Echo

Echo
November 2010

Backplane Identity Scenario

Abstract

This documents specifies an application of Backplane to a scenario involving authentication and identity information passing.

Table of Contents

1. Introduction

While Backplane is a generic framework for message exchange between trusted applications in the context of a browser session, the Identity Scenario is a specific protocol that facilitates authentication and identity transactions between those applications.

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. Representation Of Identity Information

User identity information communicated between Backplane parties in this scenario is represented in Portable Contacts [PortableContacts] (PoCo) format with some restrictions and extensions.

PoCo data is presented in the JSON [RFC4627] format.

Since the scenario deals with a single user (the one performing identity-related actions in the context of a concrete browsing session), PoCo response node MUST contain a single "entry" object.

The "entry" object MUST contain the "accounts" element with at least one item. Each of the account items represents an identity.

An identity is represented by "identityUrl" element, which can be either a plain URL [RFC1738] of user identity, or normalized SGN identity URI [RFC3986] constructed from domain/userid pair by putting these values together: `sgn://<DOMAIN>/?ident=<USERID>`. In the latter case SGN identity URI MUST be recognized by Google's SGNodeMapper [Google.SGNodeMapper].

Here is an example of a valid PoCo entry that may be transferred in a Backplane message:

```
{
    "startIndex": 0,
    "itemsPerPage": 1,
    "totalResults": 1,
    "entry": {
        "accounts": [
            {
                "identityUrl" : "http://twitter.com/johndoe",
                "username": "johndoe",
                "emails": [{
                    "value": "username@email.com",
                    "primary": "true"
                }],
                "photos": [{
                    "value": "http://img.twitter.com/johndoe.jpg"
                    "type": "avatar"
                }]
            },
            {
                "identityUrl": "http://example.com/user/johndoe"
            },
            {
                "identityUrl": "sgn://livejournal.com/?ident=johndoe"
            }
        ]
    }
}
```

3. Message Types

Below is a list of Backplane message types employed in this scenario.

3.1 identity/login

This message is transmitted whenever an application (e.g. an Identity Manager) on a Backplane channel authenticates a user against an identity provider (e.g. Twitter or Facebook). The only party that is allowed to post this message is the one that directly facilitated the authentication (login) process (e.g. an Identity Connector).

The payload of the 'identity/login' message is a JSON object which contains the following fields:

context	the URL of the page where the authentication took place
identities	a PoCo object listing the identities that the user has gained by authenticating

Note that 'identity/login' messages are incremental, communicating addition of the specified identities to the current user session.

3.2 identity/logout

This message is transmitted whenever an application (e.g. an Identity Manager) on a Backplane channel logs the user out of an identity provider (e.g. Twitter or Facebook). The only party that is allowed to post this message is the one that directly facilitated the logout process (e.g. an Identity Connector).

The payload of the 'identity/logout' message is a JSON object which contains the following fields:

context	the URL of the page where the logout took place
identities	a PoCo object listing the identities that the user has dropped with the logout

3.3 identity/ack

This message MAY be transmitted by a Backplane-enabled party to notify others that it has processed a recent login or logout event (communicated via 'identity/login' or 'identity/logout' messages respectively).

The structure of the message payload is sender-specific. Recipients are expected to check the 'source' field of the message and parse the payload accordingly. It is beyond this document to map sender identifiers to the payload structure.

To give an example, a server might send the 'identity/ack' message to notify related widgets that a user has logged in (the server has processed 'identity/login' message sent by another party). The corresponding session information could be piggybacked into the message payload or the widgets might respond to the 'identity/ack' message by issuing a separate request to fetch it. The widgets will ignore all broadcast 'identity/ack' messages except for the one containing the known sender id.

4. Use Cases

The following use cases illustrate the process highlighting implementation details of the Echo service, but it is important to note that Echo is not in any way a 'special' party. Any other similar service can employ the Backplane Identity Scenario without a need to inform or modify other parties.

4.1 Interacting Parties

The use cases mentioned below make use of the following interacting parties:

- A customer, serving content off http://customer.com.
- An independent Backplane server (running CNAMEd to backplane.customer.com), serving the customer's bus.
- Widgets on the customer's page:
  - Echo Stream Client (ESC)
  - Identity Manager (IM)
  - Login Status Widget (LSW)
- Identity Connector idcon.com, acting as a server-side counterpart of the IM.
- Echo server components:
  - an API endpoint for Echo widgets on the customer's page (available behind echoapi.customer.com CNAME).
  - an API endpoint for server-to-server communication (available as api.aboutecho.com).
  - a bus consumer - a server component listening to all messages broadcast on the customer's bus.
- Echo Submission Proxy (Section 4.1.2)

All the parties are configured to use the same Backplane bus "customer.com". The configuration includes merely stating the Backplane server address and bus identifier for widgets (operating in read-only mode). Parties that post data (e.g. the Identity Connector) or listen to the entire Backplane bus (e.g. the Echo bus consumer) also need to have a username/password set up with the Backplane server to perform secure HTTPS requests with basic HTTP authentication.

