### COps: **Common Operations Platform**

### Interim Progress Report 1

**Final Requirements, Design, Implementation/Testing & Installation/Delivery**

**Skyward Federal**

**CSC 492 - 32:**

Jonathan Balliet

Caleb Boswell

Daniel Mills

Jeen Shaji

Spencer Yoder

**North Carolina State University**

**Department of Computer Science**

**<February 3, 2020>**

#### Executive Summary

|  |  |
| --- | --- |
| Primary Author | Daniel Mills |
| Secondary Author |  |
| Editor | Spencer Yoder |

Many companies and organizations need to store data that should only be accessible to certain users, but not others. Some examples include banks, government institutions, and healthcare organizations. The way these organizations typically handle this is by keeping data with different levels of access in separate data stores. Data with one security level is kept in a different physical server than data with a different security level. While this is a good solution in terms of security, it is inefficient and leads to a system with multiple potential points of failure.

Our solution to the problem, posed by Skyward Federal, is to keep everything in a centralized data store, and give the data labels that will determine which users are allowed to access them. By using a Postgres database in tandem with SELinux labels, we will be able to implement a simple, integratable system that organizations can use to improve their data storage. An important component of the solution is the use of Docker containers to run the services which interact with the database. This virtualization improves security by preventing the user from directly interacting with the database, and also means that the system doesn’t rely on the availability of one single computer.

This report outlines in more detail our plans for the project, as well as all of the progress we have made so far. The Project Description section gives a more detailed look at our project’s sponsor, their initial problem statement, the goals and benefits of the project, our development plan, and some of the challenges we have faced so far. The Requirements section includes the functional and non-functional requirements for the project, as well as some constraints. The functional requirements are given in the form of use cases. The Resources Needed section details all of the technology and external dependencies we will need for the project. The Design section has diagrams for both the high-level system design and the low-level component design. It also describes the designs for the different modules that make up the project. The Implementation section describes our plan to implement the project, including our iterations and security considerations. The Test Plan section details how we will test our system, and the Task Plan section has a working list of development activities, including schedules and deadlines for all tasks.

#### Project Description

##### Sponsor Background

|  |  |
| --- | --- |
| Primary Author | Jeen Shaji |
| Secondary Author | Spencer Yoder |
| Editor | Caleb Boswell |

Skyward Federal is a software start-up specializing in custom software solutions for government clients. They provide forward-leaning technical solutions that incorporate security and modern software development practices to solve some of the Department of Defense’s biggest problems. The company was founded in late 2019 and is based in Colorado Springs, Boston and Washington D.C. Our particular project relates to their aim to provide multi-level security solutions to their clients as well as their aim to modernize software systems for their clients. Their particular motivation for this project was to get a proof-of-concept for a platform specifically meant for government clients with outdated data storage systems.

Our contacts at Skyward have repeatedly communicated to us that what we are able to create here will greatly inform how they approach this project when implementing it for their customers. On their end, this senior design project is a part of the research phase for COPS.

Erin Kotlyn, Solutions Architect

Contact: [erin@skywardfederal.com](mailto:erin@skywardfederal.com)

Danny Caudill, Full Stack Developer

Contact: [danny@skywardfederal.com](mailto:danny@skywardfederal.com)

Matthew Peters, CEO Skyward Federal

Contact: [matt@skywardfederal.com](mailto:matt@skywardfederal.com)

##### Problem Statement

|  |  |
| --- | --- |
| Primary Author | Jeen Shaji |
| Secondary Author | Spencer Yoder |
| Editor |  |

#### Many government and private firms have the need to separate their data by security level. These organizations traditionally store data of different security levels in disparate databases instead of a central location to preserve security. This solution has the problems of being difficult to scale, requiring a complex architecture to navigate, having multiple points of failure, and incurring a lot of time overhead when searching through data from multiple sources.

##### Project Goals & Benefits

|  |  |
| --- | --- |
| Primary Author | Jeen Shaji |
| Secondary Author | Spencer Yoder |
| Editor |  |

#### Goals:

* Centralize access-based data separation in a single system while maintaining security
* Virtualize this system so that the services aren’t tied to the availability of a single computer
* Create a simple REST API which provides access to these services
* Create, configure, and populate a secure database which prevents certain data accesses at the operating system level.
* Write controllers to handle API calls and communicate with the database.

