JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

BPSec Library (BSL) Critical Design

Prepared by JHU/APL for NASA AMMOS

Brian Sipos

Overview

- Build schedule review
- BSL Background
- Initial public and private repositories
- Doxygen running and viewing
- API Walkthrough
- Testing

Build Schedule Review

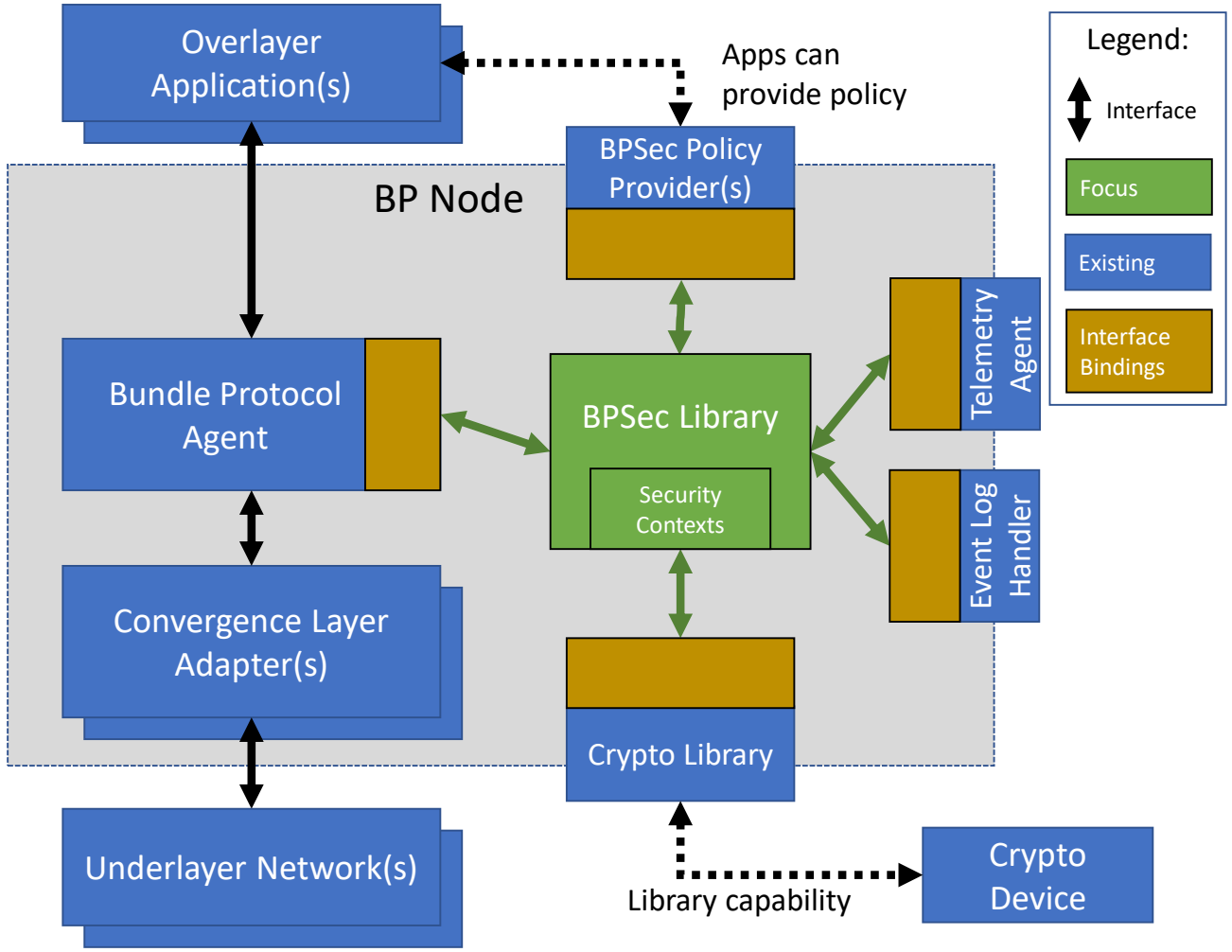
Review or Milestone	Date
Software Requirements Review	April 2024
Preliminary Design Review	May 2024
Build 0 (ION Extract)	April 2024
Critical Design Review	**September 2024**
Software Interface Specification	September 2024
Build 1 (Interfaces)	July 2024
Build 2 (Baseline set of security features necessary to support BPSec)	October 2024
Build 3 (Advanced BPSec Capabilities)	December 2024
Technical Readiness Review	January 2025
DDR	May 2025
Build 4 (Bug fixes)	April 2025
Release BPSec 1.0	July 2025
Release BPSec 1.0 Open Source	September 2025

JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

BSL Background

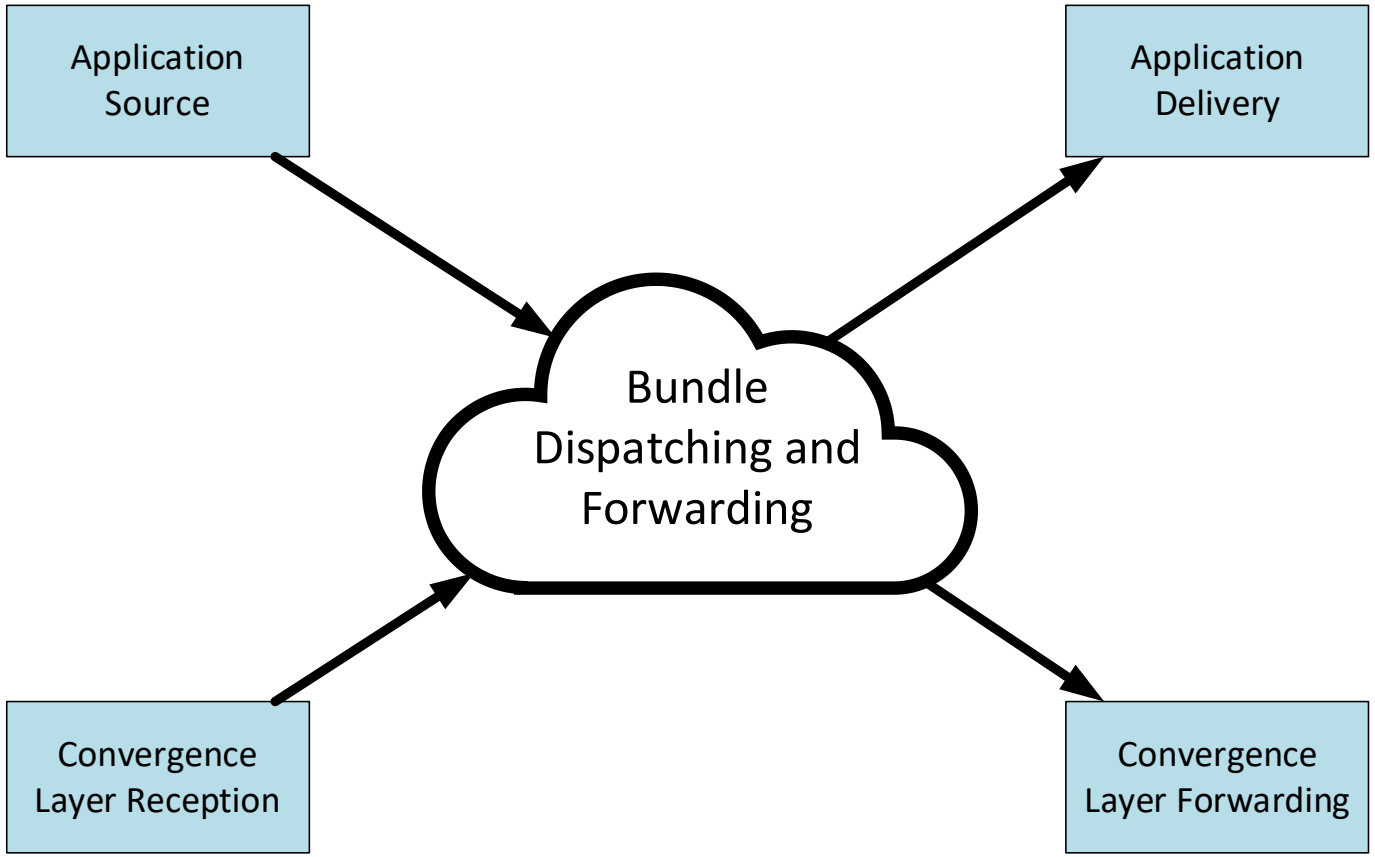
Context Diagram

- This graphic was taken from the BSL Software Requirements Document
- Indicates interfaces between BSL and systems outside of the BSL
- The BSL has two types of interface:
  - Interfaces *into* the BSL: the BPA uses the public API to drive the BSL
  - Interfaces *out of* the BSL: the BSL uses external APIs to perform functions (e.g. calls into crypto library or event log)
- The principal interface into the BSL is the Security Service interface used by the BPA



