Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 31, 2013

P. Hallam-Baker
Comodo Group Inc.

July 30, 2012

OmniBroker Protocol
draft-hallambaker-omnibroker-02

Abstract

An Omnibroker is an agent chosen and trusted by an Internet user to provide information such as name and certificate status information that are in general trusted even if they are not trustworthy. Rather than acting as a mere conduit for information provided by existing services, an Omnibroker is responsible for curating those sources to protect the user.

The Omnibroker Protocol (OBP) provides an aggregated interface to trusted Internet services including DNS, OCSP and various forms of authentication service. Multiple transport bindings are supported to permit efficient access in virtually every common deployment scenario and ensure access in any deployment scenario in which access is not being purposely denied.

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 January 31, 2013.

Table of Contents

1. Definitions

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].

2. Purpose

An Omnibroker is an agent chosen and trusted by an Internet user to provide information such as name and certificate status information that are in general trusted even if they are not trustworthy. Rather than acting as a mere conduit for information provided by existing services, an Omnibroker is responsible for curating those sources and providing autheticated, curated results to the endpoint client.

Unlike the traditional trusted information services that are expected to deliver the same response to a query regardless of who asks the question, OBP permits an Omnibroker to return a response that is tailored to the specific party asking the question. This permits the use of an authentication approach that has negligible impact on performance and permits an Omnibroker to answer questions that a traditional public Internet information service could not. In particular, an Omnibroker can broker peer to peer connections

2.1. A Curated Service

In the traditional configuration, an Internet host accepts DNS service from the IP address specified by the local network provider, frequently the DNS server advertised by the local DHCP service. This approach creates an obvious security risk as DNS is clearly a trusted service and a random DNS service advertised by the local DNS is clearly not trustworthy.

A policy of only using a chosen DNS service provides a reduction in risk but only a modest one since the standard DNS service does not provide authenticated results. Many local area networks intercept all DNS traffic and process it through the local DNS server regardless of the intended destination IP address. This practice is highly desirable if it would prevent a client from accessing an untrustworthy DNS service in place of a trustworthy local DNS service and extreemly undesirable in the contrary case.

In addition to ensuring the authenticity of DNS resolution responses, such services frequently provide filtering of Internet addresses the network provider considers undesirable. Many workplaces block access

to Web sites that are considered detrimental to productivity. Many parent subscribe to services that filter content they consider undesirable. While the value of such services is debatable they are services that those customers have chosen to deploy on their networks to meet their perceived security requirements. New security proposals that do not support such capabilities or seek to actually circumvent them will not be acceptable to those constituencies.

While DNS filtering is a form of censorship, not all forms of DNS filering are intrinsically undesirable censorship. Spam filtering is also a form of censorship albeit one that is not normally regarded as such because it most Internet users now consider it to be an essential security control. Anti-Virus tools are also a form of censorship. DNS filtering tools that block access to sites that distribute malware are also a form of censorship and are rapidly gaining popularity for the same reason.

While all forms of censorship raise civil liberties concerns, censorship should not automatically raise civil liberties objections. It is not the fact that filtering is taking place but the party that is in control of the filtering that should raise civil liberties concerns. In an Internet of 2 billion users, all users are obliged to perform some filtering. OBP is designed to make the party that is in control of the filtering process apparent and provide the end user with the ability to select the Omnibroker of their choice.

DNSSEC provides a means of determining that a DNS record is the authentic record published by the source but this capability alone does not meet the security requirements for DNS resolution services as they have come to be understood since the protocol was first proposed.

Internet users want to be safe from all forms of attack, not just the DNS resolver mani-in-the-middle attack that 'end-to-end' deployment of DNSSEC is designed to address. An Internet user is far more likely to be directed to a site with a DNS name controlled by a criminal gang than be subject to a man-in-the-middle attack.

Most users would prefer to have an Internet connection that is 'curated' to remove at least some of the locations they consider to be undesirable. The fact that an organised criminal gang has put a host on the Internet does not mean that any other Internet user should be obliged to allow it to connect to any of the machines that they own.

The same argument for curating the raw results applies to other forms of trusted information service. The fact that a Certificate Authority has issued a digital certificate and considers it to be

valid should not mean that the end party is automatically considered trustworthy by anyone and for any purpose.

The deployment of security policy capabilities presents another case in which direct reliance on raw data is undesirable. While security policies such as 'host always offers TLS' or 'mail server always signs outgoing mail with DKIM' can provide considerable security advantages, only some of the security policy information that is published is accurate and kept up to date. Curating such data sources typically proves essential if an unacceptable rate of false positives is to be avoided.

Although a client is permitted to curate its own data sources it rarely has a sufficient amount of data to make decisions as accurately as a network service that can draw on a wide variety of additional data including tracking of communication patterns, historical data series and third party reputation services.

Curation in the network offers better asgility than the client approach. Agility is an important consideration when an attacker changes their strategy rapidly and repeatedly to counter new defensive controls.

While curating trusted data sources is an established and proven practice, current practice has been to curate each source individually. This approach avoids the need to write a new protocol but limits the information a curator can leverage to detect potential danger. Leveraging multiple data sources simultaneously allows better accuracy than applying each individually.

2.2. Connection Broker

The OBP service connection broker answers the query 'what connection parameters should be used to establish the best connection to interract with party X according to protocol Y. Where 'best' is determined by the Omnibroker which MAY take into account parameters specified by the relying party.

2.2.1. Service Connection Broker

The OBP service connection broker supports and extends the traditional DNS resolution service that resolves a DNS name (e.g. www.example.com) to return an IP address (e.g. 10.1.2.3).

