5.1. PvD Name

PvD SHOULD have a human readable name in order to be presented on a GUI. The name can also be localized.

DNS TXT key/Bootstrap PvD key	JSON key	Description	Type	JSON Example
n	name	User-visible service name, SHOULD be part of the bootstrap PvD	human-readable UTF-8 string	"Foobar Service"
nl10n	localizedName	Localized user-visible service name, language can be selected based on the HTTP Accept-Language header in the request.	human-readable UTF-8 string	"Service Blabla"

5.2. Trust of the bootstrap PvD

The content of the bootstrap PvD (from the original RA) cannot be trusted as it is not authenticated. But, the extended PvD can be associated with the PvD ID (as the PvD ID is used to construct the extended PvD URL) and trusted by the used of TLS. The extended PvD SHOULD therefore include the following information elements and, if they are present, the host MUST verify that the all PIO of the RA fits into the master prefix list. If any PIO prefix from the bootstrap PvD does not fit in the master prefix array, then all information received by the bootstrap PvD must be invalidated. In short, the masterIPv6Prefix received over TLS is used to authenticate the bootstrap PvD.

The values of the bootstrap PvD (RDNSS, ...) are overwritten by the values contained in the trusted extended PvD if they are present.

DNS TXT key	JSON key	Description	Type	JSON Example
mp6	masterIpv6Prefix	All the IPv6 prefixes linked to this PvD (such as a /29 for the ISP).	Array of IPv6 prefixes	["2001:db8::/32 "]

5.3. Reachability

The following set of keys can be used to specify the set of services for which the respective PvD should be used. If present they MUST be honored by the client, i.e., if the PvD is marked as not usable for Internet access (walled garden), then it MUST NOT be used for Internet access. If the usability is limited to a certain set of domain or address prefixes (typical VPN access), then a different PvD MUST be used for other destinations.

DNS TXT key	JSON key	Description	Type	JSON Example
s	noInternet	Internet inaccessible	boolean	true
cp	captivePortal	Presence of a captive portal	boolean	false
z	dnsZones	DNS zones accessible and searchable	array of DNS zone	["foo.com","sub.bar.com"]
6	prefixes6	IPv6-prefixes accessible via this PvD	array of IPv6 prefixes	["2001:db8:a::/48","2001:db8:b:c::/64"]
4	prefixes4	IPv4-prefixes accessible	array of IPv4 prefixes in CIDR reachable via this PvD	["192.0.2.0/24","2.3.0.0/16"]

5.4. DNS Configuration

The following set of keys can be used to specify the DNS configuration for the respective PvD. If present, they MUST be honored and used by the client whenever it wishes to access a resource described by the PvD.

DNS TXT key	JSON key	Description	Value	JSON Example
r	dnsServers	Recursive DNS server	array of IPv6 and IPv4 addresses	["2001:db8::1","192.0.2.2"]
d	dnsSearch	DNS search domains	array of search domains	["foo.com","sub.bar.com"]

5.5. Connectivity Characteristics

NOTE: open question to the authors/reviewers: should this document include this section or is it useless?

The following set of keys can be used to signal certain characteristics of the connection towards the PvD.

They should reflect characteristics of the overall access technology which is not limited to the link the host is connected to, but rather a combination of the link technology, CPE upstream connectivity, and further quality of service considerations.

DNS TXT key	JSON key	Description	Type	JSON Example
tp	throughputMax	Maximum achievable throughput (e.g. CPE downlink/uplink)	object({down (int), up(int)}) in kb/s	{"down": 10000, "up": 5000}
lt	latencyMin	Minimum achievable latency	object({down (int), up(int)}) in ms	{"down": 10, "up": 20}
rl	reliabilityMax	Maximum achievable reliability	object({down (int), up(int)}) in 1/1000	{"down": 1000, "up": 800}
cp	captivePortal	Captive portal	URL of the portal	"https://example.com"
nat	NAT	IPv4 NAT in place	boolean	true
natto	NAT Time-out	The value in seconds of the NAT time-out	Integer	30
srh	segmentRoutingHeader	The IPv6 Segment Routing Header to be used between the IPv6 header and	Binary string	...

		any other headers when using this PvD		
srhDNS	segmentRoutingHeaderDnsFQDN	The DNS FQDN which is used to retrieved the actual IPv6 Segment Routing Header to be used between the IPv6 header and any other headers when using this PvD	Ascii string	srh.pvd-foo.example.org
cost	cost	Cost of using the connection	object	See Section 5.6

5.6. Connection monetary cost

NOTE: This section is included as a request for comment on the potential use and syntax.