#### Benefits:

* Reduces overhead when acquiring data from multiple sources
* Reduces system complexity
* Virtualization allows for unique system access behind simple API calls based on user access lists and removes failure related to a single computer system.
* Data protection at the operating system level preserves the security achieved by the former system of storing data in multiple silos.

##### Development Methodology

|  |  |
| --- | --- |
| Primary Author | Spencer Yoder |
| Secondary Author |  |
| Editor | Jonathan Balliet |

We intend to use an iterative methodology for our project. Much of the success of our work is based on the feedback from our sponsor. Furthermore, we’re venturing into largely unknown territory. Since we don’t know exactly what will work and what won’t until we start working and trying things out, we can’t write accurate requirements/design documents without going back and modifying them. Each iteration, we intend to have a full system which is able to take API calls all the way to the database and back. Since we’re working with the Skyward team, who uses an agile methodology, our iterations will likely take 2 weeks on average. The caveat with this estimation is that we often have to wait for resources to be configured and allocated for us by our sponsor: our timeline is not entirely up to us.

* Iteration 1 (2/6/2020 - 2/20/2020)
  + Data Storage Team (Spencer Yoder, Daniel Mills)
    - Generate and configure a full SE-Postgres database
  + Container Runtime Team (Jonathan Balliet, Jeen Shaji, Caleb Boswell)
    - Create infrastructure, program, and test a single API call.
* Subsequent iterations TBD

Once we start coding, we plan to use a Kanban project board inside of GitHub to keep track of our progress as well as what each member is currently working on. This not only helps us keep track of the pace of development, but also facilitates coordination between group members: if I can see what everyone is doing, what’s been done and what has not been done, I can make a decision about what to do.

##### Challenges

|  |  |
| --- | --- |
| Primary Author | Spencer Yoder |
| Secondary Author |  |
| Editor | Jonathan Balliet |

##### So far, our team's challenges have been related to a lack of knowledge. Our sponsor gave us a lot of research, a list of esoteric technologies, and designed a whole system for us to use. We’ve been spending most of our time figuring out what these pieces are and how they fit together. As with all learning experiences, we had a lot of misconceptions about this and every meeting has involved a correction of these misconceptions by our sponsor. While learning is a necessary element of our project, especially during this initial stage, it’s been difficult to find our footing and make proper plans.

We’ve also had a little time to try out some of the technologies we’ll be using and came across issues configuring and installing them. We solved these problems by talking to our sponsor, emailing the IT department, and doing a lot of online research.

##### So far, we’ve dealt with these challenges by being proactive in our pursuit of knowledge and adaptive to changing requirements and changing understanding of the project. We’ve done lots of research, asked lots of questions, and had a lot of discussion amongst ourselves to come to a joint understanding of the nature of our project. I would say that facing these challenges has given us a very good understanding of our system and the interactions between the technologies we plan on using.

#### Requirements

|  |  |
| --- | --- |
| Primary Author | Jeen Shaji |
| Secondary Author |  |
| Editor | Daniel Mills |

##### Overall View

This project involves the development of an application through which companies can isolate data and share relevant information with the employees. Although data is largely available, it is essential to keep it safe. On authentication, employees can only access the data they have permission for depending on their department, role in the company and the device they are using.

The final product is a web page where company employees can access important information and all access is tracked. Security and privacy of such a system are of paramount importance. Company rules protect information and also allow an admin to dictate who can access particular information.

##### Glossary

There are currently two roles in the COPS Platform system. The role grants access and their security clearance determines their viewing and editing capabilities.

* Administrator: The administrator assigns username and passwords to employees. They can assign or edit security context for each user and data.
* Employee: We consider everyone using the system to be a normal user. Access to data is purely based on security context assigned by the administrator.

##### Functional Requirements

UC1: User Functionality

UC2: Authenticate Users

UC3: Log Transactions

UC4: Password Functionality

UC5: Data Storage

UC6: View access logs

###### 

###### UC1: User Functionality

1.1 Preconditions

The COPS Admin has authenticated themselves in the system.

1.2 Main Flow

The admin selects the option to create a user [S1], delete a user [S2] or edit a user [S3]. A success message is displayed and the action is logged.