When using an Omnibroker as a service connection broker, a client specifies both the DNS name (e.g. www.example.com) and the Internet protocol to be used (e.g. _http._tcp). The returned connection parameters MAY include:

The IP protocol version, address and port number to establish a connection to.

If appropriate, a security transport such as TLS or IPSEC.

If appropriate, a description of a service credential such as a TLS certificate or a constraint on the type of certificates that the client should consider acceptable.

If appropriate, application protocol details such as version and protocol options.

If an attempt to connect with the parameters specified fails, a client MAY report the failure and request a new set of parameters.

2.2.2. Peer Connection Broker

Each OBP request identifies both the account under which the request is made and the device from which it is made. An OBP broker is thus capable of acting as a peer connection broker service or providing a gateway to such a service.

When using Omnibroker as a peer connection broker, a client specifies the account name and DNS name of the party with which a connection is to be established (e.g. alice@example.com) and the connection protocol to be used (e.g. _xmpp-client._tcp)

The returned connection parameters are similar to those returned in response to a service broker query.

2.2.3. Connection Broker API

In the traditional BSD sockets API a network client performs a series of calls to resolve a network name to a list of IP addresses, selects one and establishes an IP connection to a port specified by the chosen application protocol.

OBP anticipates a higher level abstract API that encapsulates this complexity, hiding it from the application code.

In the case that one (or more) OBP services are configured, the library supporting the SHOULD obtain connection parameters from the OBP service. Otherwise, it SHOULD establish a connection using the traditional calling sequence of resolving a host name to obtain an IP address, etc.

2.3. Service Advertisement

Service advertisement is the converse of service resolution. An Internet application wishing to accept inbound connections specifies one or more sets of connection parameters for the Omnibroker to register with whatever naming, discovery or other services may be appropriate.

2.3.1. Connection Advertisement API

OBP anticipates the use of a high level API for connection advertisement that is the converse of the Connection broker API. Instead of establishing a connection, the API is used to advertise that a connection is offered either as a service or a peer.

An application MAY offer multiple types of connection with different connection properties (e.g. IPv4/IPv6, with and without SSL, etc.). This MAY be supported by marking certain properties as being optional and/or by permitting the API to be called multiple times with different properties specified.

2.4. Credential Validation Broker

A credential validation broker reports on the validity and trustworthiness of credentials presented to a client by Internet services and/or peers.

The service provided by OBP is similar to that provided by OCSP and SCVP. Like SCVP, OBP is an agent selected by the relying party to validate certificates and/or construct trust paths on its behalf.

2.5. Authentication Gateway

Every OBP request is strongly authenticated by means of a shared secret that is unique to the account and the device. This may be leveraged to permit use as an authentication gateway, providing access to other credentials that the client has established the right to use.

An Authentication Gateway MAY provide access to account names and passwords that the account holder has chosen to store in the cloud for access to sites that do not support a stronger, cryptographically based form of authentication such as OAuth.

2.6. Credential Announcement

An Authentication Gateway can only provide access to credentials that it has notice of. A client uses the Credential Announcement

transaction to advise the service of a new credential.

3. Omnibroker Connection Maintenance Service

3.1. Authentication

The principle function of the OBP Connection Maintenance Protocol is to establish and maintain the cryptographic parameters used to authenticate and encrypt

The user needs to authenticate the broker service regardless of any authentication requirements of the broker.

3.1.1. Broker Authentication

The OBP connection maintenance protocol transport MUST provide a trustworthy means of verifying the identity of the broker (e.g. an Extended Validation SSL certificate).

If the device supports a display capability, authentication of the device and user MAY be achieved by means of an authentication image. Such an authentication image is generated by the broker and passed to the client in the OpenResponse message. The user then verifies that the image presented on the device display is the same as that presented on the account maintenance console.

3.1.2. Device Authentication

If device authentication is required, the mechanism to be used depends on the capabilities of the device, the requirements of the broker and the existing relationship between the user and the broker.

If the device supports some means of data entry, authentication MAY be achieved by entering a passcode into the device that was previously delivered to the user out of band.

The passcode authentication mechanism allows the device to establish a proof that it knows the passcode without disclosing the passcode. This property provides protection against Man-In-The-Middle type disclosure attacks.

3.1.3. Illustrative example

Alice is an employee of Example Inc. which runs its own local omnibroker service 'example.com'. To configure her machine for use with this service, Alice contacts her network administrator who assigns her the account identifier 'alice' and obtains a PIN number

from the service 'Q80370-1RA606-F04B'

Alice enters the values 'alice@example.com' and 'Q80370-1RA606-F04B' into her Omnibroker-enabled Web browser.

The Web browser uses the local DNS to resolve 'example.com' and establishes a HTTPS connection to the specified IP address. The client verifies that the certificate presented has a valid certificate chain to an embedded trust anchor under an appropriate certificate policy (e.g. compliant with Extended Validation Criteria defined by CA-Browser Forum).

Having established an authenticated and encrypted TLS session to the Omnibroker service, the client sends an OpenRequest message to begin the process of mutual authentication. This message specifies the cryptographic parameters supported by the client (Authentication, Encryption) and a nonce value (Challenge), device identification parameters (DeviceID, DeviceURI, DeviceName) and the name of the account being requested.

