Intelligent Transportation System (ITS) Cybersecurity Specs and Apps

System Requirements and Specification

www.its.dot.gov/index.htm

Report –October 31, 2024

FHWA-JPO-24-139

Source: Getty Images.

U.S. Department of Transportation

Produced by Contract: 693JJ322A000005, Operations V
U.S. Department of Transportation
Office of the Assistant Secretary for Research and Technology
Federal Highway Administration

Notice

This document is disseminated under the sponsorship of the U.S. Department of Transportation in the interest of information exchange. The U.S. Government assumes no liability for the use of the information contained in this document.

The U.S. Government does not endorse products or manufacturers. Trademarks or manufacturers' names appear in this document only because they are considered essential to the objective of the document. They are included for informational purposes only and are not intended to reflect a preference, approval, or endorsement of any one product or entity.

Non-Binding Contents

The contents of this document do not have the force and effect of law and are not meant to bind the public in any way. This document is intended only to provide information to the public regarding existing requirements under the law or agency policies. However, compliance with applicable statutes or regulations cited in this document is required.

Quality Assurance Statement

The Federal Highway Administration (FHWA) provides high-quality information to serve Government, industry, and the public in a manner that promotes public understanding. Standards and policies are used to ensure and maximize the quality, objectivity, utility, and integrity of its information. FHWA periodically reviews quality issues and adjusts its programs and processes to ensure continuous quality improvement.

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
FHWA-JPO-24-139		

4. Title and Subtitle	5. Report Date
Intelligent Transportation System (ITS) Cybersecurity Specs and Apps: System Requirements and Specification	October 31, 2024
	6. Performing Organization Code HOIT
7. Author(s) Kyle Rush, Brian Russell, Joseph Dang, Jonathan Thai, Robert Sanchez (ORCID: 0000-0002-0763-6146)	**8. Performing Organization Report No.**
9. Performing Organization Name and Address Leidos Inc. 1750 Presidents Street Reston, VA 20190	**10. Work Unit No. (TRAIS)**
	11. Contract or Grant No. 693JJ322A000005
12. Sponsoring Agency Name and Address Federal Highway Administration 1200 New Jersey Avenue, SE Washington, DC 20590	**13. Type of Report and Period Covered** Final Report, September 2022-October 2024
	14. Sponsoring Agency Code HOIT

15. Supplementary Notes

The task order manager is Usman Ali.

16. Abstract

This report presents the field device security configuration tool prototype system requirement and specification (SRS) for the intelligent transportation system (ITS) Cybersecurity Specs and Apps task order. The SRS describes the functional and non-functional requirements, such as portability, reliability, security, performance, availability, and manageability. The specification includes the functional requirements and designs for the system architecture, user interface design, cloud design, and cybersecurity schema. The field device security configuration tool prototype will provide the ITS vendor community with a reference design and example implementation of a mobile application and backend cloud service that can be used to guide ITS vendor customers in the secure configuration of ITS equipment. The tool will be used by ITS technicians as an aid, providing them with the equipment vendor's recommended security configuration details for the specific equipment type, connection type and intended use. The tool is a mobile application that an ITS technician can install on either an Android or iPhone device. This tool is powered by a secure and flexible backend cloud service that provides secure data in transit from a database to a technician's mobile device.

17. Keywords	18. Distribution Statement
System Requirements Specification, SRS, Cybersecurity, Apps, Intelligent Transportation System, ITS	No restrictions. This document is available to the public through the National Technical Information Service, Springfield, VA 22161. http://www.ntis.gov

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	82	N/A

Table of Contents

Contents

List of Figures

List of Tables

Executive Summary

The field device security configuration tool prototype will provide the intelligent transportation system (ITS) vendor community with a reference design and example implementation of a mobile application and backend cloud service that can be used to guide ITS vendor customers in the secure configuration of ITS equipment. ITS vendors will be able to use the prototype to design their own custom versions of the mobile application and offer the application to their customer base.

Vendor-specific field device security configuration tools will be used by ITS technicians as an aid, providing them with the equipment vendor's recommended security configuration details for the specific equipment type, connection type and intended use. The field device security configuration tool is a mobile application that an ITS technician can install on either an Android or iPhone device. This tool is powered by a secure and flexible backend cloud service that provides secure data in transit from a database to a technician's mobile device.

The cybersecurity schema will serve as a higher-level guideline that delivers explicit recommendations to members of the ITS Vendor community that are establishing technological infrastructures or software components. This section details best maintenance and software initialization practices that promote the development of secure and threat-aware systems. The schema prepares the Vendor community with the right configuration for tools to neutralize and identify cybersecurity vulnerabilities.

Chapter 1. Introduction

1.1 Purpose and Scope

The field device security configuration tool system design document provides a brief concept of operations and describes the system architecture, operating environment, user experience design, cloud deployment design, and cybersecurity schema.

1.2 System Overview

The field device security configuration tool system will consist of interactions between an ITS equipment vendor and an ITS technician (user) through the use of two devices. ITS equipment vendors will upload security configuration recommendations to a cloud service. ITS technicians will use mobile devices to download the specific security recommendations for an ITS device and display those recommendations during the configuration process.

Chapter 2. Concept of Operations

The field device security configuration tool will enable the easy to use, efficient, and secure dissemination of security configuration recommendations from vendors and manufacturers to technicians in the field working with the devices. This section will lay out a few sample use cases of the tool and enumerate the key groups of people who may interact with the tool and the ways they will do so.

2.1 Key System Personas

A persona in this case refers to a high-level description of a person or group of people who will fill a particular role within the systems operation, be they users or maintainers, etc. the kinds of things they look to achieve or get out of the system, as well as the kinds of experience or knowledge they bring with them into their interactions with the system.