1.3 Sub-flows

* [S1] An Admin enters a user name, a password, confirm password, the security level, and if the employee is active in the system. The Admin sends a request to add the employee[E1][E2][E3]. The user name serves as the Employee's login name. The possible security levels are:
  + Unclassified - s0
  + Confidential - s1
  + Secret - s2
  + Top secret - s3

The employee can be assigned a list of different security levels (e.g. s0-s2), or a list of individual security levels (e.g. s0,s1,s3). Within each security level, the employee can be assigned a range of categories (e.g. c0.c3), or a list of individual categories (e.g. c0,c2,c4).

* [S2] An Admin selects an employee from the list of possible employees, confirms the delete, and sends the request to delete the employee.
* [S3] An Admin selects an employee from the list of possible employees, inputs a SELinux security context, confirms the security context, and sends the request to change the employee.

1.4 Alternate Flows

* [E1] The system prompts the Admin to correct the format of a required data field because the input of that data field does not match that specified in Section 1.6.
* [E2] The system prompts the Admin to change the username if it already exists in the system.
* [E3] The password and repeated password must match or an error is displayed.

1.5 Logging

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Transaction code | Description | Logged in username | Secondary user | Transaction type |
| 100 | New employee created | Admin | Employee | Create |
| 101 | View employee | Admin | Employee | View |
| 102 | View employees | Admin | N/A | View |
| 103 | Delete employee | Admin | Employee | Delete |
| 104 | Update employee | Admin | Employee | Edit |

1.6 Data Format

|  |  |
| --- | --- |
| Field | Format |
| User name | Between 6 and 20 alpha characters and symbols - or \_ |
| Password | Between 6 and 20 characters |
| Repeated Password | Between 6 and 20 characters |

###### UC2: Authenticate Users (SG)

2.1 Preconditions

An Employee has been created in the system UC1.

2.2 Main Flow

An employee enters their username and password to gain role/ security level based entry into the COPS system [E1]. A session that has been inactive for more than ten minutes is terminated [S1]. Upon successful authentication, the employee will be able to send requests to interact with data. An authenticate session ends with the employee sending a request to log out or close the system.

2.3 Sub-flows

* [S1] Electronic sessions must terminate after ten minutes of inactivity. Ensure that authentication is reset after a period of inactivity that exceeds ten minutes.

2.4 Alternate Flows

* [E1] The user may try three times. After three failed attempts with a user id, lock out that username for 60 minutes. If the last 6 login attempts from a given IP address fail, across any number of users or non-existent usernames, that IP address will be locked out for 60 minutes. After the 60 minute lockout-period, a user gets 3 more attempts, and an IP address gets 6 more attempts.
* [E2] If a user or IP address is locked out 3 times in a 24-hour period, the offending user or IP address will be banned from the system until re-authorized by a system administrator.

2.5 Logging

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Transaction code | Description | Logged in username | Secondary user | Transaction type |
| 0 | Failed login | IP address | N/A | Other |
| 1 | Successful login | user | N/A | Other |
| 2 | Logged out | user | N/A | Other |
| 3 | User locked out | user | N/A | Other |
| 4 | IP locked out | IP address | N/A | Other |
| 5 | User banned | user | N/A | Other |
| 6 | IP banned | IP address | N/A | Other |

2.6 Data Format

|  |  |
| --- | --- |
| Field | Format |
| time | YYYY-MM-DD HH:MM:SS |
| ip | A valid IPv4 or IPv6 address. A String in the form "X.X.X.X" where each X is a integer (base 10) from 0 to 255, or "X:X:X:X:X:X:X:X" where each X is a hexadecimal number from 0000 to FFFF (no 0x- prefix), and extraneous zeros may be omitted. |
| user | A valid user within the system |

###### UC3: Log Transactions

Maintaining confidentiality and integrity of data is paramount. Complete log files are critical for performing forensics on inappropriate access (create, read, update, delete) of data and on the inappropriate granting of system privileges to users.

3.1 Preconditions

None

3.2 Main Flow

Any event which creates, views, edits, or deletes information is logged [S1]. Login failures, valid authentication, and log outs are also logged [S2].