The billing of a connection can be done in a lot of different ways. The user can have a global traffic threshold per month, after which his throughput is limited, or after which he/she pays each megabyte. He/she can also have an unlimited access to some websites, or an unlimited access during the weekends.

We propose to split the final billing in elementary billings, which have conditions (a start date, an end date, a destination IP address...). The global billing is an ordered list of elementary billings. To know the cost of a transmission, the host goes through the list, and the first elementary billing whose the conditions are fulfilled gives the cost. If no elementary billing conditions match the request, the host MUST make no assumption about the cost.

5.6.1. Conditions

Here are the potential conditions for an elementary billing. All conditions MUST be fulfill.

Note: the final version should use short-hand key names.

Key	Description	Type	JSON Example
beginDate	Date before which the billing is not valid	ISO 8601	"1977-04-22T06:00:00Z"
endDate	Date after which the billing is not valid	ISO 8601	"1977-04-22T06:00:00Z"
domains	FQDNs whose the billing is limited	array(string)	["deezer.com","spotify.com"]
prefixes4	IPv4 prefixes whose the billing is limited	array(string)	["78.40.123.182/32","78.40.123.183/32"]
prefixes6	IPv6 prefixes whose the billing is limited	array(string)	["2a00:1450:4007:80e::200e/64"]

5.6.2. Price

Here are the different possibilities for the cost of an elementary billing. A missing key means "all/unlimited/unrestricted". If the elementary billing selected has a trafficRemaining of 0 kb, then it means that the user has no access to the network. Actually, if the last elementary billing has a trafficRemaining parameter, it means that when the user will reach the threshold, he/she will not have access to the network anymore.

Key	Description	Type	JSON Example
pricePerGb	The price per Gigabit	float (currency per Gb)	2
currency	The currency used	ISO 4217	"EUR"
throughputMax	The maximum achievable throughput	float (kb/s)	1000
trafficRemaining	The traffic remaining	float (kb)	96000000

5.6.3. Examples

Example for a user with 20 GB per month for 40 EUR, then reach a threshold, and with unlimited data during weekends and to deezer:

```
[
  {
    "domains": ["deezer.com"]
  },
  {
    "prefixes4": ["78.40.123.182/32","78.40.123.183/32"]
  },
  {
    "beginDate": "2016-07-16T00:00:00Z",
    "endDate": "2016-07-17T23:59:59Z",
  },
  {
    "beginDate": "2016-06-20T00:00:00Z",
    "endDate": "2016-07-19T23:59:59Z",
    "trafficRemaining": 96000000
  },
  {
    "throughputMax": 1000
  }
]
```

If the host tries to download data from deezer.com, the conditions of the first elementary billing are fulfilled, so the host takes this elementary billing, finds no cost indication in it and so deduces that it is totally free. If the host tries to exchange data with youtube.com and the date is 2016-07-14T19:00:00Z, the conditions of the first, second and third elementary billing are not fulfilled. But the conditions of the fourth are. So the host takes this

elementary billing and sees that there is a threshold, 12 GB are remaining.

Another example for a user abroad, who has 3 GB per year abroad, and then pay each MB:

```
[
  {
    "beginDate": "2016-02-10T00:00:00Z",
    "endDate": "2017-02-09T23:59:59Z",
    "trafficRemaining": 9200000
  },
  {
    "pricePerGb": 30,
    "currency": "EUR"
  }
]
```

5.7. Private Extensions

keys starting with "x-" are reserved for private use and can be utilized to provide vendor-, user- or enterprise-specific information. It is RECOMMENDED to use one of the patterns "x-FQDN-KEY" or "x-PEN-KEY" where FQDN is a fully qualified domain name or PEN is a private enterprise number [PEN] under control of the author of the extension to avoid collisions.