The important roles identified for this system include:

- Field technicians – End users of the mobile application of the tool. Generally assumed to be knowledgeable on basic operations of the field devices and capable of adjusting security configuration parameters as recommended by the tool. The field technician is looking to improve the security of the field devices they manage in an easy and time-efficient manner.
- Vendor/manufacturer – The group within the system responsible for producing the recommended security settings for their devices. The vendors/manufacturers are assumed to be knowledgeable on cybersecurity best practices to protect their devices for any operating conditions.. They want to use the tool as a means to improve the security of devices deployed in the field while safeguarding any proprietary data or data that might be used by threat actors to exploit their devices.
- IOOs/DOTs management – IOOs/DOTs management are the group of decision-makers within IOO or DOT organizations looking to harden the cybersecurity of field devices deployed under pilot programs or as part of other ITS initiatives. They want to mitigate the risk of potential cybersecurity incidents that could result in data compromise, system downtime, or failure of safety-critical systems. IOOs/DOTs management are looking to achieve the cybersecurity goals in a cost-efficient and reliable manner.
- Application development and administration team – The application administration team is the team managing the deployed instance(s) of the field device security configuration tool. This is the team responsible for operations of the tool such as deploying it to the runtime environment, configuring the tool and the cloud services that host it, approving user accounts for field technicians and vendors/manufacturers, triaging user issue reports, and issuing bug fixes. This team is assumed to be familiar with the source code of the application and general software development practices necessary to fix bugs and add features as needed. This user group may in practice be part of the IOO/DOT organization structure or part of the device vendor/manufacturer organization depending on how the application is being deployed.

These core four user groups make up the key personas within the field device security configuration tool concept of operations.

2.2 Use Case/Concept of Operations

In the basic concept of operations for this system, the application has been deployed by a vendor to enable field technicians to access data related to that specific vendor/manufacturer's devices. To do so, the vendor has cloned the open-source field device security configuration tool's architecture and/or source code, performed an internal review of the code to audit for any security concerns and to identify any changes (branding, user experience, corporate policies, etc.), and deployed their modified or unmodified version of the application to server resources favored by the vendor/manufacturer. The vendor/manufacturer may also choose to make changes to the mobile application as well and distribute this app in any way they see fit. In such a manner, the vendor/manufacturer (or an internal team within the vendor/manufacturer) has assumed the role of the application administration team as well.

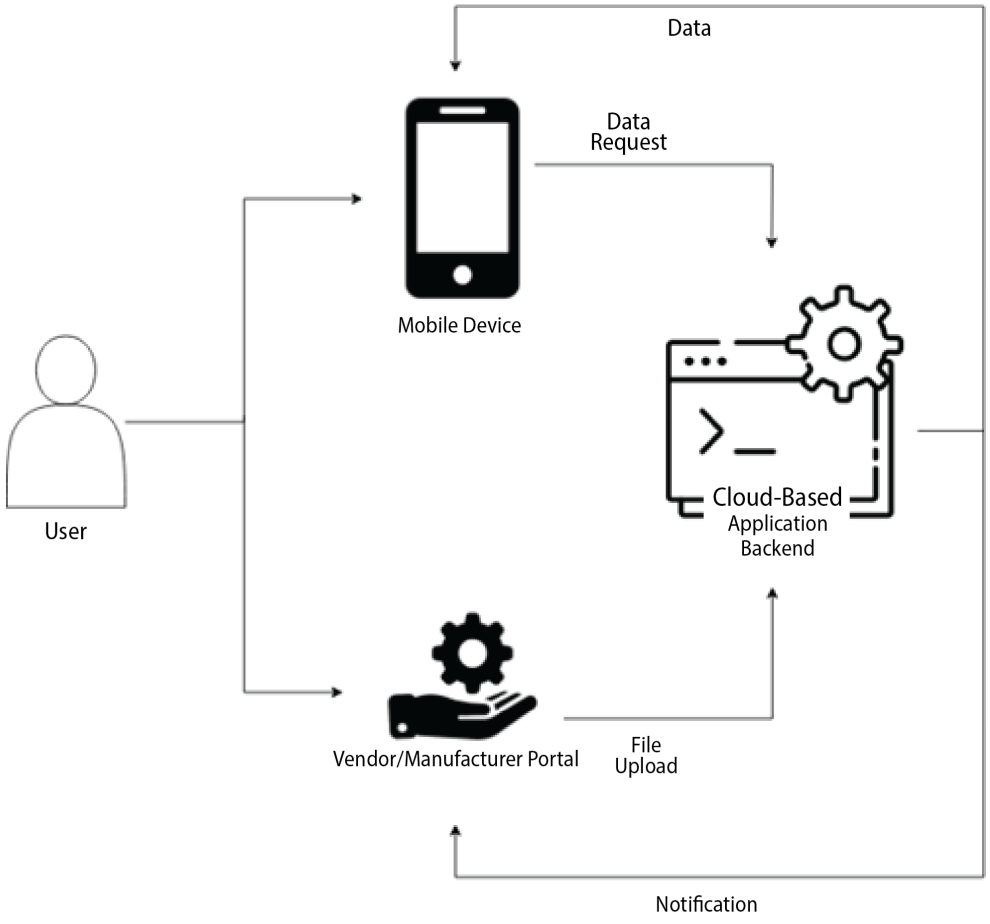
In addition to the application administration team responsibilities above, the vendor/manufacturer team would have the same responsibilities as outlined in the personas list. They would provide their own configuration data into the server instance they have deployed such that they can be accessed and used securely by authorized field technicians with confidence that the data are encrypted in transit and at rest.

A field technician looking to configure such a vendor's device would apply for an account to access the vendor's instance. If approved, the field technician would then download and install the vendor's version of the mobile application, and log in with their new user credentials. This would enable the field technician to then access the configuration recommendations for that vendor/manufacturer's devices, at which point the technician would be able to use those recommendations to reconfigure the device as they normally would.

Within the concept of operations, IOOs/DOTs would likely instruct their field technicians (or the technicians of contractors via contract language) to follow the configuration recommendations provided by these vendors.

Chapter 3. System Architecture

The field device security configuration tool system will consist of interactions between an ITS equipment vendor and an ITS technician (user) through the use of a mobile application and a cloud service-based application backend as shown in figure 1. ITS equipment vendors will upload security configuration recommendations to a cloud service. ITS technicians will use mobile devices to download the specific security recommendations for an ITS device and display those recommendations during the configuration process. The field device security configuration tool system is composed of a backend cloud service, a mobile application, and a set of Application Programming Interfaces (API).