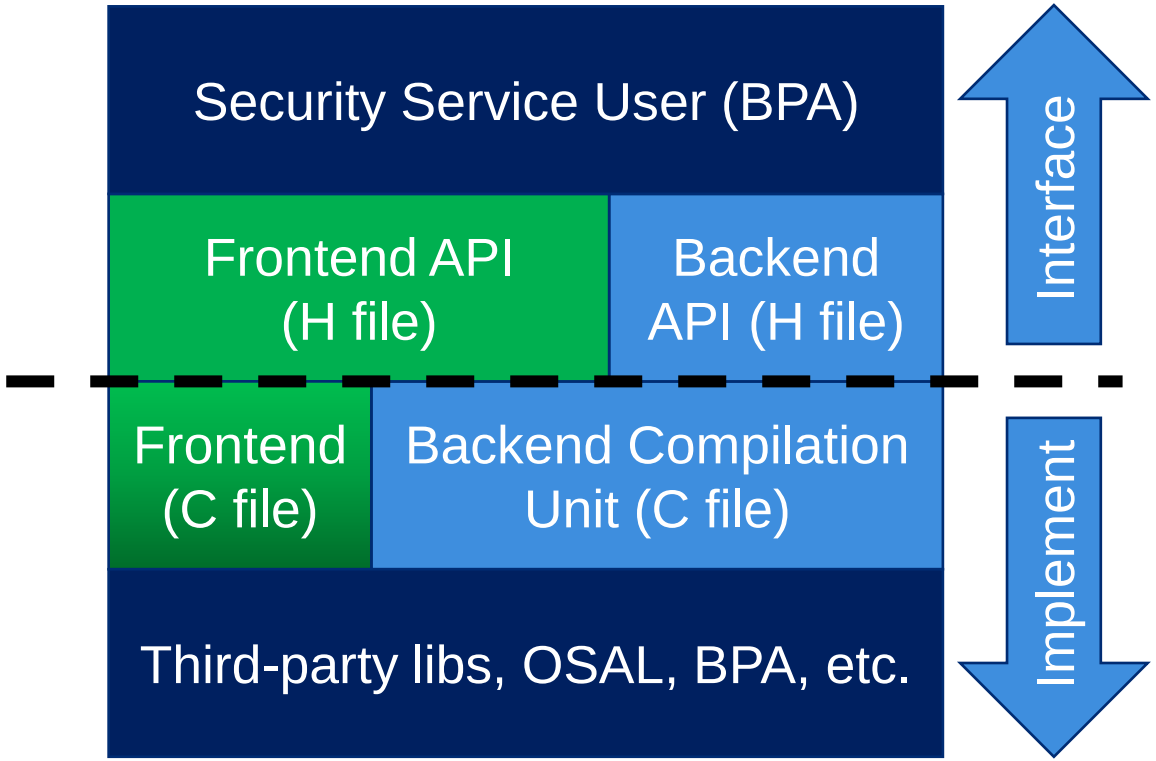
Security Interaction Points

- These four interaction points are at the boundaries of the BPA for south-side (CLA) interface and north-side (Application) interface
- Policy processing is identical for the four points except for an enumerated indication of which point is being processed



Implementation Patterns

- There is a project goal to separate a stable API from specific memory allocation strategies
  - Flight software restricts memory allocation patterns
- BSL separates internal structure into two parts:
  - Frontend API
  - Backend implementation
- Frontend is mostly header contents:
  - Forward-declared (opaque) structs
  - Public API function declarations
  - Possible pure-algorithm definitions
  - No heap allocation
- Backend is mostly compiled definitions:
  - Defined structs (i.e. member visibility)
  - Public API function definitions
  - Possible “private API” header declarations
  - Dynamic backend uses dynamic heap allocation, other backend could use static memory pool(s)



Artifact Contents

- BSL Frontend:
  - Headers declaring “abstract” interfaces used by the BSL
    - Sequential BTSD
    - Bundle Context
    - BSL Context
  - Definitions of algorithms which use Frontend APIs are independent of backend implementation
    - Overall BSL workflow
    - Security activity and operation sequencing logic
- Dynamic Backend:
  - Compilation units satisfying the BSL Frontend APIs using heap-allocated dynamic storage
  - Definition of a Bundle Context based on fully caching block information
  - Definition of Sequential BTSD based on simple, flat array iteration
  - Definition of a BSL Context with an API to register/deregister Policy Providers and Security Contexts
- Example Default Security Contexts:
  - Implementation of the BIB-HMAC-SHA2 and BCB-AES-GCM contexts within the BSL structures
- Example Policy Provider:
  - Use database and JSON representation from ION BPSec