The client does not specify the PIN code in the request, nor is the request authenticated. Instead the client informs the server that it has a PIN code that can be supplied if necessary.

```
Post / HTTP/1.1
Host: example.com
Cache-Control: no-store
Content-Type: Application/json;charset=UTF-8
Content-Length: 470

{
  "OpenRequest": {
    "Encryption": ["HS256",
      "HS384",
      "HS512",
      "HS256T128"],
    "Authentication": ["A128CBC",
      "A256CBC",
      "A128GCM",
      "A256GCM"],
    "Account": "alice",
    "Domain": "example.com",
    "HavePasscode": true,
    "HaveDisplay": true,
    "Challenge": "d2gdVeQesS3UTOgtK4JSEg==",
    "DeviceID": "Serial:0002212",
    "DeviceURI": "http://comodo.com/dragon/v3.4",
    "DeviceName": "Comodo Dragon"}}
```

The service receives the request. If the request is consistent with the access control policy for the server it returns a reply that specifies the chosen cryptographic parameters (Cryptographic), responds to the client issued by the client to establish server proof of knowsledge of the PIN (ChallengeResponse) and issues a challenge to the client (Challenge).

The cryptographic parameters specify algorithms to be used for encryption and authentication, a shared secret and a ticket value. Note that while the shared secret is exchanged in plaintext form in the HTTP binding, the connection protocol MUST provide encryption.

```
HTTP/1.1 203 Passcode
Content-Type: application/json;charset=UTF-8
Content-Length: 500

{
  "OpenResponse": {
    "Status": 203,
    "StatusDescription": "Passcode",
    "Cryptographic": [{
        "Secret": "11bmdFi9Et7KIUg8aeN2AQ==",
        "Encryption": "A128CBC",
        "Authentication": "HS256",
        "Ticket":
        "TUMnorO0SjHHS7D2uFcGlRYJ0Hd3eibwe0ogptoNMQuCYmCHfHAJcJlyvi
        j8WoXDglTSOkctnmoBzl8W0NLSlcgSyZcmsAyoWs8y1Rn2ZlO2WBgoWrFIO
        qPa4oB29dgs/ei6ieINZtmvXNCm2NUkWA=="}],
    "Challenge": "alX8aAWH6acSqO3FTT94HA==",
    "ChallengeResponse": "enT5myMw8w2hV4H32Ntx/g=="}}
```

To complete the transaction, the client sends a TicketRequest message to the serice containing a response to the PIN challenge sent by the service (ChallengeResponse). The TicketRequest message is authenticated under the shared secret specified in the OpenResponse message.

```
Post / HTTP/1.1
Host: example.com
Cache-Control: no-store
Content-Type: Application/json;charset=UTF-8
Content-Length: 78
Content-Integrity:
  mac=cjkMkfnnYP8JYWZAbRLvtpqImmOK3rsrOT1XcvAgHDk=;
  ticket=TUMnorO0SjHHS7D2uFcGlRYJ0Hd3eibwe0ogptoNMQuCYmCHfHAJcJlyvi
    j8WoXDglTSOkctnmoBzl8W0NLSlcgSyZcmsAyoWs8y1Rn2ZlO2WBgoWrFIOqPa4
    oB29dgs/ei6ieINZtmvXNCm2NUkWA==

{
  "TicketRequest": {
    "ChallengeResponse": "TctLOG74cwpm26YNpEibcQ=="}}
```

If the response to the PIN challenge is correct, the service responds with a message that specifies a set of cryptographic parameters to be used to authenticate future interactions with the service (Cryptographic) and a set of connection parameters for servers supporting the Query Service (Service).

In this case the server returns three connections, each offering a different transport protocol option. Each connection specifies its own set of cryptographic parameters (or will when the code is written for that).

```
HTTP/1.1 200 Complete
Content-Type: application/json;charset=UTF-8
Content-Length: 1907
Content-Integrity:
  mac=nKhjR1r2eYPga0rmDfHT4HOvgQ+EuUoQPwzIl0btljs=;
  ticket=TUMnorO0SjHHS7D2uFcGlRYJ0Hd3eibwe0ogptoNMQuCYmCHfHAJcJlyvi
    j8WoXDglTSOkctnmoBzl8W0NLSlcgSyZcmsAyoWs8y1Rn2ZlO2WBgoWrFIOqPa4
    oB29dgs/ei6ieINZtmvXNCm2NUkWA==

{
  "TicketResponse": {
    "Status": 200,
    "StatusDescription": "Complete",
    "Cryptographic": [{
        "Protocol": "OBPConnection",
        "Secret": "HQuQg4GkzTwTVoGxar0EXg==",
        "Encryption": "A128CBC",
        "Authentication": "HS256",
        "Ticket":
        "0ulMVMMfY/pLHZ0FlIy2zDnNycYz9Znvs3JJYQGlZ+dWaxMNxm/jLEsJd/
        0qsAc5qp8fjBoMN49V9DkDgM4UYJxVriqfr64RyTTgug2taHY="}],
    "Service": [{
```

```
            "Name": "obp1.example.com",
            "Port": 443,
            "Address": "10.1.2.3",
            "Priority": 1,
            "Weight": 100,
            "Transport": "WebService",
            "Cryptographic": {
              "Protocol": "OBPQuery",
              "Secret": "kezeXxhkzXgxY7vpkHUb1g==",
              "Encryption": "A128CBC",
              "Authentication": "HS256",
              "Ticket":
              "jpBXvI7/0WTmwI2NN4n7Vvw96nbS9LpSsSNMIkdapiUoLikSkjpgMrtb
              VKz5lHOPloCgAyZXxfZpQRsp4oPY4BcRaMw6F5na62IHnBVDeXg="}},
          {
            "Name": "dns1.example.com",
            "Port": 53,
            "Address": "10.1.2.2",
            "Priority": 1,
            "Weight": 100,
            "Transport": "DNS",
            "Cryptographic": {
              "Protocol": "OBPQuery",
              "Secret": "Wk3m7DlL/GStBBm3xUjyzg==",
              "Encryption": "A128CBC",
              "Authentication": "HS256",
              "Ticket":
              "Q9r4hXefHhLvgpKHVg3w2p7VptVH9qidGiIa4Nw3Zp5hZR816h9+PRj5
              sye1jmIhy4sYA/jfK/g4OrSngK9xWO7Qg3/iQ+YTAchKQjdJtN4="}},
          {
            "Name": "udp.example.com",
            "Port": 5000,
            "Address": "10.1.2.2",
            "Priority": 1,
            "Weight": 100,
            "Transport": "UDP",
            "Cryptographic": {
              "Protocol": "OBPQuery",
              "Secret": "wBiguG9FGj08nS/c/njp4Q==",
              "Encryption": "A128CBC",
              "Authentication": "HS256",
              "Ticket":
              "F8LPKTL+XaAX0eJsM22fdJ37BRS816dKXD66UbD8NAVKOgOu556uS8WW
              AMj+dJbJaErUzo/vw7tY0icCu1bw8qHmOO4gzhbSbD4Nga2EAU4="}}]}
              }
```