Source: FHWA.

Figure 1. Diagram. System Concept.

3.1 Mobile Application

The goal of the field device security configuration tool is cross-platform support for both Apple and Android devices. To achieve this, Flutter was chosen as the mobile development framework. Flutter is a cross-platform framework that allows the field device security configuration mobile application to be ported to both iOS and Android. This allows vendors to develop the application without the need for creating and maintaining multiple codebases as would be required using native development tools.

The Flutter framework uses the programming language Dart and has built-in widgets that are used to design each of the different sub-components of the mobile application. These widgets include rows, columns, sized Boxes, and more. The framework allows for screens/views/pages of the application to be constructed out of these widgets by arranging them in a top-down sequence on the screen

To connect the mobile application to the backend database, Amazon Web Services (AWS) services was chosen. After the field technician user has entered their valid credentials, the user will be redirected to the home screen. From there, the schema will automatically detect and validate which files should be downloaded. With IAM and Cognito user pools from AWS, the user will only receive and display devices that are authenticated to the user. Cognito follows a least-privileges principle, so users who are not authenticated with any files will not see any devices on the home screen. Otherwise, after successful retrieval, the file will be downloaded from S3 Storage. This file is a JSON file and has the field of Device. From there, to be able to call the JSON file wherever on the application, or to use it offline, Flutter Secure Storage will be used. The JSON file will be stored locally in a secure storage and can be called to display to the mobile application screen.

3.1.1 Flutter Packages

The following are some of the Flutter packages that will be used to implement the mobile application. These packages provide prebuilt functionality to enable some of the features of the application, either in whole or in part:

- Gangadhare, A, (2021). Flutter Barcode Scanner 2.0.0 [Package]: https://pub.dev/packages/flutter_barcode_scanner
- Lachdef, S, (2022). Dropdown Search 5.0.5 [Package]: https://pub.dev/packages/dropdown_search
- Flutter Dev, (2023). Go Router 6.0.2 [Package]: https://pub.dev/packages/go_router
- Yako.io, (2022). Settings UI 2.0.2 [Package]: https://pub.dev/packages/settings_ui
- srtraj, (2023). Dropdown Textfield 1.0.8 [Package]: https://pub.dev/packages/dropdown_textfield
- Flutter Dev, (2023). Shared Preferences 2.0.17 [Package]: https://pub.dev/packages/shared_preferences
- AWS Amplify, (2023). Amplify API 0.6.12 [Package]: https://pub.dev/packages/amplify_api
- AWS Amplify, (2023). Amplify Datastore 0.6.12 [Package]: https://pub.dev/packages/amplify_datastore
- AWS Amplify, (2023). Amplify Storage S3 0.6.12 [Package]: https://pub.dev/packages/amplify_storage_s3/

- AWS Amplify, (2023). Amplify Auth Cognito 0.6.12 [Package]: https://pub.dev/packages/amplify_auth_cognito
- Firebase Google, (2023). Firebase Core 2.4.1 [Package]: https://pub.dev/packages/firebase_core
- Flutter Dev, (2023). Path Provider [Package]: https://pub.dev/packages/path_provider
- Ponnam K, (2023). Fluttertoast 8.2.1 [Package]: https://pub.dev/packages/fluttertoast
- Flutter Community, (2023). Connectivity Plus 3.0.3 [Package]: https://pub.dev/packages/connectivity_plus

3.2 Features and Functionality

Table 1 summarizes the mobile application features and functionality.

Table 1. Mobile Application Feature Functionality.

Feature	Function
Register User Account	Allow technicians to register an account
Multi-Factor Authentication	Enforces multi-factor authentication for access to mobile applications
Login	Allow technician to login to their accounts
Forgot Password	Allow technicians to reset their passwords if password is forgotten
Scan ITS Equipment QR Code	Allow technicians to scan a QR code to find the ITS equipment
Mode Selection	Expert or wizard walk through
Wizard Walk Through of Security Configurations	Technicians will receive an aid to help them configure the device; this mode is the default (automatic) mode selected when the application starts
Expert Mode Configuration	Index of all configuration options (self-guided)
Search	Allow users to search through a list of products
Device Selection	Allow technicians to swipe through a list of devices to select; alternatively, the technician can simply scan the QR code on the search bar and automatically identify the device vendor, type, and model
Vendor/Model Selection	Allow technicians to select the vendor and their model; alternatively, the technician can simply scan the QR code on the search bar and automatically identify the device vendor, type, and model
Report an Issue	Allow technicians to report an issue regarding the app or with any inquiries
Voice Recognition	Use phone's built-in functionality
Get Configuration	Update the latest configuration data from the server
Logout	Allows a user to log out of the mobile application
Screen Timeout	Automatically times out the screen after a configurable period of inactivity
Encrypted Data in Transit	Encryption of mobile to cloud interface
Encrypted Data at Rest	Encryption of the configuration data stored on the mobile device
Re-verification for security configuration	Prompts the user to reauthenticate to the mobile phone upon mobile application timeout

3.2.1 Backend Application Support

This application utilizes AWS Amplify and its libraries to manage users and content. This includes authentication, REST API, and GraphQL.

AWS Amplify Authentication

The application developer should run the following command in the project's root folder: # *amplify add auth*.

After selecting the default configurations, the developer should run *amplify push* in order to set the changes. The developer should install any dependencies that Amplify requires. When installation is complete, the developer can import Amplify Auth Cognito to utilize it in the application.

AWS Amplify Storage

The application developer should run the following command in the project's root folder: # *amplify add storage*.

Depending on the application's specifications, the developer may choose their desired options. The developer should install the Amplify Storage S3 dependency and after that may import it to the code.

3.2.2 Offline Capabilities

The application will be able to be used in an offline setting. Flutter Secure Storage offers offline capabilities to fetch or update data from the local storage. To prevent any loss of data, or inability to use the application, the state of the connection to the server will be checked prior to loading the security configuration data. If a connection is available, the mobile application will check for updates to the existing configuration data and download them if needed. If no connection is available, the mobile application will load data from the local device's secure storage. Flutter Secure Storage is used to store data in secure storage. It uses AES encryption for Android with a key stored in KeyStore. Key size options for AES within Flutter Secure Storage depend on the platform. Android devices support AES with 128- or 256-bit keys. For iOS, AES key sizes can range from 128, 192, 256 or 512 bits.OniOS, Flutter Secure Storage uses keychain encryption.