3.3 Sub-flows

* [S1] For creating, viewing, modifying, or deleting information, the following information is recorded: the username of the logged in user, any appropriate secondary username of the user whose information is being accessed, a transaction type corresponding to the given action, and the current timestamp. Individual audit codes related to specific use cases are presented within each Use Case description. The subflow and transaction values are based on Use Case. For example, any in the range of 100-199 are for UC1, any in the range of 300-399 are in UC3, etc. The exception is for authentication [S2].
* [S2] The values from range 0-99 are logging events which do not exist in any use case but are concerned with the system as a whole. Logging associated with authentication [UC2] is also in this range. Miscellaneous transaction codes 1-99 are presented in Section 3.5 below.

3.4 Alternate Flows

None

3.5 Logging

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Transaction code | Description | Logged in username | Secondary user | Transaction type |
| 0 | Failed login | IP address | N/A | Other |
| 10 | Successful login | user | N/A | Other |
| 5 | User locked | user | N/A | Other |

###### UC4: Password Functionality (SG)

4.1 Preconditions

A User has been created in the system UC1.

4.2 Main Flow

A user may change their password [S1].

4.3 Sub-flows

* [S1] An existing user in the system may change their current password. The user sends a request with their current password, a new password, and a confirmation of their new password. If the user attempts to change their password to their current password, an error is thrown [E1]. If the input fields for "new password" and "confirm new password" are different, an error is thrown [E2].

4.4 Alternate Flows

* [E1] If a user attempts to change their password to a password that is identical to their current password, an error is thrown stating "new password must be different", and the user is prompted to "enter a new password".
* [E2] If a user is attempting to change their password and enters two different values for "new password" and "confirm new password", then an error is thrown stating "confirm new password by repeating it in the input field".

4.5 Logging

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Transaction code | Description | Logged in username | Secondary user | Transaction type |
| 400 | Failed password update | user | N/A | Other |
| 401 | Successful password update | user | N/A | Other |

4.6 Data Format

|  |  |
| --- | --- |
| Field | Format |
| New Password | Between 6 and 20 characters |
| Confirm Password | Between 6 and 20 characters |

###### UC5: Data Interaction

5.1 Preconditions

The user has authenticated themselves in COPS Platform system UC2.

5.2 Main Flow

An employee can submit data [S1], retrieve data [S2] and view reports [S3]. An admin can store and label data in the database [S4].

5.3 Sub-flows

* [S1] Employee sends a request to the system with the following information [E1]:
  + Data
* [S2] Employee sends a request to view specific data and the system returns this information [E2].
* [S3] Employee sends a request to view reports and gets an aggregated report on the [E2].
* [S4] Only Admin can access the database directly. They can add data and change the security label.

5.4 Alternate Flows

* [E1] If the Employee is trying to submit information to a security level not allowed to them, the system returns an error message saying “Access denied as you can only submit data in your own security context”.
* [E2] If the Employee is trying to access information from a higher security level, the system returns an error message saying “Access denied”.

5.5 Logging

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Transaction code | Description | Logged in username | Secondary user | Transaction type |
| 500 | Successful data submission | User | N/A | Submit |
| 501 | Retrieve data | User | N/A | View |
| 502 | View report | User | N/A | View |
| 503 | Failed access | User | Security level attempted to access | View |
| 504 | Failed data submission | User | Security level attempted to access | Submit |

5.6 Data Format

|  |  |
| --- | --- |
| Field | Format |
| Data | Follows format for the organization/ file they edit |

###### UC6: View access logs (SG)

6.1 Preconditions

A user is a registered user of the COPS system (UC1). The user has authenticated himself or herself into the system (UC2).

6.2 Main Flow

The user logs in and sees with the top entries in his or her access log [S1]. The user can send a request to view his or her access log [S2] separately. The user can then choose the beginning and end date for the period of time they would like to view their access log for [S3] and send the request. The resulting list should include the following for each access:

* Name of accessor
* Role of accessor
* Date and time of access
* Transaction type

6.3 Sub-flows

* [S1] When the user logs in, they can see ten most recent events.
* [S2] By default, upon requesting a detailed list, the user is presented with a list of all entries sorted by dates, most recent access first.
* [S3] The user may enter a date range to view all entries within the range [E1].

6.4 Alternate Flows

* [E1] The user sends an invalid date, or sends an end date that is before the start date. No events are returned and the user has the opportunity to send different dates.

6.5 Logging

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Transaction code | Description | Logged in username | Secondary user | Transaction type |
| 700 | Access logs viewed | User | N/A | View |

##### Non-functional Requirements

NFR1: The system shall prioritize security over performance when that comes into conflict.