Frontend Interface API Concepts

- Declared **BSL Context** is an opaque struct containing configuration for a single (thread) use of the BSL
  - Logically a container for any memory pools associated with BSL instance
- Declared **Bundle Context** is an opaque struct containing back-references to the BPA for obtaining and manipulating individual bundle's data
  - Logically a container for caching or sharing data between operations on the same bundle
- Declared **Sequential Reader** and **Writer** are opaque structs used to read or write BTSD for a single canonical block within a bundle
- Declared **Policy Action** struct and related functions
- Defined **Security Operation** struct and related algorithms
- Declared security functions specific to the Default Security Context needs

Dynamic Interface API Concepts

- Defined **Host Descriptor** is a copyable struct containing callbacks to host functions:
  - Memory management
  - Time keeping
  - Event logging
  - Telemetry registration
- Defined **BPA Descriptor** is a copyable struct containing callbacks to access the BPA
- Defined **Policy Provider Descriptor** is a copyable struct containing callbacks to Policy Provider functions and user data pointer
- Defined **Security Context Descriptor** is a copyable struct containing callbacks to Security Context functions and user data pointer
- Defined **BSL Context** which consists of:
  - A *Host Descriptor* and *BPA Descriptor*
  - A dynamic list of *Policy Provider Descriptor* and *Security Context Descriptor*
- Defined **Bundle Context** which consists of:
  - Primary block information and encoded form
  - A list of canonical block metadata and BTSD, along with lookup maps (cached)
- Defined **Sequential Reader** and **Writer** using flat buffers

JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

Initial Repositories

Repositories

- The BSL repository
  - https://github.com/NASA-AMMOS/BSL
  - Host for the Wiki content and eventually the Doxygen generated HTML
- The BSL-docs repository
  - https://github.com/NASA-AMMOS/BSL-docs
  - Host for design documents and eventual Product and User Guide
- The BSL-private repository
  - https://github.com/NASA-AMMOS/BSL-private
  - Temporary source repository before the API “goes public”
  - Accessible to those in the NASA-AMMOS Github organization

Project Documentation Areas

- The BSL documentation will take a few different forms for different purposes
- BSL source repository Wiki
  - AMMOS Task Description
  - Dynamic Release Plan
  - Ticket Workflow
- BSL source repository content
  - Contents of README.md, CONTRIBUTING.md, SECURITY.md, LICENSE.txt
  - Contents of actual C header files with Doxygen markup
  - Contents of Doxygen-read markdown files
  - Generated documentation in *Pages*
- BSL source repository metadata
  - Pull Requests and per-branch job status
  - Static analysis results
- BSL-docs repository
  - Copies of approved AMMOS design documentation
  - Source for User Guide and Product Guide
  - Generated User Guide and Product Guide documents in *Pages*
- BSL project
  - Views into tickets and pull requests across all repositories

JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

Documentation

Building API Documentation

- Build script automates some of the process:
  - Run `./build.sh docs`
  - View with `xdg-open build/default/docs/doxygen/html/index.html`
- CI job also builds documentation for pushed branches
  - For the `main` branch on push and weekly
  - For all Pull Requests to verify breaking doxygen typos
- This is also explained in the Software Interface Specification for BSL
- It is a manual activity to review the Doxygen output for consistency and completeness
  - This will need to be part of normal BSL review and “testing” activity

API Walkthrough

- Introduction page
- Background and Conventions pages
- Module listing
- Frontend module (file set)
- BSL library context (`bsl_ctx.h`)
  - Include dependency graph
- Bundle context (`bundle_ctx.h`)
  - Bibliography and citation conventions
- Struct `bsl_sec_opt_s`
  - Collaboration diagram
  - Cross-link to `bsl_role_t`
- Other content