After successfully authenticating with a connection for the first time, the application will check for the local storage that contains data. If it is unable to find a file from storage, then the application will download the file from the S3 database. From there, it will save the file to local storage, and display the file to the application. The vendor will provide an offline secure storage key for each mobile application.

If there is no network connection established on the device, then the application will not try to download from the S3 database and will instead try to pull from the local secure storage. After successful retrieval, the application will then continue to function as intended.

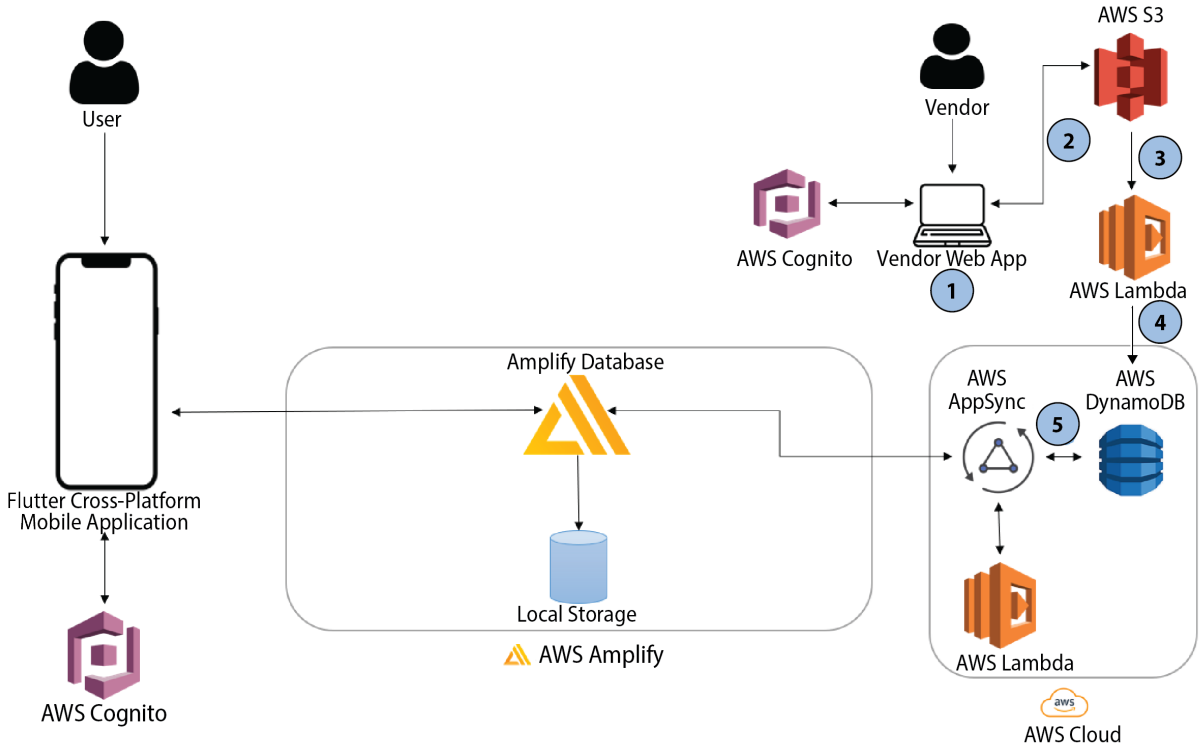
3.2.3 Technician API

The mobile application API (also referred to as the technician API), allows the mobile application to pull configurations from the cloud database. These configurations, which are in the form of JSON files, are

then processed and reformatted by Flutter, which also handles displaying the file data in the application itself.

3.3 Application Backend

The application backend (see figure 2) is core logic of the field device security configuration tool that is deployed into a cloud compute environment and handles the upload, download, storage, validation, and access management of the device configuration data and user accounts. In this prototype application the application backend makes use of AWS functionality for individual components of the application backend such as user account management, permissions management, data encryption and data storage. The services used for these core functionalities are specifically the AWS implementations, but other implementations of these functionalities exist for other cloud platforms. The AWS implementations are chosen due to the development team's existing familiarity with AWS.



Source: FHWA.

Figure 2. Diagram. Application Backend Architecture.

The services used are described below:

3.3.1 AWS Amplify

AWS Amplify provides the system with simplified full stack web app deployment with little to no overhead for engineers. The provided webapp natively links with other AWS components allowing for effortless prototyping and testing of other features in the architecture. Amplify also provides built-in authentication methods with React and AWS Cognito, which can be used to make sure system permissions work properly across different user account types.

3.3.2 AWS Cognito

AWS Cognito bears the responsibility of segmenting system permissions through user groups and identity pools. These groups and pools authenticate and restrict webapp and mobile application access for its users. User groups follow least-privileges principles by designating roles with specified permissions to each group. Upon authentication with Cognito on either the webapp or mobile application, users are then assigned to the most appropriate group that permits actions only fit for their usage case. For example, an admin user group would have read and write permissions while a regular user group would only have read permissions.

3.3.3 AWS Authentication

AWS authentication provides the application with a streamlined, secure authentication flow. AWS Cognito allows the app to control which accounts can login to the app and restrict users on what files the user can see. With AWS authentication, the app can utilize prebuilt UI components and even social identity provider logins. These prebuilt components also include account creation and forgotten password features.

3.3.4 Amplify DataStore

Amplify DataStore is a persistent on-device storage repository for developers to write, read, and observe changes to data. All DataStore operations are local first. This means that when a query is run, it returns results from the local system, which can be sorted and filtered. The same is true for mutations or observations to data. So, no network latencies or constraints on the backend are present.

3.3.5 AWS AppSync

When paired with AWS AppSync, Amplify DataStore synchronizes the application data with an application programming interface (API) when network connectivity is available. Amplify DataStore automatically leverages AWS AppSync to achieve near real-time synchronization between the devices and the cloud backend.