NFR2: The installation procedure must work in an offline/disconnected environment.

NFR3: The final configuration must run on CentOS 7 with SELinux enabled.

NFR4: It shall ensure container startup time is as fast as possible. The Docker images shall include all executables, libraries, and configuration data so that application startup is the only action required when the container starts.

NFR5:

##### Constraints

C1: AWS shall be used for the API gateway and serving endpoints.

C2: The PostgreSQL database should use scram-sha-256 for password authentication.

C3: This component shall allow connections from the API Gateway to the Docker Daemon on TCP Port 2376 with configured encrypted communication.

C4: Each RESTful service shall be given login credentials to connect to the SE-PostgreSQL database within the Data Storage component.

C5: The RESTful services shall handle incoming requests on TCP Port 443.

#### Resources Needed

|  |  |
| --- | --- |
| Primary Author | Jonathan Balliet |
| Secondary Author |  |
| Editor |  |

#### 

|  |  |  |
| --- | --- | --- |
| Resource | Explanation | Status |
| AWS | AWS is used for hosting all our software components remotely in the cloud. Skyward has provided the resources for AWS and given all of us access with personal accounts. Our two main components (Data Storage and Container Runtime) will both be on a single EC2 instance using a CentOs 7 image. A S3 bucket will store the identification key that will be used for us gaining access (via ssh) to any remote running instances on AWS. | ✅ |
| GitLab | Skyward’s Gitlab will be used as a remote repository in order for Skyward to directly access our code and project. It also functions as a DevOps tool to enable CI/CD with Jenkins. | ✅ |
| GitHub | NCSU provides us with a GitHub repository for our project where all our development code must be remotely committed and stored. Our team plans to make use of Kanban boards on GitHub to keep track of issues and the progress of our iterations. | ✅ |
| Bastion | The Bastion sandbox environment provided by Skyward will serve as our main source of development. This environment hosts the EC2 instance on AWS where we are developing and storing the Data Storage and Container Runtime components. | ✅ |
| Confluence | Confluence is a wiki tool used to help teams collaborate and share documentation. We need access to Skyward's confluence in order to fully see all of the information and research that is referenced in the original documentation that was provided by Skyward. | ❌ |
| JIRA | JIRA is used for tracking issues and software development project progress. We need to access Skyward's JIRA in order to integrate into Skyward’s two week sprints so they can track our progress for this project. | ❌ |
| KeyCloak | KeyCloak is used for authenticating an individual and giving them a specific level of authorization. We needed KeyCloak in order to authenticate ourselves when setting up our personal accounts with Skyward’s GitLab, Jenkins, and AWS to ensure security. | ✅ |
| Jenkins | Jenkins is used for CI/CD. It ensures we pass our required test coverage and that there are no significant errors before our software is built and deployed. We plan to use Skyward’s Jenkins for automating the testing, building, and deploying of our project. | ✅ |
| Slack | Slack is an instant messaging platform that we will use for communicating directly with Skyward Federal. They set up a channel for our student team specifically. On this channel we can ask questions, communicate about our project and meetings, and update them of our progress and action items. | ✅ |

#### 

#### Design

|  |  |
| --- | --- |
| Primary Author | Jonathan Balliet |
| Secondary Author | Spencer “Spencer Yoder” Yoder (Database Design) |
| Editor | Spencer “Spencer” Yoder |

#### High Level Design

Figure 1: High Level System Design of the COPS Platform

Figure 1 shows the COPS Platform system at a high level. This system is made up of the two main components: the *Container Runtime* and the *Data Storage*. The Data Storage component is a PostgreSQL table that makes use of Security Enhanced Linux (SELinux) labels for all stored data. This is to ensure that a user can only access the data they are authorized to do so. An admin is responsible for storing and labeling all of the data in the Data Storage component.

The Container Runtime creates a Docker Container after receiving a service request from a consumer. The Container Runtime will make use of Python’s Flask framework for handling the service requests and for container creation and shutdown. The running service, inside of each container, will also make use of the Flask framework to handle REST requests and responses. For communicating from a running to service to the Data Storage, the Python SQLAcademy library will be used. A third component, that is a stretch goal, is the Log Aggregator. This will make use of the Rsyslog software utility for aggregating logs from all components for this system. The Container Runtime and Data Storage components will both be stored in a single Elastic Compute Cloud (EC2) instance using Amazon Web Services (AWS). This EC2 instance will use a CentOs 7 image. Using this specific operating system image was given to us as a constraint for this system due to its capability of easily integrating with the SELinux module.