JOHNS HOPKINS
APPLIED PHYSICS LABORATORY

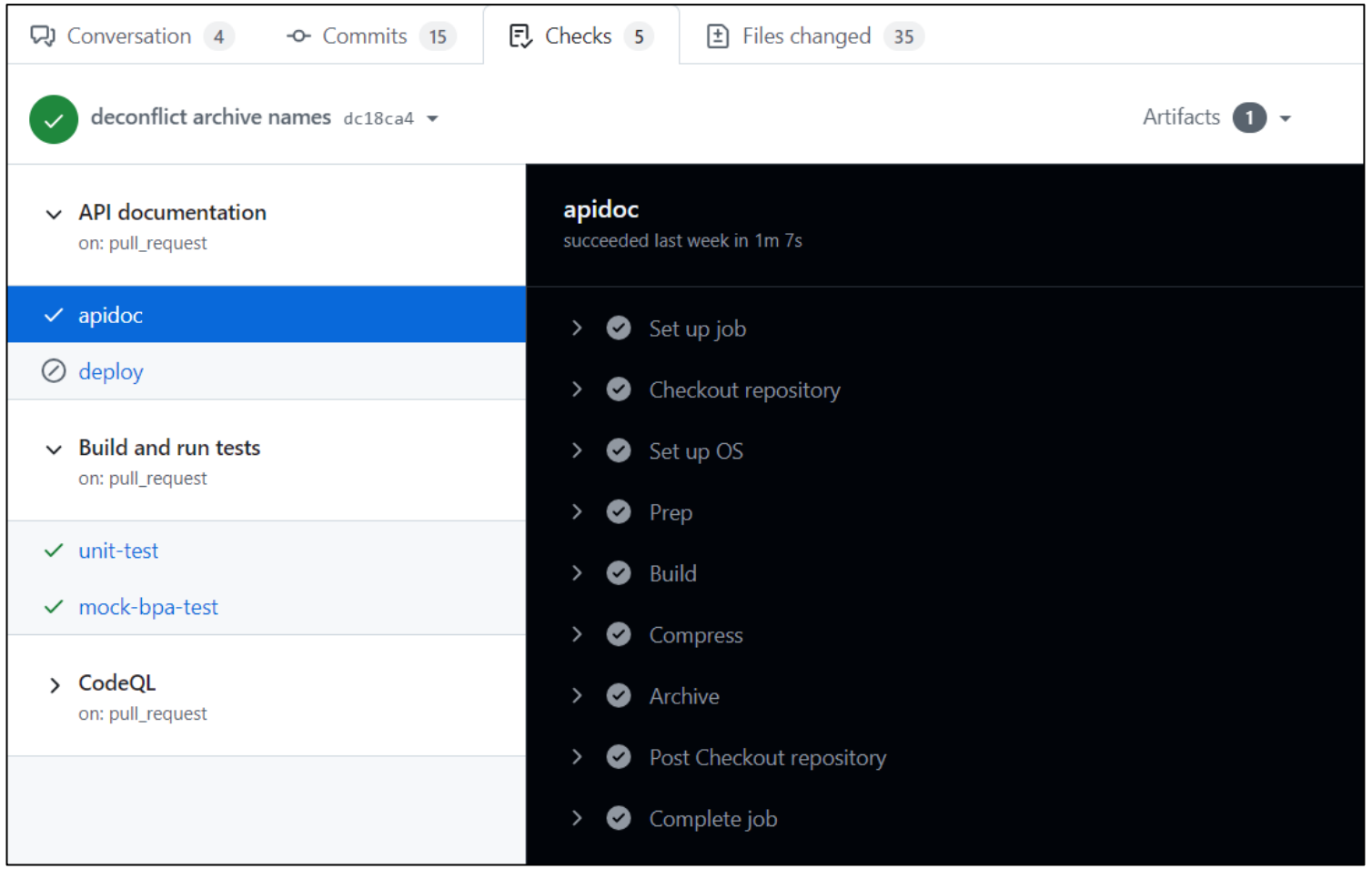
Testing

Unit and Integration Testing

- Unit Tests:
  - Library sources and unit tests are in parallel trees `src` and `test` respectively
  - Mock interfaces for testing provided for BPA and Host interfaces
  - Unit test assertions and logic provided by Unity C library
  - Test sequencing provided by CTest tool
  - Test coverage accounting and reporting provided by `gcovr` library and utilities
- Integration testing is deferred to integrations with specific BPAs and target operating systems and architectures
  - There are external projects which intend to integrate and test with BPAs/OSes and provide feedback

Current CI Snapshot

- CI applies to all Pull Requests
- Verifies build, unit tests, mock BPA tests, and Doxygen run
  - Test artifact of coverage report
  - Doxygen artifact of HTML
- On main branch deploys Doxygen output and runs CodeQL scanning
- Future CI could include RPM building if desired for AMMOS



JOHNS HOPKINS

APPLIED PHYSICS LABORATORY