3.3.6 AWS DynamoDB

Amazon DynamoDB is a key-value and document database. Using the CLI, developers can automatically create tables representing the schema defined at the application level on DynamoDB.

3.3.7 AWS Lambda

AWS Lambda provides serverless, cost-efficient cloud computing in the form of Lambda functions, which are linked to AWS APIs that have been created to send and receive data. There are two Lambda functions in this system that handle API logic for technicians and vendors. The technician Lambda function, (which is to be used with the technician API) extracts configuration data from the cloud and sends it to the technician API to be displayed to users. The vendor Lambda function handles configuration uploads and updates and is initialized using the vendor API.

3.3.8 AWS S3

AWS S3 serves as the database. Once the user is authenticated through Cognito and logged into the mobile application, the app will check if a connection exists. If a connection is available to be connected to and there is no data in Flutter Secure Storage, the application will then download the configuration data from AWS S3 and save it to Secure Storage for offline usage.

3.3.9 GraphQL

GraphQL is used to facilitate the data modeling process by writing schemas. Once the schema is defined, domain native structures called models, are generated. Your developers can then use the DataStore API to save, query, update, delete, or observe changes.

3.4 Vendor/Manufacturer Portal

The vendor/manufacturer portal is the webpage(s) that allow the vendor/manufacturer users to register for accounts, upload and validate configuration recommendations, and manage users privileges to access configuration data for specific devices. This portal will act as a front-end interface to the vendor/manufacturer functionalities implemented in the application backend. The vendor/manufacturer portal will be implemented on top of the React web framework to enable a modular and adaptable software package. Upon initially accessing the portal, the user will be prompted to login or apply for an account. By applying for an account, the user will send the necessary registration information to the application administration team for review and approval or denial. If the user is approved or an account or already has an account they may log in to the portal and manage both users and configuration for devices. The portal will provide functionality that will allow the user to upload JSON files in the cybersecurity schema described below. This schema enumerates and describes the different recommendations that can be defined and their possible values. The vendor/manufacturer user will generate such a JSON file describing the recommended security settings for a device they wish to upload (generating the file itself is left as an exercise for the user) and will upload it via the vendor/manufacturer portal. After the file has been uploaded the vendor will be able to manage, update, and delete the recommended settings for that device. Additionally, the portal will provide an interface to enable the vendor/manufacturer user to authorize specific users or groups of field technician users to access security configuration recommendations of specific devices. Table 2 shows the vendor/manufacturer portal features.

Table 2. Vendor/Manufacturer Portal Features.

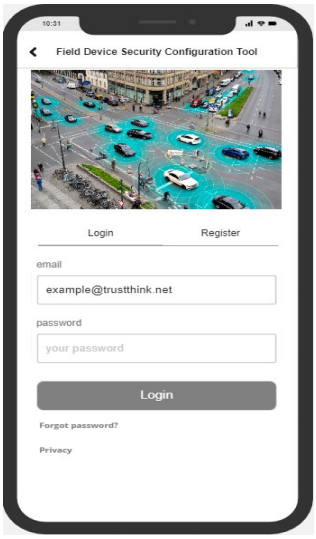
Feature	Function
Register Organization Account	Allows either a vendor or SLTT agency to register an organization; users may be mapped to the organization
Manage User Account	Allows an administrative user to manage user accounts
Manage Organization Account	Allows an administrative user to manage organization accounts
Manage User Account Permissions	Allows an administrative user to manage user permissions on the mobile application
Maintain Schema	Allow vendor to upload and manage versions of schema in the database
Filter Submissions (schema alignment)	Configuration submissions will be validated against schema via API
Password Configuration Management	Allow vendors to secure their device
Login	Allow technician to login to their accounts
Forgot Password	Allow technicians to reset their passwords if password is forgotten
Logout	Allows a user to log out of the mobile application.
Screen Timeout	Automatically times out the screen after a configurable period of inactivity
Encrypted Data in Transit	Encryption of mobile to cloud interface
Re-verification for security configuration	Prompts the user to reauthenticate to the mobile phone upon mobile application timeout

Chapter 4. Mobile Application Design

4.1 Login Page

The application should be initialized with Amplify plugins and Firebase options. After successful configuration of Amplify, the application will redirect to the login page (shown in figure 3). At the login page, the page was designed with boilerplate code provided by AWS. This code should be able to create an account, reset passwords, and log in to the application. The login page should also create the GoRouter, which utilizes a URL-based API to navigate between different screens.

The login page has been designed with AWS authentication boilerplate code, as it streamlines the process of creating a login screen and provides quick authentication. The UI consists of an email text input and password text input. It should also have a button for account registration and password recovery. The login system operates using AWS Cognito which offers SMS text message codes and time-based one-time password in order to support multi-factor authentication.[*]



Source: FHWA.

Figure 3. Illustration. Login Page.

[*] Developer Note: At the time of prototype development, adding biometric authentication has been discussed but not implemented due to time constraints. To add biometric authentication to the mobile application, it is recommended to evaluate Flutter's OS-level authentication with biometrics package. This would enable a second layer of biometric authentication on top of Cognito's email and password login.

4.2 Home Page

After successful authentication, the user will be redirected to the home page (see figure 4), where the application will automatically download a file or files containing the security configuration recommendations (provided the user has appropriate permissions) which will have its data displayed to the screen. If the authentication is unsuccessful, the user will not be able to access any data and there will be a brief timeout prior to being able to attempt to login again. This timeout should be long enough to deter potential attackers from utilizing brute-force techniques to guess a password but should not be arduously long that a user would be locked out for a simple typo in their password. The file should be read as a JSON file with fields defined according to the cybersecurity schema described later in this document.

