Multi-Level Security API

North Carolina State Students

Spring 2020

Sponsors

- Erin Kotlyn
  - Solutions Architect
  - NC State CSC Alumni (2013)
  - erin@skywardfederal.com
- Danny Caudill
  - Full Stack Developer
  - MLS Subject Matter Expert(SME)
  - danny@skywardfederal.com
- Matthew Peters
  - Jack of all trades
  - CEO Skyward Federal
  - matt@skywardfederal.com

Skyward Federal

- Federal Contractor
- ~10 employees
- Started January 2019

Communication

- Slack!
- Access to Confluence/JIRA
- Access to Skyward Federal’s AWS environment

Overview

- Purpose
- Components
- Capabilities and Requirements
- Assumptions
- References

Purpose

The MLS API provides a suite of tools for hosting RESTful services through a single interface while isolating them at different security levels. It provides a mechanism for securing user access, storing data using SELinux labels, and running services in isolated containers according to SELinux labels.

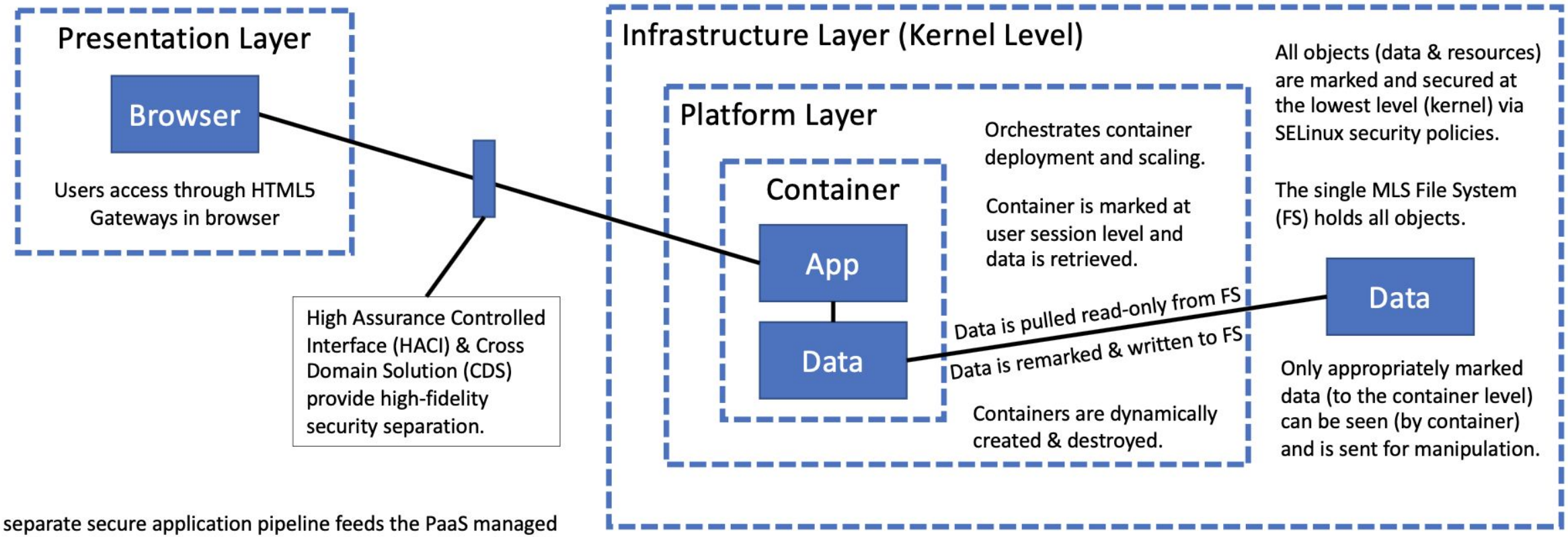
One or more MLS API instances may be present in a system, and each one can handle data across different security levels and categories. Furthermore, each component could be running within the same Operating System (server, VM, instance, etc), or they could be spread across different computers.