When Alice's machine is to be transfered to another employee, the Unbind transaction is used. The only parameter is the Ticket

identifying the device association (Ticket).

```
Post / HTTP/1.1
Host: example.com
Cache-Control: no-store
Content-Type: Application/json;charset=UTF-8
Content-Length: 25
Content-Integrity:
  mac=bZU61eCOW4nVnfdJNS719HL4IsNVxtoTgoRt+mqLbWY=;
  ticket=0ulMVMMfY/pLHZ0FlIy2zDnNycYz9Znvs3JJYQGlZ+dWaxMNxm/jLEsJd/
    0qsAc5qp8fjBoMN49V9DkDgM4UYJxVriqfr64RyTTgug2taHY=

{
  "UnbindRequest": {}}
```

Since the unbind response represents the termination the relationship with the Omnibroker, the response merely reports the success or failure of the request.

```
HTTP/1.1 0
Content-Type: application/json;charset=UTF-8
Content-Length: 26
Content-Integrity:
  mac=9P1FmroeFU7y9qHgXdSFXH2qSImh0cQpaSgZrx5IswM=;
  ticket=0ulMVMMfY/pLHZ0FlIy2zDnNycYz9Znvs3JJYQGlZ+dWaxMNxm/jLEsJd/
    0qsAc5qp8fjBoMN49V9DkDgM4UYJxVriqfr64RyTTgug2taHY=

{
  "UnbindResponse": {}}
```

The 'Ticket' value presented in the foregoing examples is a sequence of binary data generated by the service is opaque to the client. Services MAY generate ticket values with a substructure that enable the service to avoid the need to maintain server side state.

In the foregoing example, the ticket structures generated by the service encode the cryptographic parameter data, the shared secret, account identifier and an authentication value. The initial ticket value generated additionally encodes the values of the client and service challeng values for use in calculating the necessary ChallengeResponse.

3.2. OBPConnection

3.2.1. Message: Message

3.2.2. Message: Response

Status : Integer [0..1]

Application layer server status code

StatusDescription : String [0..1]

Describes the status code (ignored by processors)

3.2.3. Message: ErrorResponse

An error response MAY be returned in response to any request.

Note that requests MAY be rejected by the code implementing the transport binding before application processing begins and so a server is not guaranteed to provide an error response message.

3.2.4. Message: Request

Ticket : Binary [1..1]

Opaque ticket issued by the server that identifies the cryptographic parameters for encryption and authentication of the message payload.

3.2.5. Structure: Cryptographic

Parameters describing a cryptographic context.

Protocol : Label [0..1]

OBP tickets MAY be restricted to use with either the management protocol (Management) or the query protocol (Query). If so a service would typically specify a ticket with a long expiry time or no expiry for use with the management protocol and a separate ticket for use with the query protocol.

Secret : Binary [1..1]

Shared secret

Encryption : Label [1..1]

Encryption Algorithm selected

Authentication : Label [1..1]

Authentication Algorithm selected

Ticket : Binary [1..1]

Opaque ticket issued by the server that identifies the cryptographic parameters for encryption and authentication of the message payload.

Expires : DateTime [0..1]

Date and time at which the context will expire

3.2.6. Structure: ImageLink

Algorithm : Label [0..1]

Image encoding algorithm (e.g. JPG, PNG)

Image : Binary [0..1]

Image data as specified by algorithm

3.2.7. Structure: Connection

Contains information describing a network connection.

Name : Name [0..1]

DNS Name. Since one of the functions of an OBP service is name resolution, a DNS name is only used to establish a connection if connection by means of the IP address fails.

Port : Integer [0..1]

TCP or UDP port number.

Address : String [0..1]

IPv4 (32 bit) or IPv6 (128 bit) service address

Priority : Integer [0..1]

Service priority. Services with lower priority numbers SHOULD be attempted before those with higher numbers.

Weight : Integer [0..1]

Weight to be used to select between services of equal priority.

Transport : Label [0..1]

OBP Transport binding to be used valid values are HTTP, DNS and UDP.

Expires : DateTime [0..1]

Date and time at which the specified connection context will expire.

3.2.8. Bind

Binding a device is a two step protocol that begins with the Start Query followed by a sequence of Ticket queries.

3.2.9. Message: BindRequest

The following parameters MAY occur in either a StartRequest or TicketRequest:

Encryption : Label [0..Many]

Encryption Algorithm that the client accepts. A Client MAY offer multiple algorithms. If no algorithms are specified then support for the mandatory to implement algorithm is assumed. Otherwise mandatory to implement algorithms MUST be specified explicitly.