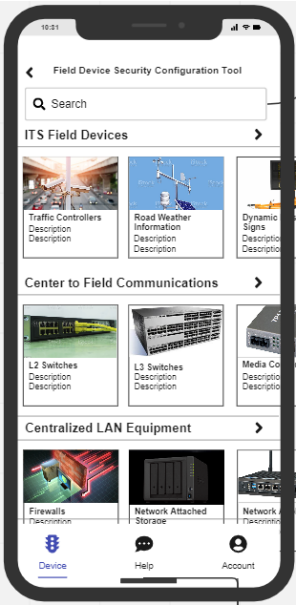
The JSON file should be parsed into in-memory data structures and objects that describe the recommendations and devices in ways that the mobile application can consume and render to the user. These objects should include:

- SearchItemList()
    - doesItemExist(String str) - returns the Boolean if the item exists in the hashmap
    - add(SearchItem searchItem) - adds a SearchItem to the searchList array
    - getList() - get the list of SearchItems.getSearchItemFromString(String str) - returns a SearchItem from a string
    - reset() - resets the SearchItems list whenever the user signs in again
- SearchItem(name, model) - This object takes in two parameters of name and model. Name which should be the name of the vendor or manufacturer. Model which should be the model of the device. SearchItem should also have the methods:
    - getModel() - returns the model
    - getName() - returns the name
    - getFullNameList() - returns the list of appended string of name and model
    - getFullName() - returns a list of appended strings of name and model
- serialNumber – A hashmap of key “serial_number” and value “searchItem_name searchItem_model”
- DeviceList()
    - getList() - returns the devices list
    - reset() - reset the devices list
    - add(Device device) - adds a device to the devices list
- Device(title, category, type, subtype, description, imageUrl) - This object takes in the parameters title, description, and imageUrl. With the methods:
    - getName() - returns the name
    - getDescription() – returns the device description

- getCategory() – returns the category of the device, category being a string value but intended to represent a broad grouping of devices such as devices for roadside, devices for intersection, device for TMC, networking devices, etc.
- getType() – returns the type of the device, such as camera, router, etc.
- getSubType() – returns the subtype, within the type, such as infrared camera
- getImageUrl() – returns the URL to be used to download an image to display associated with the device

After the objects are created and are successfully parsed, the application should then create a ListView, on the main page of the user interface, which builds a list of items taken from an array. These items are referred to as cards, where each card is built from a SizedBox. Each SizedBox contains a ImageUrl, which is the image of the device, the description, and the name of the device. The SizedBox also has an InkWell widget, which allows for a user to be able to tap on the card. On tap of a specific card, it will redirect the user to the device details page, while passing device.title.

The application takes advantage of the DropdownSearch library which streamlines creating a search bar. The search bar takes in a field, Items, and the application passes in SearchItem.getFullNameList(), which returns every vendor appended to the model's name. As an example, the search list should be populated with this format “Vendor Model.” A feature with DropdownSearch is that it provides a search bar that automatically recognizes user input and filters the list depending on the input. After selecting a device, this will provoke the onChanged method and the string is then parsed to the values model and vendor. The method that is used is getModel() and getName(), after retrieving the searchItem with getSearchItemFromString. With that the application will route to the operational context page, passing the parameters “device,” “vendor”: getName(), “model”: getModel(). The search bar will also have a QR barcode scanner button.



Source: FHWA.

Figure 4. Illustration. Home Page.

To create the serial lookup feature, the application should utilize a hashmap. A hashmap makes it easy to format key, value pairs. In the application, the user should be able to tap a button which displays a dialog with a TextField and TextFieldController. The TextFieldController, will take in the string from TextField and which will be passed into the serialNumber hashmap. Upon verification, if the item does exist then the vendor and model fields should be routed to the operational context screen. Otherwise, a Toast that states that the device was not found should be displayed.

To implement the QR scanner, an outside library was used. Upon tapping the Scan QR button, the application will use the library Flutter Barcode Scanner, which mounts a QR scanner to the application. After a successful QR scan, the application will parse the results and pass it on to the operational context page. The QR code should return the vendor's name and the device's name.

The home page should display a NavigationDrawer with the buttons Home, Settings, Report an Issue, and Settings. Each of these buttons should redirect the user to their respective screens. Lastly there should be a Logout button which should clear the secure local storage and sign the user out of the application.

4.3 Settings Page

The settings page (shown in figure 5) should include a button that allows the user to toggle Expert Mode. The state of the button should persist throughout and after the application's lifetime. For Flutter or Android, SharedPreferences may be used to store items in the local storage of the application for persistence. There should also be an Adjust Zoom button available for the user. On tap of the button, the application should redirect the user to the Adjust Zoom screen. Here, there should be a drop-down button of different numbers to set the scale of the drop-down buttons. It should increase the height of the drop-down button by (number set by user) * (drop-down button height). This feature should also persist. To maintain persistence the value should be stored in the local storage of the device.

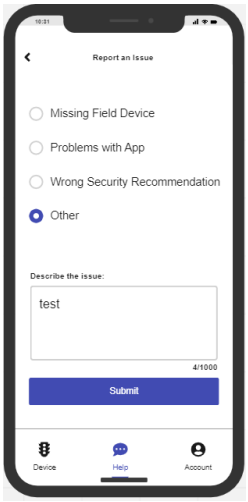
Source: FHWA.

Figure 5. Illustration. Settings Page.

4.4 Report an Issue Page

The report an issue page (see figure 6) should include a drop-down menu specifying the issue (i.e., missing field device, problems with app, wrong security recommendation, or other.) There should also be a TextField that limits the text to 1,200 characters. When the user taps the Send Feedback button, the button should validate the TextField text. If it is greater than 1,200 characters, or if it is empty, then display a Toast stating the description is not filled out correctly. Otherwise, call Firebase Firestore and send a JSON document with the fields: “timestamp,” “feedback,” and “issuetype.” If there is an error with the application, error logs will be sent to Firestore. Reporting is designed to automatically log application status and send that status to the vendor upon submission of an issue. The mobile application uses Crashlytics to send automatic logs to the application administration team whenever an error (fatal or non-fatal) occurs. The application administration team may then share the data with vendors/manufacturers as appropriate.

The report an issue page can also be used to report potential issues with the vendor’s recommended security configurations. If a specific configuration policy recommendation seems incorrect, then the technician can report that issue to the vendor for resolution using Wrong Security Recommendation. Deployers of the field device security configuration tool can also customize the reporting categories based on their specific use case.