Components

- Consumers request services through the API Gateway. Those services may allow things like:
  - Submitting data for storage with specific SELinux sensitivities and categories
  - Retrieving data based on the user's authorization and the SELinux labels on the requested data
  - Viewing reports that a service has compiled based on an aggregation of data from multiple sources

Note also that the API Gateway + IdAM may be sufficient in some configurations (without the Container Runtime and Data Storage Components). For example, a trusted application may store its own data internally, and the API Gateway could still handle connections and user authentication.

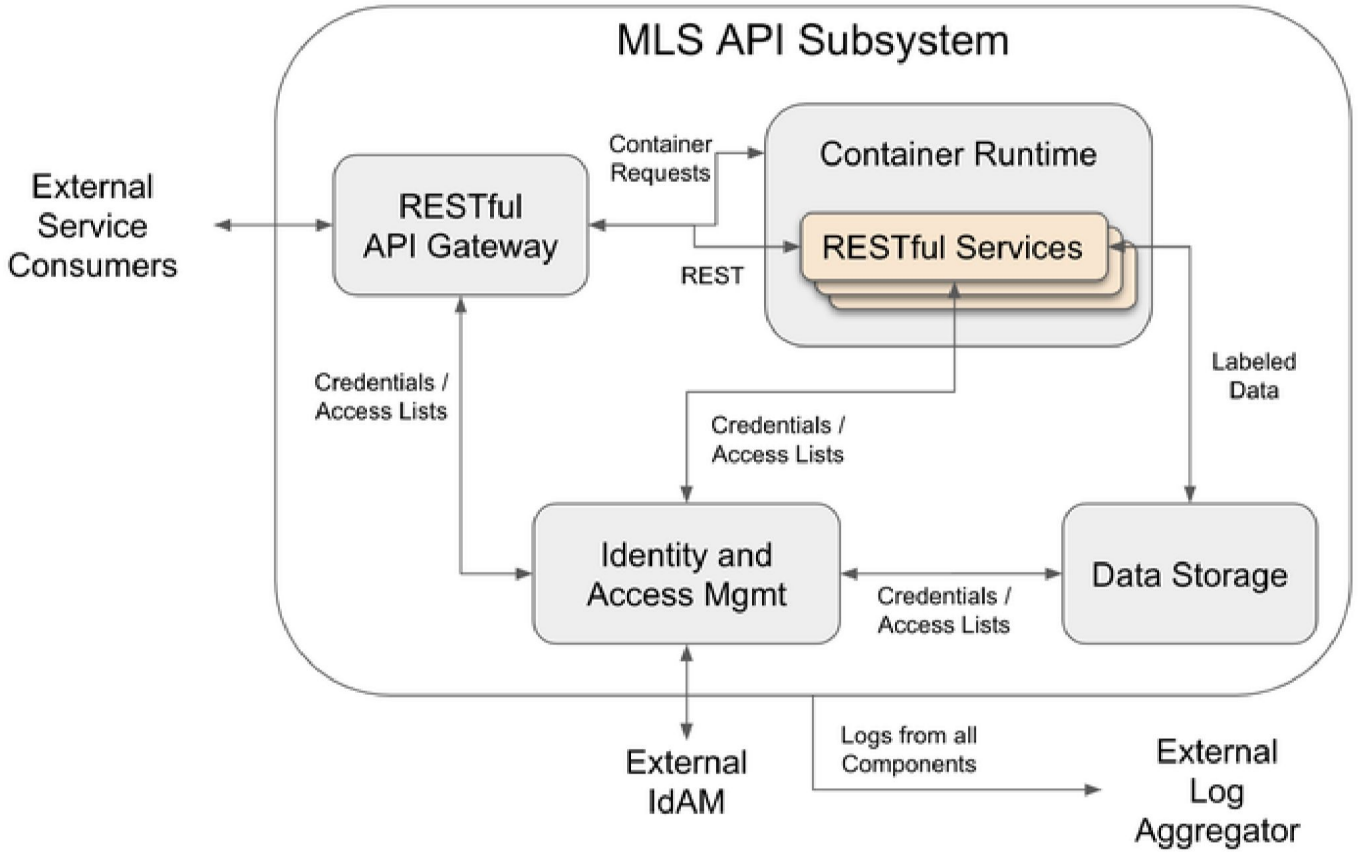
MLS/MAC Cloud Security Model Overview



A separate secure application pipeline feeds the PaaS managed 'app store'. PaaS orchestrators manage all scaling and presents a user-friendly web-based console.