4.1.1 Identity Manager

Identity Manager (IM) is an Echo-specific widget that allows user to manage their identity. IM is not required to be a part of the workflow and is described here just to aid in understanding of the use cases.

Once the page loads, Identity Manager fetches a list of supported identity providers (e.g. Twitter, Facebook, Customer's Own Login System) from its per-domain configuration. Each item of the list contains:

- Identity Provider's favicon.
- Identity Provider's name ("Twitter", "Facebook").
- A URL of an Identity Connector that facilitates logins/logouts for the Identity Provider in this context (e.g. for this customer).

This way a customer may mix and match which Identity Connectors are used to deal with specific Identity Providers.

Identity Manager presents the Identity Provider list to the user for selection. Once the user chooses to authenticate against a certain provider, the Identity Manager opens the Identity Connector's URL in a popup passing it the Backplane channel id as a parameter.

This procedure is not detailed below in the use cases for the sake of brevity; it is referred to only as "the user initiates a login using IM".

4.1.2 Echo Submission Proxy

Echo servers don't accept insecure data submissions: all such requests have to be securely signed and originate from a server side. This prevents a widget from submitting data directly to Echo APIs (otherwise a secret would be exposed).

In order to circumvent such limitation Echo widgets use server counterparts which:

- receive data from the widgets via an ad-hoc protocol;
- perform authorization checks to make sure that the current user is indeed allowed to submit the data;
- place the data on the Echo server in a secure manner.

These components are called "submission proxies" since they pass through data submission requests.

Echo Submission Proxy (ESP) is the submission proxy for Echo Stream Client.

4.2 Backplane Initialization

All use cases below share the following initialization sequence, taking place at the very beginning:

1. Echo bus consumer connects to the Backplane bus (starts polling the Backplane server with GET requests to http://backplane.customer.com/v1/bus/customer.com).
2. An end user loads a Backplane-enabled page at http://customer.com/pages/1.
3. Once the page loads, Backplane library checks if there is a cookie named 'backplane-channel' set against customer.com.
4. If the cookie is set, its value is parsed and the library gets information about a channel name associated with bus name customer.com. Otherwise the library generates a new channel name and records information about the channel name and bus name association in the cookie set against customer.com domain.
5. Backplane library starts asynchronous polling of channel http://backplane.customer.com/v1/bus/customer.com/channel/<CHANNEL_NAME> with GET requests.
6. ESC, IM and LSW all call Backplane library and register their callbacks.
7. ESC makes a call to http://echoapi.customer.com/v1/users/whoami to learn the user's identity, passing the channel id as a parameter.

4.3 Use case 1: A brand-new user logs in

1. The initialization takes place.
2. Echo server behind echoapi.customer.com looks up if there is an active session object associated with the channel id. Since it's a brand new login, there isn't one so the Echo server replies to the request with "you are anonymous" response.
3. The user initiates a login using IM.
4. IM performs the login sequence communicating with idcon.com and using the channel id as a unique identifier of the session.
5. Once the login workflow is complete, idcon.com creates an 'identity/login' message and posts it to the Backplane channel (this is a secure server-to-server call).
6. Echo bus consumer receives the 'identity/login' message from the bus and:
   - uses the 'context' field to look up a customer by the URL's domain name;
   - uses 'identities' object to look up a user account in the customer's namespace;
   - creates or retrieves a session for the user account and associates it with the channel id;
   - pushes identity/ack backplane message to the channel ID (http://backplane.customer.com/v1/bus/customer.com/channel/<CHANNEL_NAME>) with information about the session
7. Backplane library receives the 'identity/login' and waits for the 'identity/ack' message

8. Backplane library receives the 'identity/ack' message, gets the information about the sender from the 'source' field. And using particular rules for the specified sender get session identifier from the 'payload' field
9. Backplane library retrieves information about session using session identifier with the help of any proprietary protocol
10. Backplane library notifies all interested widgets (ESC, IM, LSW).
11. LSW trusts the message without verification and updates its display (now shows "Logged in as ...").'
12. Backplane library passes it to ESC (as part of subscribers notification process).
13. ESC updates visual representation of the echo stream using new information about the user.