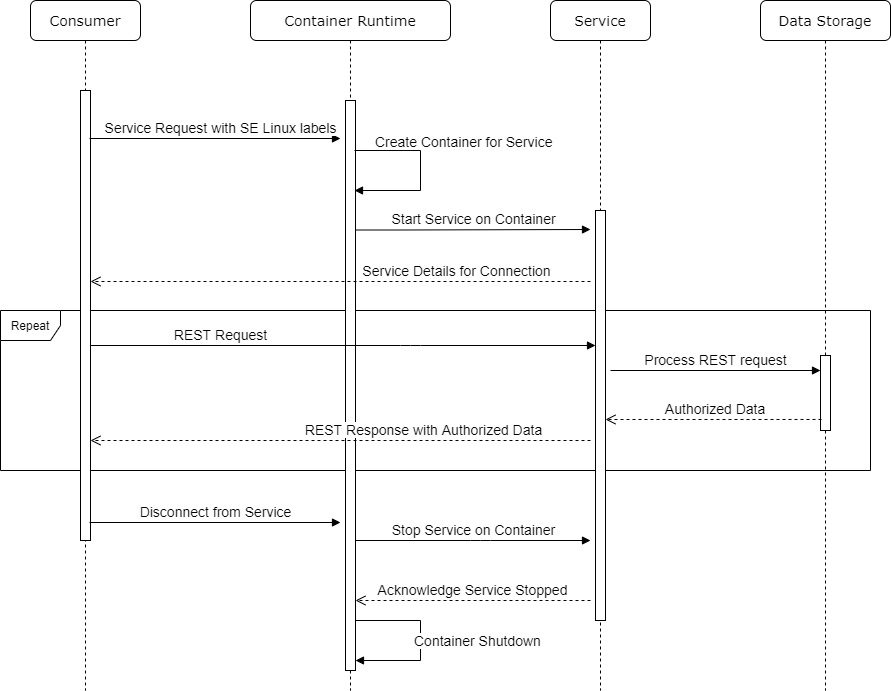


Figure 2: Sequence diagram of a service request to COPs Platform

This sequence diagram demonstrates the events that occur when a service request is sent from a consumer to the COPS Platform. This service request is sent with SELinux security labels to the Container Runtime. These security labels determine what data the service is authorized and not authorized to access. After receiving this request, the Container Runtime starts up a container instance to run this RESTful service. The container enforces the privileges of the running service to exactly match the SELinux labels sent with the request. This running service establishes a connection with the consumer, and then handles any REST requests that are sent from the consumer. The service communicates with the Data Storage component to process the REST request and retrieve any requested data it is authorized to. This authorized data is then sent back to the consumer as a REST response. REST requests and responses between the consumer and the running service continue in this manner until the consumer disconnects from the service. The service is then stopped on the container. After the container receives an acknowledgement that the service has been stopped, the Container Runtime terminates this running container instance.

**Low Level Design**

**Overview of Modules**

**[ TODO: Create a UML diagram showing the relationship of all our modules with an explanation** ]

**Modules**

**Container Runtime Module**

*Purpose*

This module is responsible for handling service requests that are sent by consumers. The Container Runtime isolates running services by creating a specific container instance for the service based on the consumer’s security levels that it receives in the request. This module also handles shutting down its container instances once the consumer has disconnected from the service.

*Responsibilities*

* Handles the REST API endpoint for service requests from consumers.
  + GET request
  + endpoint: {base\_path}/api/container/runtime/{service}/{SELinux labels}
  + The specific *service* and the consumer’s *SELinux labels* are sent with the GET request as query string parameters
* Creates a container instance for the requested service
  + Each service will have its own Docker Image with the needed dependencies, configurations, and files for running this service
  + Enforces SELinux labels on any processes/services running on the container
  + Starts the RESTful service on the container instance
* Shut downs running container instances
  + The container instance stops a running service when the consumer has disconnected from the service
  + Destroys the container instance when notified by the container of the service being stopped
* Logs all transactions

**Postgres Service Module**

*Purpose*

This module is responsible for handling REST requests from the consumer, communicating these requests to the Data Storage, and returning the response to the consumer.