Authentication : Label [0..Many]

Authentication Algorithm that the client accepts. If no algorithms are specified then support for the mandatory to implement algorithm is assumed. Otherwise mandatory to implement algorithms MUST be specified explicitly.

3.2.10. Message: BindResponse

The following parameters MAY occur in either a StartResponse or TicketResponse:

Cryptographic : Cryptographic [0..Many]

Cryptographic Parameters.

Service : Connection [0..Many]

A Connection describing an OBP service point

3.2.11. Message: OpenRequest

The OpenRequest Message is used to begin a device binding transaction. Depending on the authentication requirements of the service the transaction may be completed in a single query or require a further Ticket Query to complete.

If authentication is required, the mechanism to be used depends on the capabilities of the device, the requirements of the broker and the existing relationship between the user and the broker.

If the device supports some means of data entry, authentication MAY be achieved by entering a passcode previously delivered out of band into the device.

The OpenRequest specifies the properties of the service (Account, Domain) and Device (ID, URI, Name) that will remain constant throughout the period that the device binding is active and parameters to be used for the mutual authentication protocol.

Account : String [0..1]

Account name of the user at the OBP service

Domain : Name [0..1]

Domain name of the OBP broker service

HavePasscode : Boolean [0..1] Default =False

If 'true', the user has entered a passcode value for use with passcode authentication.

HaveDisplay : Boolean [0..1] Default =False

Specifies if the device is capable of displaying information to the user or not.

Challenge : Binary [0..1]

Client challenge value to be used in authentication challenge

DeviceID : URI [0..1]

Device identifier unique for a particular instance of a device such as a MAC or EUI-64 address expressed as a URI

DeviceURI : URI [0..1]

Device identifier specifying the type of device, e.g. an xPhone.

DeviceName : String [0..1]

Descriptive name for the device that would distinguish it from other similar devices, e.g. 'Alice's xPhone".

3.2.12. Message: OpenResponse

An Open request MAY be accepted immediately or be held pending completion of an inband or out-of-band authentication process.

The OpenResponse returns a ticket and a set of cryptographic connection parameters in either case. If the

Challenge : Binary [0..1]

Challenge value to be used by the client to respond to the server authentication challenge.

ChallengeResponse : Binary [0..1]

Server response to authentication challenge by the client

VerificationImage : ImageLink [0..Many]

Link to an image to be used in an image verification mechanism.

3.2.13. Message: TicketRequest

The TicketRequest message is used to (1) complete a binding request begun with an OpenRequest and (2) to refresh ticket or connection parameters as necessary.

ChallengeResponse : Binary [0..1]

The response to a server authentication challenge.

3.2.14. Message: TicketResponse

The TicketResponse message returns cryptographic and/or connection context information to a client.

3.2.15. Unbind

Requests that a previous device association be deleted.

3.2.16. Message: UnbindRequest

Since the ticket identifies the binding to be deleted, the only thing that the unbind message need specify is that the device wishes to cancel the binding.

3.2.17. Message: UnbindResponse

Reports on the success of the unbinding operation.

If the server reports success, the client SHOULD delete the ticket and all information relating to the binding.

A service MAY continue to accept a ticket after an unbind request has been granted but MUST NOT accept such a ticket for a bind request.

4. Omnibroker Query Service

4.1. Illustrative example

[For illustrative purposes, all the examples in this section are shown using the Web Services Transport binding. Examples of other transport bindgins are shown in section [TBS].]

The Alice of the previous example uses her Web browser to access the URL www.example.com://www.example.com/. Assuming this was done while the prior binding was still active (i.e. before the UnbindRequest message was sent), the Web browser would send a QueryConnectRequest request to obtain the best connection parameters for the http protocol at www.example.com:

```
Post / HTTP/1.1
Host: example.com
Cache-Control: no-store
Content-Type: Application/json;charset=UTF-8
Content-Length: 131
Content-Integrity:
  mac=9ZSkLYKFMYenvqt/MwkAtvetqvM7Nydh6Rc2bvbKTbM=;
  ticket=jpBXvI7/0WTmwI2NN4n7Vvw96nbS9LpSsSNMIkdapiUoLikSkjpgMrtbVK
    z5lHOPloCgAyZXxfZpQRsp4oPY4BcRaMw6F5na62IHnBVDeXg=

{
  "QueryConnectRequest": {
    "Identifier": {
      "Name": "http",
      "Service": "www.example.com",
      "Port": 80}}}
```

The service responds with an ordered list of possible connections. In this case the site is accessible via plain TCP transport or with TLS. Since TLS is the preferred protocol, that connection is listed first.

```
HTTP/1.1 200
Content-Type: application/json;charset=UTF-8
Content-Length: 347
Content-Integrity:
  mac=1oFa8fNsRbKiCCnwd4feSqq+h/by+tCLbw2bzf235TU=;
  ticket=jpBXvI7/0WTmwI2NN4n7Vvw96nbS9LpSsSNMIkdapiUoLikSkjpgMrtbVK
    z5lHOPloCgAyZXxfZpQRsp4oPY4BcRaMw6F5na62IHnBVDeXg=

{
  "QueryConnectResponse": {
    "Status": 200,
    "Connection": [{
        "IPAddress": "10.3.2.1",
        "IPPort": 443,
        "Transport": "TLS",
        "TransportPolicy": "TLS=Optional",
        "ProtocolPolicy": "Strict"},
      {
        "IPAddress": "10.3.2.1",
        "IPPort": 80,
        "ProtocolPolicy": "Strict"}]}}
```