Source: FHWA.

Figure 6. Illustration. Report an Issue Page.

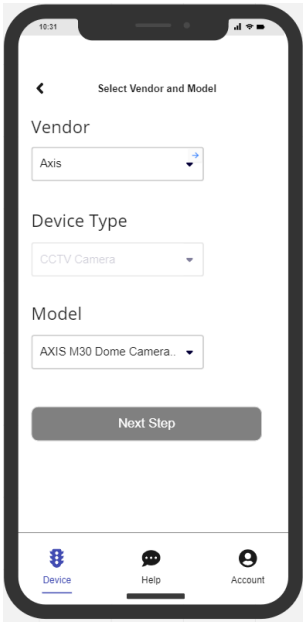
4.5 Devices Details Page

After tapping on a device on the home page (or directly based on QR scan results), the device details page should open and call the secure local storage and retrieve the devices list to load the device configuration information. To display the configuration recommendations to the field technician, the same algorithm from the home page should be used. The algorithm should instead have a conditional, when the selected device from the home page is equal to the devices that the vendor has, then add it to the vendor

name array and hashmap. The application should create a hashmap with the key being the name of the vendor, and the value representing the list of devices the vendor has. There should also be an array of vendor names.

The device details page (figure 7) should have three drop-down menus on screen. They should be titled Vendor, Device Type, and Model. Vendor should contain the list of vendors that have the current device type. These drop-down menus allow the user to provide more specific information to precisely identify a specific model of device if multiples exist (such as if the field technician user has access to configurations for multiple camera type devices). If the deployment of the field device security configuration tool is to support exclusively a single vendor, then the vendor tab will be pre-populated with that vendor and the field technician will be unable to select a different vendor. In Device Type, the device selected from the previous home screen should be displayed, and it should be immutable. The Model should be dependent on the Vendor and Device Type, so it should populate accordingly. The items in Model, should be of hashmap[selectedVendor]!, where the hashmap has the key of the name of the vendor and should return the list of models the vendor has. This state should change, where selectedVendor will update every time a new vendor is selected in the vendor drop-down menu.

An example would be if the vendor was “A” with models “temp1” and “temp2” and another vendor was “B” with models “num1” and “num2.” Then whenever A is selected, it should display temp1 and temp2 under the model drop-down menu. Otherwise, if vendor B is chosen, it should populate models with num1 and num2 only. After selecting the vendor and model and tapping the Next Page button, the application should pass “device,” “vendor,” and “model” as arguments to the next page, operational context.

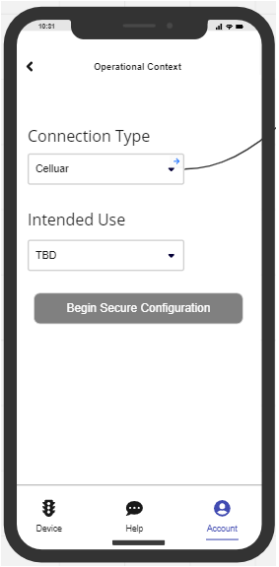


Source: FHWA.

Figure 7. Illustration. Device Details Page.

4.6 Operational Context Page

The operational context page (figure 8) should have two drop-down menus. The first drop-down menu is Connection Type with the items of Cellular, Hardwired, Wifi, and Satellite. The second drop-down menu is Intended Purpose. The screen should redirect the user after tapping the Begin Secure Configuration button which should pass the parameters, “device,” “vendor,” “model,” “connectionType,” and “intendedPurpose.”

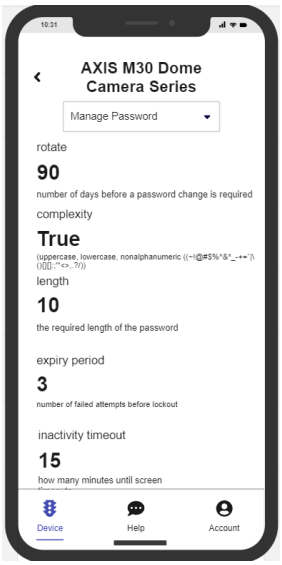


Source: FHWA.

Figure 8. Illustration. Operational Context Page.

4.7 Security Recommendations Page

The security recommendation page (figure 9) should take the queries of “device,” “vendor,” and “model” to find a specific security recommendation to present to the user. The application should display the vendor and model name at the top. Below that should be the current security recommendation. Depending on the user’s Expert Mode preference setting, the application should display one or the other feature. With Expert Mode toggled on, the user should be able to select the desired recommendation from a drop-down menu with a list of recommendations. Otherwise, if Expert Mode is toggled off, then the user should have two buttons of Previous and Next. The buttons need to account for the range of how many recommendations there are. It should not go past index 0 or the length of the recommendation list.