5.8. Examples

5.8.1. Using JSON

```
{
  "name": "Orange France",
  "localizedName": "Orange France",
  "dnsServers": ["8.8.8.8", "8.8.4.4"],
  "throughputMax": {
    "down": 100000,
    "up": 20000
  },
  "cost": [
    {
      "domains": ["deezer.com"]
    },
    {
      "prefixes4": ["78.40.123.182/32","78.40.123.183/32"]
    },
    {
      "beginDate": "2016-07-16T00:00:00Z",
      "endDate": "2016-07-17T23:59:59Z",
    },
    {
      "beginDate": "2016-06-20T00:00:00Z",
      "endDate": "2016-07-19T23:59:59Z",
      "trafficRemaining": 96000000
    },
    {
      "throughputMax": 1000
    }
  ]
}
```

5.8.2. Using DNS TXT records

```
n=Orange France
r=8.8.8.8,8.8.4.4
tp=100000,20000
cost+0+domains=deezer.com
cost+1+prefixes4=78.40.123.182/32,78.40.123.183/32
cost+2+beginDate=2016-07-16T00:00:00Z
cost+2+endDate=2016-07-17T23:59:59Z
cost+3+beginDate=2016-06-20T00:00:00Z
cost+3+endDate=2016-07-19T23:59:59Z
cost+3+trafficRemaining=96000000
cost+4+throughputMax=1000
```

6. Use case examples

TBD: 1 or 2 examples when PvD are critical

6.1. Multihoming

First example could be multihoming (very much in-line with bowbakova draft).

6.2. VPN/Extranet example

using PvD to reach a specific destination (such as VPN or extranet).

7. Security Considerations

While the PvD ID can be forged easily, if the host retrieve the extended PvD via TLS, then the host can trust the content of the extended PvD and verifies that the RA prefix(es) are indeed included in the master prefixed of the extended PvD.

8. Acknowledgements

Many thanks to M. Stenberg and S. Barth: Section 5.3, Section 5.5 and Section 5.7 are from their document [I-D.stenberg-mif-mpvd-dns].

Thanks also to Ray Bellis, Lorenzo Colitti, Marcus Keane, Erik Kline, Jen Lenkova, Mark Townsley and James Woodyatt for useful and interesting brainstorming sessions.

9. References

9.1. 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, <http://www.rfc-editor.org/info/rfc2119>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <http://www.rfc-editor.org/info/rfc5952>.

[RFC7556] Anipko, D., Ed., "Multiple Provisioning Domain Architecture", RFC 7556, DOI 10.17487/RFC7556, June 2015, <http://www.rfc-editor.org/info/rfc7556>.

9.2. Informative references

[I-D.bowbakova-rtgwg-enterprise-pa-multihoming]
Baker, F., Bowers, C., and J. Linkova, "Enterprise Multihoming using Provider-Assigned Addresses without Network Prefix Translation: Requirements and Solution", draft-bowbakova-rtgwg-enterprise-pa-multihoming-01 (work in progress), October 2016.

[I-D.stenberg-mif-mpvd-dns]
Stenberg, M. and S. Barth, "Multiple Provisioning Domains using Domain Name System", draft-stenberg-mif-mpvd-dns-00 (work in progress), October 2015.

[PEN] IANA, "Private Enterprise Numbers", <https://www.iana.org/assignments/enterprise-numbers>.

Authors' Addresses

Basile Bruneau
Ecole polytechnique
Vannes 56000
France

Email: basile.bruneau@polytechnique.edu

Pierre Pfister
Cisco
11 Rue Camille Desmoulins
Issy-les-Moulineaux 92130
France

Email: ppfister@cisco.com

David Schinazi
Apple

Email: dschinazi@apple.com

Tommy Pauly
Apple

Email: tpauly@apple.com

Eric Vyncke (editor)
Cisco
De Kleetlaan, 6
Diegem 1831
Belgium

Email: evyncke@cisco.com

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

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 http://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 September 14, 2017.