Although the QueryConnectResponse returned the hash of a PKIX certificate considered valid for that connection, the server returns a different certificate which the client verifies using the ValidateRequest query.

```
Post / HTTP/1.1
Host: example.com
Cache-Control: no-store
Content-Type: Application/json;charset=UTF-8
Content-Length: 124
Content-Integrity:
  mac=S+xlW2U6z1REQmbNHiOnAw4xpUXP8wJXZiCJzMzQelc=;
  ticket=jpBXvI7/0WTmwI2NN4n7Vvw96nbS9LpSsSNMIkdapiUoLikSkjpgMrtbVK
    z5lHOPloCgAyZXxfZpQRsp4oPY4BcRaMw6F5na62IHnBVDeXg=

{
  "ValidateRequest": {
    "Credential": {
      "Type": "application/x-x509-server-cert",
      "Data": "AAECAwQ="}}}
```

The service validates the certificate according to the Omnibroker service policy.

```
HTTP/1.1 200
Content-Type: application/json;charset=UTF-8
Content-Length: 47
Content-Integrity:
  mac=jrsfxojksHBVs1WWxVbX3nn+CaIIix2JrrTTQn0X43k=;
  ticket=jpBXvI7/0WTmwI2NN4n7Vvw96nbS9LpSsSNMIkdapiUoLikSkjpgMrtbVK
    z5lHOPloCgAyZXxfZpQRsp4oPY4BcRaMw6F5na62IHnBVDeXg=

{
  "ValidateResponse": {
    "Status": 200}}
```

4.2. OBPQuery

4.2.1. Message: Payload

4.2.2. Message: Message

4.2.3. Message: Request

Every query request contains the following common elements:

Index : Integer [0..1]

Index used to request a specific response when multiple responses are available.

4.2.4. Message: Response

Every Query Response contains the following common elements:

Status : Integer [1..1]

Status return code value

Index : Integer [0..1]

Index of the current response.

Count : Integer [0..1]

Number of responses available.

4.2.5. Structure: Identifier

Specifies an Internet service by means of a DNS address and either a DNS service prefix, an IP port number or both. An Internet peer connection MAY be specified by additionally specifying an account.

Name : Name [1..1]

The DNS name of the service to connect to.

Internationalized DNS names MUST be encoded in punycode encoding.

Account : Label [0..1]

Identifies the account to connect to in the case that a peer connection is to be established.

Service : Name [0..1]

The DNS service prefix defined for use with DNS records that take a service prefix including SRV.

Port : Integer [0..1]

IP Port number.

A service identifier MUST specify either a service or a port or both.

4.2.6. Structure: Connection

IPVersion : Integer [0..1]

Contains the IP version field. If absent, IPv4 is assumed.

IPProtocol : Integer [0..1]

Contains the IP protocol field. If absent, TCP is assumed.

IPAddress : Binary [0..1]

IP address in network byte order. This will normally be an IPv4 (32 bit) or IPv6 (128 bit) address.

IPPort : Integer [0..1]

IP port. 1-65535

TransportPolicy : String [0..1]

Transport security policy as specified in [TBS]

ProtocolPolicy : String [0..1]

Application security policy specification as specified by the application protocol.

Advice : Advice [0..1]

Additional information that a service MAY return to support a service connection identification.

4.2.7. Structure: Advice

Additional information that a service MAY return to support a service connection identification. For example, DNSSEC signatures chains, SAML assertions, DANE records, Certificate Transparency proof chains, etc.

Type : Label [0..1]

The IANA MIME type of the content type

Data : Binary [0..1]

The advice data.

4.2.8. Structure: Service

Describes a service connection

Identifier : Identifier [0..Many]

Internet addresses to which the service is to be bound.

Connection : Connection [0..1]

Service connection parameters.

4.2.9. QueryConnect

Requests a connection context to connect to a specified Internet service or peer.

4.2.10. Message: QueryConnectRequest

Specifies the Internet service or peer that a connection is to be established to and the acceptable security policies.

Identifier : Identifier [0..1]

Identifies the service or peer to which a connection is requested.

Policy : Label [0..Many]

Acceptable credential validation policy.

ProveIt : Boolean [0..1]

If set the broker SHOULD send advice to permit the client to validate the proposed connection context.

4.2.11. Message: QueryConnectResponse

Returns one or more connection contexts in response to a QueryConnectRequest Message.

Connection : Connection [0..Many]

An ordered list of connection contexts with the preferred connection context listed first.

Advice : Advice [0..1]

Proof information to support the proposed connection context.

Policy : Label [0..Many]

Policy under which the credentials have been verified.

4.2.12. Advertise

Advises a broker that one or more Internet services are being offered with particular attributes.

4.2.13. Message: AdvertiseRequest

Specifies the connection(s) to be established.

The attributes required depend on the infrastructure(s) that the broker is capable of registering the service with.

Service : Service [0..Many]

Describes a connection to be established.

4.2.14. Message: AdvertiseResponse

Specifies the connection(s)

Service : Service [0..Many]

Describes a connection that was established.

4.2.15. Validate

The Validate query requests validation of credentials presented to establish a connection. For example credentials presented by a server in the process of setting up a TLS session.

4.2.16. Message: ValidateRequest

Specifies the credentials to be validated and the purpose for which they are to be used.

Service : Service [0..1]

Describes the service for which the credentials are presented for access.

Credential : Credential [0..1]

List of credentials for which validation is requested.

Policy : Label [0..Many]

Policy under which the credentials have been verified.

4.2.17. Message: ValidateResponse

Reports the status of the credential presented.