*Responsibilities*

* Establishes connection with the consumer to the PostgreSQL database
  + Connect to the database using credentials that are based on the consumer’s SELinux security context
  + Send response to consumer letting them know whether connection has been successfully established
* Handles read requests to the database from the consumer
  + GET request
  + endpoint: {base\_path}/api/read/{table}
  + Returns REST response of the data (if successful) or error message if not
* Handles write requests to the database from the consumer
  + PUT request
  + endpoint: {base\_path}/api/write/{table}/{data}
  + Returns REST response of success or error message if not successful
* Handles view aggregate report requests to the database from the consumer
  + GET request
  + endpoint: {base\_path}/api/view/reports/{table}/{filter}
  + Returns REST response of the aggregated report (if successful) or error message if not
* Shuts down service when notified consumer disconnects
  + Disconnects from the database
* Logs all transactions

**Logging Util Module**

*Purpose*

This utility module is responsible for logging all transactions that occur in other modules in this system and sending the logs to the Log Aggregrator.

*Responsibilities*

* Each module contains its own Logging Util module with the static codes and descriptions for the specific logging transactions it handles.
* Sends all log transactions to the Log Aggregrator
  + Logs are sent to the running Rsyslog software utility
* Notifies an operator when a log cannot be sent
  + Send an alert when the sending of the log to Rsyslog fails

**Database Design**

**[TODO: Create Schema for our Postgres Database with explanation ]**

**Overview of what we plan to design:**

No data has been provided by our sponsor, so we have decided to generate our own datasets. This will include information about *Customers* for an organization. However, some columns and tables will be at different security levels, such as *Social Security Number* column for a customerbeing *top-secret* information. The tables and their relationships are still in development. However, we will design these in such a way we can make sure we are thoroughly testing data different security sensitivities and categories of several users with unique security levels.

#### Our Postgres database will be configured with SELinux labels protecting columns and tables. We intend to design a database which is not overly complicated beyond the scope of this project, but still manages to cover the following cases:

#### A table with uniform, highest-level protection

#### A table with uniform, lower-level protection

#### A table with mixed-protection columns

#### One-to-one, one-to-many, and many-to-many relationships, covering

#### Protected unique ids

#### Uniform protection between related tables

#### Heterogeneous protection between related tables

#### Heterogeneous protection within related tables

#### Implementation

|  |  |
| --- | --- |
| Primary Author | Daniel Mills |
| Secondary Author | Spencer “Spencer ‘Spencer Yoder’ Yoder” Milton Yoder |
| Editor | Spencer |

#### Current Implementation Status

#### Iterations

#### Iteration 0

* Research the technologies pertinent to our system
* Create proper requirements and design documents
* Acquire access to the resources needed for development

#### Iteration 1

* Design, configure, and populate a properly labeled sample database
* Configure a docker contained with the proper technologies installed
* Create the infrastructure to facilitate a single, successful REST API call

#### Iteration 2

#### Iteration 3

#### Security Considerations

TODO: Explain how SELinux labels work and how they improve security

TODO: Explain how using Docker containers improves security.

#### Test Plan & Results

|  |  |
| --- | --- |
| Primary Author | Caleb Boswell |
| Secondary Author | Jonathan Balliet (System Test Plan) |
| Editor |  |

#### Overview:

Blackbox testing tool - We are planning to use pyhttptest, though the decision has not been finalized.

Whitebox testing tool - We will unittest for Python, though the decision has not been finalized.

Expected Coverage - We are anticipating 70% coverage as recommended by the sponsors.

System - The system of the project will be tested by automating the REST API calls

Security - For security it will be necessary to test every combination of access control. For example we will use three users, one with low level security access, one with high level security access and the administrator access, and have each user try to access data from each level of security.

Unit - The unit testing will not need to be as extensive as the testing security as it will not directly correlate to the security of the project the more testing is covered in it.