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

Internet Engineering Task Force K. Larose
Internet-Draft D. Dolson
Intended status: Informational Sandvine
Expires: September 10, 2017 March 9, 2017

CAPPORT Architecture
draft-larose-capport-architecture-00

Abstract

This document aims to document concensus on the CAPPORT architecture. DHCP, ICMP, and an HTTP API are used to provide the solution.

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 http://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 September 10, 2017.

Copyright Notice

Table of Contents

1. Introduction

Problems with captive portals have been described in [I-D.nottingham-capport-problem].

This document standardizes an architecture for implementing captive portals that provides tools for addressing most of those problems.

The architecture also attempts to enable IoT devices, in particular devices without user interfaces, to navigate a captive portal.

The architecture uses the following mechanisms:

- DHCP/DHCP6 providing end-user devices with a URI in the Captive-Portal Router Advertisement option [RFC7710]. This URI is an API that the end-user devices access for information about what is required to escape captivity.

- Notifying end-user devices of captivity with ICMP/ICMP6 "unreachable" messages. This notification can work with any Internet protocol, not just clear-text HTTP. This notification does not carry the portal URI, rather triggers the DHCP-

provisioned portal to be accessed. This notification carries a "reason" that allows the devices to receive customized work-flows at the portal.

- Receipt of the ICMP/ICMP6 messages inform an end-user device that it is captive. This permits the device to take immediate action to satisfy the portal (according to its configuration/policy). The architecture recommends the device to query the DHCP-provisioned CAPPORT URI with the specified "reason". This API returns a status and a menu for navigating the captive portal. Typically one of the menu items is a web page suitable for browsing.

The architecture attempts to provide privacy, authentication, and safety mechanisms to the extent possible.

1.1. Requirements Language

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 RFC 2119 [RFC2119].

1.2. Terminology

Captive Network: A network for which communication outside of it is subject to a captive portal

Captive Portal Enforcement: The device which enforces the captive portal in the captive netowrk

Captive Portal User Equipment: Also known as User Equipment. A device which wants to communicate outside the captive network

2. Components

2.1. User Equipment

The User Equipment is the device that a user desires to communicate with a network. The User Equipment communication is typically restricted by the Captive Portal Enforcement, described in Section 2.4, until site-specific requirements have been met.

- May be interactive or non-iteractive

- May have different mechanisms for notifying the user of the captive portal

- Needs to recognize the ICMP unreachable message, and to invoke its captive portal handling in response to it.

- Needs to cache the URI for the captive portal API from the DHCP lease.

- May cache credentials to automatically respond to captive portal notifications

- Interactive User Equipment typically ask their users how to proceed through interacting with the captive portal. Interactions may be as simple as accepting a terms of agreement, or as complicated as filling out some forms.

- An example interactive User Equipment is a smart phone.

- Non interactive User Equipment may be provisioned with credentials out of band (e.g., via USB programming) in order to automatically gain access.

- An example non interactive User Equipment is an IoT device such as a smart thermostat.

- May need to distinguish between types of User Equipment here.

2.2. DHCP Server

A standard for providing a portal URI is described in [RFC7710]. The CAPPORT architecture expects this URI to access the API described in Section 2.3.

Although it is not clear from RFC7710 what protocol should be executed at the specified URI, it may have been assumed to be an HTML page, and hence there may be User Equipment assuming a browser should open this URI. For backwards compatibility, it might be necessary for the server to check Agent-Id when serving the URI.

2.3. Captive Portal API Server

The User Equipment performs GET at the DHCP-specified URI. The API is implemented at the CAPPORT API Server. The response is a JSON document. The following information should be available in the response document, allowing User Equipment devices to choose the next step:

- Quota information (remaining time/bytes/etc.)

- Whether the device is allowed through captive portal or blocked.

- Method of providing credentials to gain access.
- Describe the required credentials to gain access.
- URL of a web page for devices with browsers and humans.
- A token used to verify later ICMP messages are valid.

The CAPPORT API is intended to provide information and a menu of choices to support options for interactive or non-interactive User Equipment.

The CAPPORT API should support TLS for privacy. [Does this API need to be secure, or do we place security at the interfaces it points to?]

2.4. Captive Portal Enforcement

The Captive Portal Enforcement component restricts network access to User Equipment according to site-specific policy. Typically User Equipment is denied network access until it has performed some action.