Policy : Label [0..Many]

Policy under which the credentials have been verified.

4.2.18. QueryCredentialPassword

The QueryCredentialPassword query is used to request a password credential that the user has previously chosen to store at the broker.

4.2.19. Message: CredentialPasswordRequest

Requests a password for the specified account.

Account : String [0..1]

The account for which a password is requested.

4.2.20. Message: CredentialPasswordResponse

Returns a password for the specified account.

Password : String [0..1]

The requested password.

5. Mutual Authentication

A Connection Service MAY require that a connection request be authenticated. Three authentication mechanisms are defined.

PIN Code: The client and server demonstrate mutual knowledge of a PIN code previously exchanged out of band.

Established Key: The client and server demonstrate knowledge of the private key associated with a credential previously established. This MAY be a public key or a symmetric key.

Out of Band Confirmation: The request for access is forwarded to an out of band confirmation service.

5.1. PIN Authentication

[Motivation]

Although the PIN value is never exposed on the wire in any form, the protcol considers the PIN value to be a text encoded in UTF8 encoding.

[Considerations for PIN character set choice discussed in body of draft, servers MUST support numeric only, clients SHOULD support full Unicode]

The PIN Mechanism is a three step process:

The client sends an OpenRequest message to the Service containing a challenge value CC.

The service returns an OpenResponse message containing to the client a server challenge value SV and a server response value SR.

The client sends a TicketRequest message to the service containing a client response value CR.

Since no prior authentication key has been the OpenRequest and OpenResponse messages are initially sent without authentication and authentication values established the Challenge-Response mechanism.

The Challenge values CC, and SC are cryptographic nonces. The nonces SHOULD be generated using an appropriate cryptographic random source. The nonces MUST be at least as long as 128 bits, MUST be at least the minimum key size of the authentication algorithm used and MUST NOT more than 640 bits in length (640 bits should be enough for anybody).

The server response and client response values are generated using an authentication algorithm selected by the server from the choices proposed by the client in the OpenRequest message.

The algorithn chosen may be a MAC algorithm or an encrypt-with-

authentication (EWA) algorithm. If an EWA is specified, the encrypted data is discarded and only the authentication value is used in its place.

Let A(d,k) be the authentication value obtained by applying the authentication algorithm with key k to data d.

To create the Server Response value, the UTF8 encoding of the PIN value 'P' is first converted into a symmetric key KPC by using the Client challenge value as the key truncating if necessary and then applied to the of the OpenRequest message:

KPC = A (PIN, CC) SR = A (Secret + SC + OpenRequest, KPC)

In the Web Service Binding, the Payload of the message is the HTTP Body as presented on the wire. The Secret and Server Challenge are presented in their binary format and the '+' operator stands for simple concatenation of the binary sequences.

This protocol construction ensures that the party constructing SR:

- Knows the PIN code value (through the construction of KPC).
- Is responding to the Open Request Message (SR depends on OpenRequest).
- Has knowlege of the secret key which MUST be used to authenticate the following TicketRequest/TicketResponse interaction that will establish the actual connection.
- Does not provide an oracle the PIN value. That is, the protocol does not provide a service that reveals the (since the value SR includes the value SC which is a random nonce generated by the server and cannot be predicted by the client).

To create the Client Response value, secret key is applied to the PIN value and server Challenge:

CR = A (PIN + SC + OpenRequest, Secret)

Note that the server can calculate the value of the Client Response token at the time that it generates the Server Challenge. This minimizes the amount of state that needs to be carried from one request to the next in the Ticket value when using the stateless server implementation described in section Section 5.4

This protocol construction ensures that the of CR

Knows the PIN value.

Is respoding to the OpenResponse generated by the server.

Note that while disclosure of an oracle for the PIN value is a concern in the construction of CR, this is not the case in the construction of SR since the client has already demonstrated knowledge of the PIN value.

5.2. Example: Latin PIN Code

The Connection Request example of section Section 3.1.3 demonstrates the use of an alphanumeric PIN code using the Latin alphabet.

The PIN code is [] and the authentication algorithm is []. The value KP is thus:

[TBS]

The data over which the hash value is calulated is Secret + SC + OpenRequest:

[TBS]

Applying the derrived key to the data produces the server response:

The data for the client response is PIN + SC:

[TBS]

Applying the secret key to the data produces the client response:

[TBS]

5.3. Example: Cyrillic PIN Code

If the PIN code in the earlier example was [] the value KP would be:

[TBS]

The Server Response would be:

[TBS]

The rest of the protocol would then continue as before.

5.4. Stateless server

The protocol is designed to permit but not require a stateless implementation by the server using the Ticket value generated by the server to pass state from the first server transaction to the second.

In the example shown in Section 3.1.3, the server generates a 'temporary ticket' containing the following information:

If a server uses the Ticket to transmit state in this way it MUST protect the confidentiality of the ticket using a strong means of encryption and authentication.

5.5. Established Key

The Established Key mechanism is used when the parties have an existing shared key or public key credential.

The [Open request open response are authenticated under the respective keys]

SR=CC, CR=SC

5.6. Out of Band Confirmation

The Out Of Band Confirmation mechanism is a three step process in which:

- The client makes an OpenRequest message to the service and obtains an OpenResponse message.
- The service is informed that the service has been authorized through an out of band process.
- The client makes a TicketRequest to the service and obtains a TicketResponse message to complete the exchange.

Since no prior authentication key has been the OpenRequest and OpenResponse messages are sent without authentication.

The principal concern in the Out Of Band Confirmation mechanism is ensuring that the party authorizing the request is able to identify which party originated the request they are attempting to identify.