**System Test Plan**

*Testing Overview*:

TBD

*Test Data Summary*:

The following users exist in the system:

***Tester #1***

**Username:** tester

**First Name:** Tester

**Last Name:** One

**Attributes:**

**sensitivity\_range:** “s0-s4”

**categories:** “c0,c12,c42”

***Tester #2***

**Username:** testertwo

**First Name:** Tester

**Last Name:** Two

**Attributes:**

**sensitivity\_range:** “s0-s6”

**categories:** “c2.c15”

The database contains the following tables and data:

TBD

[NOTE: This Test Plan is very rough. We will be able to better formulate this after finalizing our designs for the database and API calls ]

|  |  |  |  |
| --- | --- | --- | --- |
| **Test ID** | **Description** | **Expected Results** | **Actual Results** |
| **authorizedRead1** | **Preconditions:**  Postgres Service is running in the Container Runtime.  Service is connected to the Data Storage with the security levels of Tester1.  **Steps:**   1. Send REST request to read data that has a sensitivity of ***s4*** and category of ***c0***. 2. Await response 3. Send REST request to read data that has a sensitivity of ***s4*** and category of ***c12***. 4. Await response. 5. Send REST request to read data that has a sensitivity of ***s4*** and category of ***c42***. 6. Await response. | Response with an OK status and the data is returned.  Response with an OK status and the data is returned.  Response with an OK status and the data is returned. |  |
| **authorizedRead2** | **Preconditions:** Postgres Service is running in the Container Runtime.  Service is connected to the Data Storage with the security levels of Tester2.  **Steps:**   1. Send REST request to read data that has a sensitivity of ***s6*** and category of ***c2***. 2. Await response. 3. Send REST request to read data that has a sensitivity of ***s6*** and category of ***c8***. 4. Await response. 5. Send REST request to read data that has a sensitivity of ***s6*** and category of ***c15***. 6. Await response. | Response with an OK status and the data is returned.  Response with an OK status and the data is returned.  Response with an OK status and the data is returned. |  |
| **unauthorizedRead1** | **Preconditions:** Postgres Service is running in the Container Runtime.  Service is connected to the Data Storage with the security levels of Tester1.  **Steps:**   1. Send REST request to read data that has a sensitivity of ***s5*** and category of ***c0***. 2. Await response. 3. Send REST request to read data that has a sensitivity of ***s3*** and category of ***c1***. 4. Await response. | Response with an *Access Denied* status is returned  Response with an *Access Denied* status is returned |  |
| **unauthorizedRead2** | **Preconditions:** Postgres Service is running in the Container Runtime.  Service is connected to the Data Storage with the security levels of Tester2.  **Steps:**   1. Send REST request to read data that has a sensitivity of ***s7*** and category of ***c2***. 2. Await response. 3. Send REST request to read data that has a sensitivity of ***s4*** and category of ***c1***. 4. Await response 5. Send REST request to read data that has a sensitivity of ***s4*** and category of ***c16***. 6. Await response. | Response with an *Access Denied* status is returned.  Response with an *Access Denied* status is returned.  Response with an *Access Denied* status is returned |  |

#### Task Plan

|  |  |
| --- | --- |
| Primary Author | Caleb Boswell |
| Secondary Author |  |
| Editor |  |

Project Scope: The immediate focus of our portion of the project we are working on is the REST API and the containers for storing and retrieving the data. We will be focusing on making sure the processes are secure when users are authorized and when retrieving data from the containers so that they are only able to retrieve data within their security level. Afterwards we will begin testing API and security of our implementation. We will also be personally constructing a GUI for the project for presentation use.

Members and Responsibilities:

Name: Spencer Yoder

Email: [smyoder@ncsu.edu](mailto:smyoder@ncsu.edu)

Phone: 919-818-7079

Role: Data Storage Team

Name: Jeen Shaji

Email: jshaji@ncsu.edu

Phone: 747-229-8907

Role: Container Runtime Team

Name: Daniel Mills

Email: dcmills2@ncsu.edu

Phone: 919-909-4330

Role: Data Storage Team

Name: Jonathan Balliet

Email: jballie@ncsu.edu

Phone: 919-608-3331

Role: Container Runtime Team

Name: Caleb Boswell

Email: caboswel@ncsu.edu

Phone: 336-816-8172

Role: Container Runtime Team

Dates: Task planning is expected to be finalized by Feb 14. The time before that will be spent producing a rough draft of iterations, division of roles during each iteration, and an estimated due dates for implementation and testing of the project. The final deliverable is expected to be accomplished before April 26th when the project is to be presented.