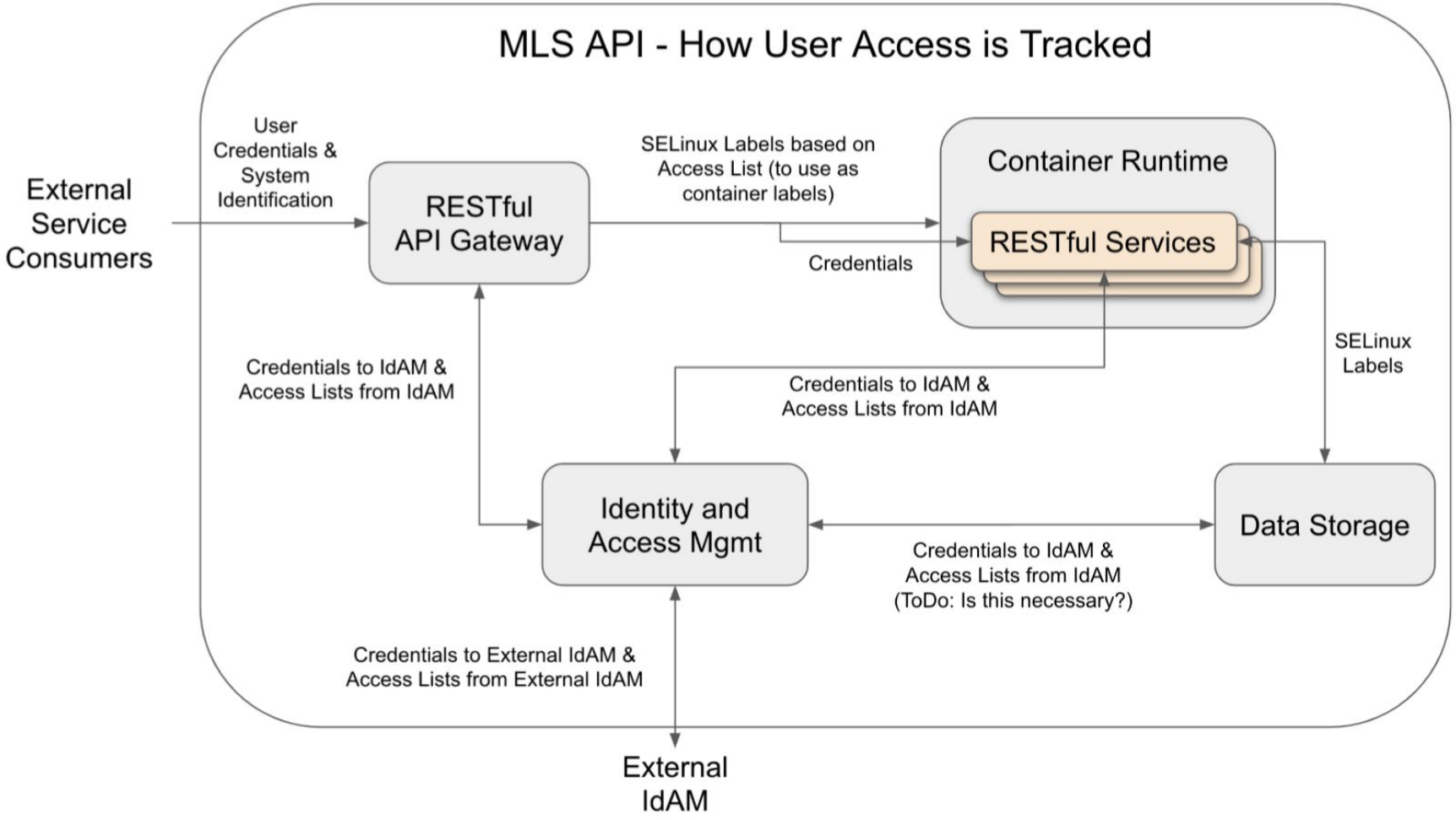
Note: All events (computer-generated & human-generated) are logged and audited in real-time.

Components (cont.)



MLS API Components and how they communicate

Components (cont.)



