Documentation of PlugFest Testing

Date: 2024-11-29 17:00

Introduction

During today's PlugFest in ASSEE, we successfully tested both WebSub and MQTT protocols, confirming that they work bidirectionally. These tests were conducted with at least two different groups, ensuring interoperability and compliance with the specifications.

WebSub Testing

Initial Issues and Fixes

We encountered several issues with our WebSub implementation that required adjustments:

- **Wrong Payload Format:** Initially our Enpoint `/auctions` returned invalid Post data. It just send an empty array `[]` instead of a JSON object with an `auctions` field (i.e., `{ "auctions": [] }`).
- **Assumption about hub.topic Parameter:** We initially assumed that the WebSub hub always sends the hub.topic request parameter during the POST callback. However, it does not include this parameter when performing a POST to our callback URL. See section on Callback URL Handling for more details on how we resolved this issue.
- **Content-Type Acceptance:** Our implementation accepted only standard JSON (`application/json`). The hub, however, sends content with the media type `application/job-list+json`, especially when performing fat pings by sending the request directly. We updated our system to accept any content type.

Callback URL Handling

- **Initial Approach with URL Encoding:** We tried using URL encoding for the URI of other auction houses and appending it to our callback URL (e.g., `/auctions/notifications/<EncodedURI>`). This approach led to invalid data in the URL due to unsafe characters.
- **Solution with Base64 Encoding:** To resolve this, we switched to Base64 encoding the URI of the other auction house. This method worked perfectly, allowing us to decode the URI upon receiving a notification. We then attach `/<auction-id>/bids` to the URI to send winning bids via HTTP.

Alternative Considerations

- **Using the Link Header in Fat Pings:** After discussions with another team, we realized that if the WebSub hub uses fat pings, it includes a Link header with rel#"self". We

could extract the necessary link from this header, eliminating the need to encode the URI in the callback URL. But this would result to an Issue with Light Pings: If the hub sends light pings (empty POST requests), the Link header is absent. Therefore, relying solely on this method is not fail-proof and our method seems to be more reliable.

Compliance with WebSub Specification

- **Specification Guidelines:** According to the WebSub specification, “The callback URL SHOULD be an unguessable unique URL.”
- **Proposed Solution:** To align with the specification, we could generate a unique salted hash to include in the URI and store the hash-URI mapping in a database or in-memory store. When we receive a POST request, we use the hash to retrieve the original auction house’s URI from memory or database. We did not implement this solution due it being out of scope but it is a good practice to consider in a future project.

Suggestions for Improvement

- Including bidURI in /auctions Endpoint: The complexity of handling callback URLs could be avoided if the task force included the bidURI field in the GET /auctions endpoint payload. Alongside auctionId and jobType, this would allow us to directly obtain the bidding URI without additional logic described above.

MQTT Testing

Successful Implementation

- Out-of-the-Box Functionality: Our MQTT implementation worked successfully without any code changes. We were pleased with its immediate functionality.

Broker Stability Issues

- **Intermittent Authentication Errors:** Occasionally, the MQTT broker failed to subscribe, reporting that we were unauthenticated, even though no authentication is required.
- **Temporary Workaround:** Introducing a 2-second delay reduced the frequency of these errors but did not eliminate them entirely. Maybe consider an alternative broker or investigate the root cause of this issue.

Testing Environment

- **Remote and Local Setup:** For testing purposes, we ran the WebSub version remotely on our auction house while running the MQTT version locally. We connected to the remote broker, which facilitated efficient testing.

Deployment Challenges

Code Changes and Deployment Delays

- Pipeline Execution Time: Implementing code changes for WebSub required waiting approximately 4 minutes for the pipeline to finish and redeploy on the virtual machine.
- Startup Time: After deployment, an additional 1-2 minutes were needed for all services to start.

Manual Subscription Process

- **Calling `/subscribeToDirectory/` Endpoint:** We manually invoked the `/subscribeToDirectory/` endpoint after deployment. This process was time-consuming due to the presence of invalid or non-running auction houses in the directory, causing timeouts before proceeding to the next. Maybe in the future, consider automating this process on startup and maybe run multiple subsccriptions parallel.

Overall Experience

Positive Outcomes

- **Necessity of the PlugFest:** The PlugFest was a valuable experience, highlighting the importance of interoperability testing and adherence to specifications.
- **Identifying Overlooked Errors:** The event helped us uncover and correct errors that were missed during our internal testing.
- **Working Implementation:** We successfully demonstrated the bidirectional functionality of both WebSub and MQTT protocols.

Conclusion

Participating in the PlugFest was instrumental in refining our implementations of the WebSub and MQTT protocols. It emphasized the importance of collaborative testing and strict adherence to specifications to ensure seamless interoperability among different systems.