The Captive Portal Enforcement component:

- Allows traffic through for allowed User Equipment.
- Blocks traffic and sends ICMP notifications for disallowed User Equipment.
- Permits disallowed User Equipment to access necessary APIs and web pages to fulfill requirements of exiting captivity.
- May modify responses to canary URLs, or perform other methods of notification.
- Updates policy per User Equipment in response to operations from the Captive Portal API.

2.5. ICMP/ICMP6

A mechanism to trigger captive portal work-flows in the User Equipment is proposed earlier in [I-D.wkumari-capport-icmp-unreach]. Additionally, the Unreachable message carries a token to prove it is a valid notification.

The Captive Portal Enforcement function is required to send such ICMP messages when disallowed User Equipment attempts to send to the network.

The ICMP messages MUST NOT be sent to the Internet devices. The indications are only sent to the User Equipment.

The User Equipment MUST verify that the token matches the token received earlier via the CAPPORT API. If tokens do not match, the ICMP message MUST be discarded with no further impact. (It MAY be counted.)

The User Equipment does not necessarily deliver the impact of the ICMP message to the application that triggered it. The User Equipment may be able to satisfy the Captive Portal requirements quickly enough that existing transport connections are not impacted.

2.6. Component Diagram

The following diagram shows the communication between each component.

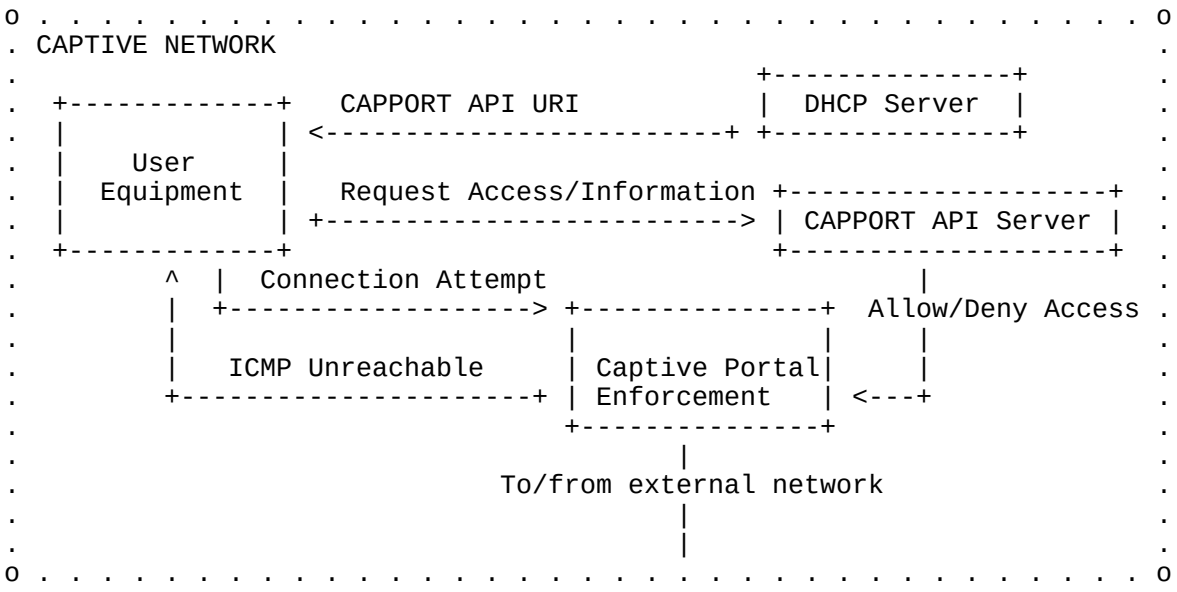

Figure 1: Captive Portal Architecture Component Diagram

In the diagram:

- The User Equipment communicates with the DHCP Server to get access to the captive network, and learn about the CAPPORT API URI.
- The User Equipment attempts to communicate through the captive portal enforcement device.
- The Captive Portal Enforcement device either lets the User Equipment’s traffic through, or responds with an ICMP Unreachable
- The User Equipment requests access to outside the captive network, or requests more information, from the CAPPORT API server
- The CAPPORT API server directs the Captive Portal Enforcement device to either allow or deny access in response to requests from the User Equipment or quota/timing restrictions.