4.4 Use case 2: A logged-in user loads the page

1. The initialization takes place. Since the user is logged in, the Backplane channel id is passed along to the 'whoami' API endpoint of Echo.
2. Echo server looks up a session associated with the channel id.
3. If the session still exists, the user account information is passed back to ESC.

4.5 Use case 3: A logged-in user logs out

1. The initialization takes place. Since the user is logged in, the Backplane channel id is passed along to the 'whoami' API endpoint of Echo.
2. Echo server looks up a session associated with the channel id.
3. If the session still exists, the user account information is passed back to ESC.
4. The user initiates a logout using IM.
5. IM performs the logout sequence communicating with idcon.com and using the channel id as a unique identifier of the session.
6. Once the logout workflow is complete, idcon.com creates an 'identity/logout' message and posts it to the Backplane channel (this is a secure server-to-server call).
7. Echo bus consumer receives the 'identity/logout' message from the bus and:
   - uses the 'context' field to look up a customer by the URL's domain name;
   - uses 'identities' object to look up a user account in the customer's namespace;
   - retrieves the session for the user account and marks corresponding identity (identities) as "logged out".
   - pushes identity/ack backplane message to the channel ID (http://backplane.customer.com/v1/bus/customer.com/channel/<CHANNEL_NAME>)
8. Backplane library receives the 'identity/logout' message and waits for the 'identity/ack' message
9. Backplane library receives the 'identity/ack' message, gets the information about the sender from the 'source' field. And using particular rules for the specified sender get session identifier from the 'payload' field
10. Backplane library retrieves information about session using session identifier with the help of any proprietary protocol
11. Backplane library notifies all interested widgets (ESC, IM, LSW).
12. LSW trusts the message without verification and updates its display.

4.6 Use case 4: A third-party widget needs to learn the login status of the user after the fact

Backplane Identity Scenario is a stateless protocol. There is no way to retrieve the current state from Backplane. In order to do that a widget or client needs to keep track of Backplane events occuring on a channel. Another possibility is to use an API of a service which does that (for example, if a widget is certain that Echo is on the Backplane channel, the former can get user account information from the Echo server directly).

4.7 Use case 5: A logged-in user opens another copy of the same window

Since the Backplane channel name is stored in the cookie, opening a new window without closing the original one is no different from Use case 2 (Section 4.4).

4.8 Use case 6: A user logs in on a page where no Echo Stream Client is present

Since Echo uses Backplane channel id which is stored in a cookie, the only requirement is that the Backplane library executed on the page where login happens. This will ensure that Echo will later be able to look up its session object by the channel id (Echo bus consumer will catch the 'identity/login' message and set up the association).

4.9 Use case 7: A user leaves a 'like' or reply

1. The initialization takes place. The channel name is either retrieved from the cookie or generated anew. ESC learns the user identity (if Echo server is aware of one).
2. Echo server looks up a session associated with the channel id.
3. If the session still exists, the user account information is passed back to ESC.
4. The user clicks 'like' button or submits a comment/reply.
5. ESC sends the data (using an ad-hoc protocol) to ESP running on esp.customer.com. Channel ID value is passed along.
6. ESP makes a (server-to-server) request to Echo servers to check if the Backplane channel ID received from ESC is associated with an active session on the Echo side. If no Backplane channel was received, ESP checks if anonymous posting/liking is allowed.
7. ESP then makes a decision (based on Echo server's response and established policy) if it is okay to submit the data.
8. ESP either submits the data and returns an acknowledgment to ESC or discards the data and returns error.

5. References

5.1 Normative References

[RFC1738] Berners-Lee, T., Masinter, L., and M. McCahill, "Uniform Resource Locators (URL)", December 1994, <http://tools.ietf.org/html/rfc1738>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", March 1997, <http://tools.ietf.org/html/rfc2119>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", January 2005, <http://tools.ietf.org/html/rfc3986>.

5.2 Informative References

[Google.SGNodeMapper] Fitzpatrick, B., "SocialGraph Node Mapper", 2010, <http://code.google.com/p/google-sgnodemapper/>.

[PortableContacts] Smarr, J., "Portable Contacts 1.0 Draft C", August 2008, <http://portablecontacts.net/draft-spec.html#response-format>.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", July 2006, <http://tools.ietf.org/html/rfc4627>.

Author's Address

Echo