Passing user access information between components

Capabilities and Requirements

The MLS API Shall:

- Host RESTful services internally within containers
- Provide Multi Level Data Storage in the form of a SQL databases
- Provide isolation (encryption and access control) between security levels of:
  - Data at rest
  - Compute resources
  - Data transiting through the API Gateway Authenticate users via an external IdAM system(use Keycloak)
- Validate user authorization for access before granting access to individual APIs Provide a secure API Gateway through which RESTful services can be accessed by authenticated users work seamlessly
- Pass SELinux labels alongside all data traveling through the system Use SELinux labels for isolation whenever possible Synchronize all Operating System clocks to a common time server via the Network Time Protocol (NTP) so that log timestamps are meaningful Configure the clocks to use UTC time (or when this is not possible, document the time zone that is used for components that waive this requirement)

Capabilities and Requirements

- The MLS API Shall:
- Host RESTful services internally within containers
- Provide Multi Level Data Storage in the form of a SQL databases
- Provide isolation (encryption and access control) between security levels of:
  - Data at rest
  - Compute resources
  - Data transiting through the API Gateway Authenticate users via an external IdAM system (such as JADE)
- Validate user authorization for access before granting access to individual APIs
- Provide a secure API Gateway through which RESTful services can be accessed by authenticated users
- Pass SELinux labels alongside all data traveling through the system
- Use SELinux labels for isolation whenever possible
- Synchronize all Operating System clocks to a common time server via the Network Time Protocol (NTP) so that log timestamps are meaningful
  - Configure the clocks to use UTC time (or when this is not possible, document the time zone that is used for components that waive this requirement)

Assumptions

- In the absence of clear requirements, prefer to design for flexibility. These assumptions constrain the scope of the design. If any become invalid, then the design will need to change, so please let us know!
- All hosted services provide a RESTful API to external consumers.
  - All data that passes through must be wrapped in one of the supported REST media types (usually JSON, XML, or HTML; but vendor-specific media types may be supported as well)
- The generic term ‘data’ is used throughout this document to refer to any series of bytes that can be transferred using REST. It is up to individual services to interpret that data properly.
- SELinux labels are the core of the Multi Level Security implementation.
  - Every application that touches data must be aware of them (or at least pass them unchanged). Include support for systems that use the Multi Category Security (MCS) instead of MLS.

Assumptions(cont.)

- All services store data in a SQL database. In particular, a PostgreSQL database that uses SELinux labels.
- Humans provide the SELinux labels for all data. In future phases, a machine learning algorithm will make recommendations (requiring human approval). Eventually, that may evolve to radically reduce the human workload.
- The internal IdAM knows the level of the system where a user is requesting services (through something like a configuration file or information passed as part of the login credentials). All logs (included access logs, error logs, and status logs) are forwarded to an external log aggregator. If log forwarding fails for some reason, an Alert must appear where an operator will see it
- All logs (included access logs, error logs, and status logs) are forwarded to an external log aggregator. If log forwarding fails for some reason, an Alert must appear where an operator will see it

Development Tools

- Access to Skyward Federal’s AWS environment
- Access to Confluence/JIRA

References

- Overview of MLS Requirements and Guidelines https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.2.0/com.ibm.zos.v2r2.e0ze100/ch1.htm Overview of RESTful APIs https://en.wikipedia.org/wiki/Representational_state_transfer
- Documentation about SELinux labels related to Multi Level Security https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/selinux_users_and_administrators_guide/mls
- These are the best overviews I've found regarding the range field within an SELinux security context https://selinuxproject.org/page/MLSStatements and https://wiki.postgresql.org/wiki/SEPostgreSQL_SELinux_Overview#MLS_and_MCS
- Here is a good summary of the difference between Authentication and Authorization https://www.okta.com/identity-101/authentication-vs-authorization/
- Oracle's recommendations for a successful MLS implementation https://www.oracle.com/database/technologies/security/label-security-multi-level.html
- This SE-PostgreSQL reference is from the initial slides that led to the creation of this MLS API document https://selinuxproject.org/page/NB_SQL_9.3
- Amazon article about configuring a single API Gateway to serve different types of customers https://aws.amazon.com/blogs/architecture/using-api-gateway-as-a-single-entry-point-for-web-applications-and-api-microservice s/