If a device has the ability to display an image it MAY set the HasDisplay=true in the OpenRequest message. If the broker recieves an OpenRequest with the HasDisplay value set to true, the OpenResponse MAY contain one or more VerificationImage entries

specifying image data that is to be displayed to the user by both the client and the confirmation interface.

Before confirming the request, the user SHOULD verify that the two images are the same and reject the request in the case that they are not.

Many devices do not have a display capability, in particular an embedded device such as a network switch or a thermostat. In this case the device MAY be identified by means of the information provided in DeviceID, DeviceURI, DeviceImage and DeviceName.

6. Transport Bindings

To achieve an optimal balance of efficiency and availability, three transport bindings are defined:

Supports all forms of OBP transaction in all network environments.

Provides efficient support for a subset of OBP query transactions that is accessible in most network environments.

Provides efficient support for all OBP query transactions and is accessible in most network environments.

Support for the HTTP over TLS binding is REQUIRED.

An OBP message consists of three parts:

Ticket [As necessary] If specified, identifies the cryptographic key and algorithm parameters to be used to secure the message payload.

Payload [Required] If the ticket context does not specify use of an encryption algorithm, contains the message data. Otherwise contains the message data encrypted under the encryption algorithm and key specified in the ticket context.

Authenticator [Optional] If the ticket context specifies use of a Message Authentication Code (MAC), contains the MAC value calculated over the payload data using the authentication key bound to the ticket.

Note that although each of the transport bindings defined in this specification entail the use of a JSON encoding for the message data, this is not a necessary requirement for a transport binding.

6.1. HTTP over TLS

OBP requests and responses are mapped to HTTP POST requests and responses respectively. Java Script Object Notation (JSON) encoding is used to encode requests and responses.

6.1.1. Message Encapsulation

Requests and responses are mapped to HTTP POST transactions. The content of the HTTP message is the message payload. The Content-Type MUST be specified as application/json. The Character set MUST be specified as UTF-8.

The Ticket and Authenticator are specified using the Integrity header as follows:

Integrity: <base64 (authenticator)> ; ticket=<base64 (ticket)>

6.1.2. Example

[To be generated from spec]

6.2. DNS Tunnel

The DNS Tunnel mode of operation makes use of DNS TXT resource record requests and responses to tunnel OBP Query requests. Due to the constraints of this particular mode of operation, use of this transport is in practice limited to supporting transactions that can be expressed within 500 bytes. These include the QueryConnect and ValidateRequest interactions.

6.2.1. Request

Requests are mapped to DNS TXT queries. The request is mapped onto the DNS name portion of the query by encoding the Ticket, Authenticator and JSON encoded Payload using Base32 encoding and appending the result to the service prefix to create a DNS name as follows:

<base32(payload)>.<base32(authenticator)>.<base32(ticket)>.Suffix

The payload MAY be split across multiple DNS labels at any point.

6.2.2. Response

Responses are mapped to DNS TXT records by encoding the Authenticator and JSON encoded Payload using Base64 encoding and cocatenating the result with a periods as a separator as follows:

<base32(payload)>.<base32(authenticator)>

6.2.3. Example

[To be generated from spec]

6.3. UDP

The UDP transport MAY be used for transactions where the request fits into a single UDP packet and the response can be accomadated in 16 UDP packets. As with the Web Service Binding, Java Script Object Notation (JSON) encoding is used to encode requests and responses.

6.3.1. Request

The request consists of four message segments containing a Header, Ticket, Payload and Authenticator. Each message segment begins with a two byte field that specified the length of the following data segment in network byte order. The Payload is encoded in JSON encoding and the remaining fields as binary data without additional encoding.

The header field for this version of the protocol (1.0) contains two bytes that specify the Major and Minor version number of the transport protocol being 1 and 0 respectively. Future versions of the transport protocol MAY specify additional data fields.

[TBS diagram]

6.3.2. Response

The response consists of a sequence of packets. Each packet consists of a header section and a data section.

The header section consists of a two byte length field followed by two bytes that speciofy the Major and Minor version number of the transport protocol (1 and 0), two bytes that specify the frame number and the total number of frames and two bytes that specify the message identifier.

[TBS diagram]

[Question, should the authenticator be over the whole message or should each packet have its own authenticator?]

6.3.3. Example

[To be generated from spec]

7. Acknowledgements

[List of contributors]

8. Security Considerations

8.1. Denial of Service

8.2. Breach of Trust

8.3. Coercion

9. To do

The specification should define and use a JSON security object.

Formatting of the abstract data items needs to be improved

Need to specify the UDP transport binding

Should specify how each data item is represented in JSON format somewhere. This is obvious for some of the data types but needs to be fully specified for things like DateTime.

Run the code to produce proper examples.

Write a tool to transclude the example and other xml data into the document source.

Fully document the API section.

10. For discussion.

Should the specification use the form urlencoded convention like OAUTH does?

How should responses be cryptographically linked to requests?

11. IANA Considerations

[TBS list out all the code points that require an IANA registration]

12. Normative References

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC4366] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, April 2006.

[X.509] International Telecommunication Union, "ITU-T Recommendation X.509 (11/2008): Information technology - Open systems interconnection - The Directory: Public-key and attribute certificate frameworks", ITU-T Recommendation X.509, November 2008.

[X.680] International Telecommunication Union, "ITU-T Recommendation X.680 (11/2008): Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, November 2008.

Appendix A. Example Data.

A.1. Ticket A

A.2. Ticket B

Author's Address

Phillip Hallam-Baker
Comodo Group Inc.

Email: philliph@comodo.com