AllJoyn®
Open Source Project

Core Working Group Meeting

Reminder:

This call is being recorded

Antitrust Compliance Notice

- AllJoyn Open Source Project (AJOSP) meetings involve participation by industry competitors, and it is the intention of AJOSP to conduct all of its activities in accordance with applicable antitrust and competition laws. It is therefore extremely important that attendees adhere to meeting agendas, and be aware of and not participate in any activities that are prohibited under applicable US state, federal or foreign antitrust and competition laws.

- Examples of types of actions that are prohibited at AJOSP meetings and in connection with AJOSP activities are described in the Linux Foundation Antitrust Policy: https://www.linuxfoundation.org/antitrust-policy

Agenda

- Meeting schedule for rest of 2016?
  - Cancel and reconvene 1/5?
  - Next meeting will be 1/5
- 16.10 retrospective/Post-Mortem
  - Carrie volunteered to take notes

Microsoft

Good

- Way: Like the previous release, I thought our collaborations in the Alliance went very well.
- Way: We learned from our past mistake (specifically the string migration work) that we need to avoid such a scale of change later in the release. We prevented a couple of changes that later were found to be problematic.
- Way (Borderline): Reliability of Jenkins tests have continuously been improved, but still not perfect.

Opportunities

- Way: LF’s sudden removal of Jenkins validation on Windows was extremely disruptive. While I understood the business and all, it would have been more courteous to send out a warning before pulling the plug. Without Jenkins validation, my confidence in the quality significantly reduced, but luckily we had OSU helping out validating stuff on Windows.
- Way: TCL and Test should have been W4 clean. Many test failures would have been prevented had we spent the time making these repos W4-clean and enable warning-as-error. When we went to SCL to get things cleaned up, we found and fixed a bunch of real issues.
- Pawel: Whenever we add a new public C++ API, the C version should be done in parallel as part of the same task. Otherwise we end up with what’s happening right now – adding some after the release when it’s already a bit late and people don’t remember the code that well any more, so it takes more time to do a review. Having bindings for other languages would be probably even better, but while having one person write both C and C++ code is probably not an issue, it’s harder to find someone who could produce nice code additionally in Java and Obj-C.

Recommendations

- Pawel: I think it would be very beneficial to put more pressure on writing better test code. By “better” I mean shorter (5-10 lines max, ideally 1-3 lines), testing only one thing at a time and enclosing setup and teardown in test classes, not in the tests themselves. This way:
  - It’s a lot faster to understand what is exactly being tested. With a good test name you might not even have to look at the code in case of a test failure.
  - You can tell which part is the setup, so the part that’s not being tested at the moment.
  - It’s a lot easier to control freeing up allocated memory in case of a test failure (you just put everything in the teardown method).
  - In case of any code updates it’s easier to update the tests as well. Take a look at SecurityManagementTest.cc for instance – there are tests with hundreds of lines of code, where most of it is setup copy-pasted to other tests. Now in case of changes you have to check and compare this setup code to make sure it’s doing the same thing and then change it multiple times. That’s just a huge waste of time and it’s very error-prone.
- Pawel: The Jenkins builds seem to be a lot stable now, but it would still be good to have someone assigned to finding out why do we sometimes have random test failures. That may indicate a bug in the product code, or in the test itself or in the test environment’s setup. Besides eliminating these random failures will make actual regression bugs more visible.

OSU

Good

- Developer communication on issues was fast and issue resolution was usually quick
  - Conference calls with shared screens made for quicker resolutions
- TestPlan project in Jira will help with quality of the Testlink test plans
- Ultimately, we released a quality product, even if late.

Opportunities

- 16.10 released 3 weeks late
- Started late on overall feature testing due to IPv6 not being ready
- Realized late in the game that all of the Security 2.0 samples were not working or in place. Should have tested that sooner.
- Many tests are written with the assumption that the tester is familiar with the process and expected results already. This makes the learning curve rather steep. (working on fixing this with TestPlan project in Jira)
- It appears that there are tests that ultimately are duplicated. Need to do some pruning to save time in future.
- OpenWRT branch was not ready for 16.10 testing until late in the test cycle.
- iOS seemed to take awhile to resolve issues found.
- OpenWRT installation was different than previously documented. (Documentation is now updated)

Recommendation

- Review this retrospective prior to the start of the next major test cycle and apply mitigating steps sooner.

Affinegy

Good

- Delivered everything scoped for 16.10 release.
- Appreciated OSU's efficient testing and collaboration.
- Core WG and Triage meetings were very effective in drawing attention of key people to particular issues, and provide forum to revisit issues as needed.
- Effective collaboration tools.
  - Professional and quality code review via gerrit, and timely issue management via JIRA.

Opportunities I

ObjC binding/iOS

- Complicated to work with.
- Some iOS/Mac samples not working correctly.
- Obj C code is mixed with C++/C even in samples. It’s difficult to read, understand and update for the end-user.
- Many method descriptions and comments are outdated. Developer often sees wrong examples of method usage in Xcode quick help.
- Missing some necessary unit tests. Reduces confidence in some modules.
- Are all of the Public APIs implemented?

Opportunities II

Documentation

- Documentation not up to date.
- Some documentation that was updated (webdocs git) is not yet published to public website.
- The Security 2.0 API is complicated and not documented sufficiently (the only document is the HLD, which is general and does not have code examples).

Code Reviews

- Sometimes took a week before a commit would be merged to master.
- Some commits have been quite large.
  - Should we have some rule of thumb for commit size? Possibly subdivide larger commits based on some best practice.

Opportunities III

Testing

- OSU started testing toward the end of the development phase, just before end of release.
  - With 6 months of potential commits, earlier reporting of behavior and performance issues may be beneficial.
  - Maybe we could define some KPIs (key performance indicator) and test them at least once per month?
- Was test coverage sufficient? E.g. IPv6 could be tested in many different configurations.
- Missing some necessary unit test coverage in Android and iOS bindings.
- Unit tests could benefit from use of Mock framework.

Opportunities IV

Builds, Jenkins

- The SCL/TCL builds do not build some of the samples and tests.
  - Code changes to SCL or TCL might break tests or samples that aren’t being built, so broken code can go unnoticed.
- Verification builds often <u>slow</u> or <u>failed</u> due to build server issues or timing issues.
  - Extreme wait times and backlogs occur week of coding milestone dates.
  - Example win10-dbg-coretest-bvt failed often; and not because of code changes in the commit being verified.
- Some engineers don’t have permission to manually launch failed builds.

Recommendations

- More comprehensive Security 2.0 API documentation, and easy to understand code samples, for the various language bindings.
  - Current samples only cover a fraction of the API (and rely on the C++ Security Manager to do the rest).
- Work on the iOS code to make it more friendly for developers.
  - This includes samples, documentation, and possibly API changes.
- Automatic setup (AJATS?) could help, which would ran all possible sample and test apps periodically (e.g. weekly).
  - Problem may be that some (most?) of those apps may not be applicable for automatic verification.
  - Reviewers could manually verify samples as needed.
- Review latest code changes in respect to documentation impact.
- Provide an SDK (per release) for some of the language bindings (e.g. Android, iOS).
  - This would help the community (ease of use).

Marcello

Good

- Collaboration effective as always
- Seemed to be more direct engagement between test and dev
- Identified test cases that need to be updated/removed

Opportunities

- Release was almost a month late.
- Several misunderstandings on what needed to be delivered vs. what was delivered
  - Unit tests for Security 2.0
- Features were late
- Several code reviews took far too long to happen
- Testing estimates for time of completion wrong – largely due to “unknown unknowns”

Thank you