3. Solution Workflow

This section describes the general workflow of solutions adhering to the architecture.

3.1. Initial Connection

1. The User Equipment joins the captive network by acquiring a DHCP lease
2. The User Equipment learns the URI for the Captive Portal API from the DHCP response ([RFC7710]).
3. The User Equipment accesses the CAPPORT API to receive parameters of the Captive Network, including the token.
4. The User Equipment communicates with the CAPPORT API to gain access to the outside network.
5. The Captive Portal API server indicates to the Captive Portal Enforcement device that the User Equipment is allowed through
6. The User Equipment attempts a connection outside the captive network
7. If the requirements have been satisfied, the access is permitted; otherwise the "Expired" behavior occurs
8. The User Equipment accesses the network until conditions Expire

3.2. Connection About to Expire

1. The User Equipment sends a packet to the outside network.

2. The Captive Portal Enforcement detects that the User Equipment's access is about to expire (low quota/time/etc)

3. The Captive Portal Enforcement sends an ICMP unreachable to the User Equipment indicating that it needs to refresh its access. [I-D.wkumari-capport-icmp-unreach]. The message contains the token given to the User Equipment earlier.

4. The User Equipment verifies the message, including the token

5. The User Equipment handles this message by invoking its captive portal handling infrsatructure.

6. The captive portal handling infrastructure communicates with the Captive Portal API to gain access to outside the captive network

7. The Captive Portal API Server gives more quota (time, bytes, etc.) to the User Equipment by indicating to the Captive Portal Enforcement the new, extended quota.

8. The User Equipment continues unaffected.

3.3. Connection expired

1. The User Equipment sends a packet to the outside network.

2. The Captive Portal Enforcement device detects that the User Equipment's access has expired.

3. The remaining workflow is that same as for the initial connection.

User Equipment may attempt to maintain transport connections, leaving it to the application to determine timeouts.

User Equipment may preemptively invoke its captive portal handling infrastructure when receiving the DHCP response indicating that it is behind a captive portal, rather than waiting for the ICMP unreachable message.

4. IANA Considerations

This memo includes no request to IANA.

5. Security Considerations

5.1. Authenticated APIs

The solution described here assumes that when the User Equipment needs to trust the API server, server authentication will be utilized.

TODO: this document has not specified the authentication mechanism.

5.2. Risk of Nuisance Captive Portal

It is possible for any user on the Internet to send ICMP packets in an attempt to cause the receiving equipment to go to the captive portal. This has been considered and addressed in the following ways:

The ICMP packet does not carry the URL, making this method safer than 307-redirect methods currently in use.

The ICMP packet carries a token that would not be available, even to an on-path attacker. Although possible to guess by brute force, the impact is nuisance due to other precautions. We suggest a 32-bit token would be sufficient to deter nuisance attacks.

Even when redirected, the User Equipment securely authenticates with API servers.

5.3. User Options

The ICMP messaging informs the end-user device it is being held captive. There is no requirement that the device do something about this. Devices may permit users to disable automatic reaction to captive-portal indications. Hence, end-user devices may allow users to manually control captive portal interactions.

6. References

6.1. 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, <http://www.rfc-editor.org/info/rfc2119>.

[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>.

6.2. Informative References

[I-D.nottingham-capport-problem] Nottingham, M., "Captive Portals Problem Statement", draft-nottingham-capport-problem-01 (work in progress), April 2016.

[I-D.wkumari-capport-icmp-unreach] Bird, D. and W. Kumari, "Captive Portal ICMP Destination Unreachable", draft-wkumari-capport-icmp-unreach-01 (work in progress), April 2015.

Authors' Addresses

Kyle Larose
Sandvine
408 Albert Street
Waterloo, ON N2L 3V3
Canada

Phone: +1 519 880 2400
Email: klarose@sandvine.com

David Dolson
Sandvine
408 Albert Street
Waterloo, ON N2L 3V3
Canada

Phone: +1 519 880 2400
Email: ddolson@sandvine.com