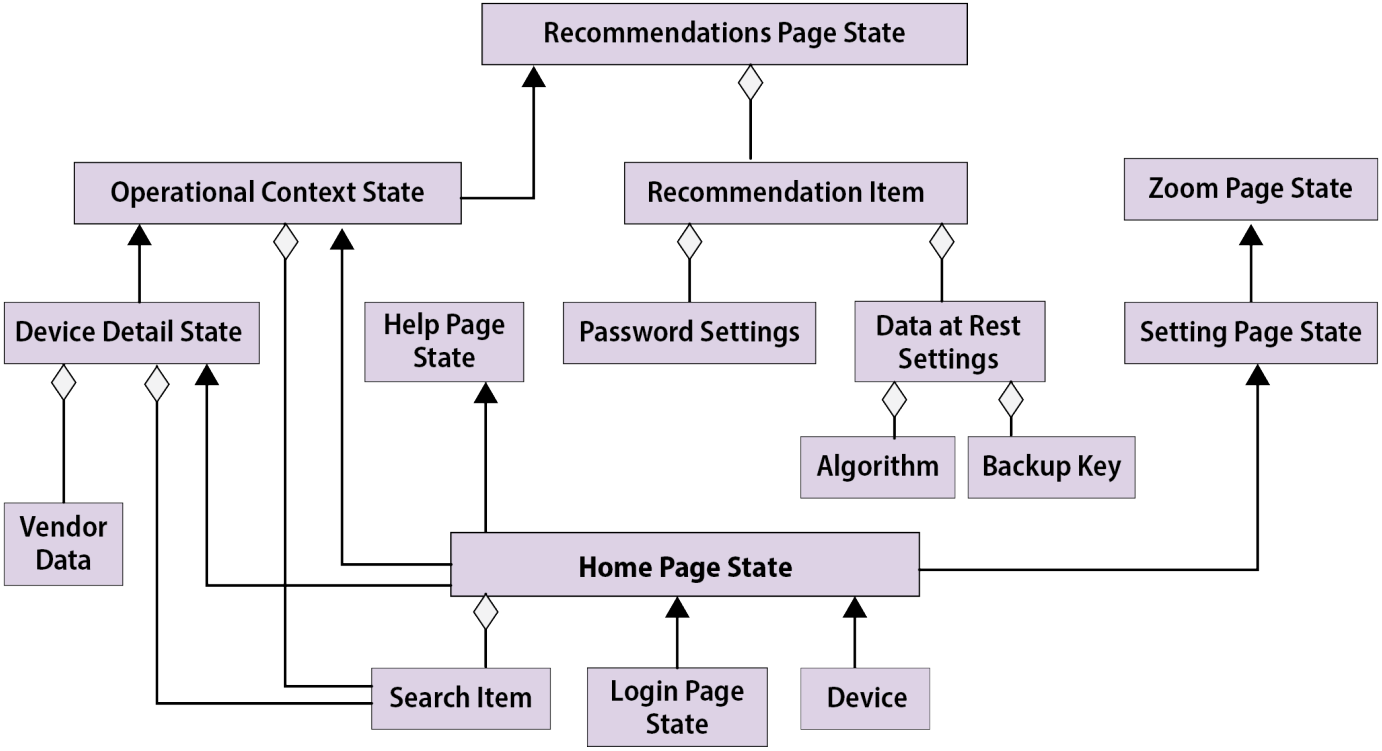


Source: FHWA.

Figure 9. Illustration. Security Recommendations Page.

4.8 Screen Relationships

Figure 10 details the relationships between mobile application screens.

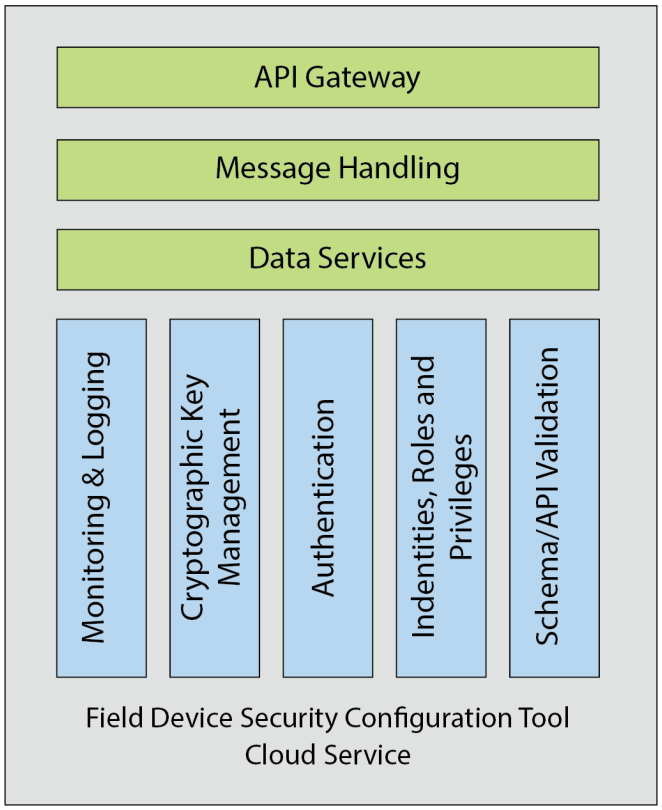


Source: FHWA.

Figure 10. Diagram. Page Relationships.

Chapter 5. Application Backend Design

The generic basis of this application backend/cloud architecture (shown in figure 11) relies on several mandatory components that must be implemented in any system for it to function properly and securely. Serverless cloud computing is necessary in order to provide cost-efficient logic execution and low overhead for engineers. A REST API manager shall be utilized in order to both maintain and create REST APIs to be used with cloud components. To ensure best security practices, an identity management system shall also be established in order to provide least privilege access. This identity management system shall distribute permissions accordingly to higher-management and regular users that interact with the cloud. A key management system shall be implemented to work alongside a user authentication handler in order to further secure access to each cloud component. The key management system shall also encrypt/decrypt objects and secure data at rest. Activity across this architecture shall be monitored through a logging system which keeps track of component interactions and user access. A summary of cloud functions is shown in table 3.



Source: FHWA.

Figure 11. Illustration. Application Backend/Cloud Design.

Table 3. Cloud Functions.

Function	Description
Create Vendor Account	Allow vendors to create account using email and password
Manage Vendor Account	Allow vendors to change account details; credentials
Post Security Configuration	Allow vendors to submit a standardized set of security configurations to the cloud
Encrypt Security Configuration	Securely encrypt configuration data in upload and storage
Version Control Security Configurations	Store different versions of configurations in cloud
Delete Security Configurations	Allow vendors to delete existing configurations in database
Create Vendor User Account	Allow technicians to create accounts using email and password. AWS Cognito supports multi-factor authentication when creating and managing vendor user accounts; both SMS text message-based and time-based one-time passwords are offered as means of additional authentication*
Update Vendor User Accounts	Allow technicians to change account details; credentials
Update Vendor User Account Permissions	Allow vendor accounts to update technician accounts
Update Cloud Encryption Keys	Allows vendors to update cloud encryption keys for database
Delete Vendor User Accounts	Allows vendors and technicians to delete technician accounts
Delete Vendor User Account Permissions	Allows vendors to manage and delete technician account permissions
Delete Cloud Encryption Keys	Allows vendors to delete encryption keys used with their data
Log Security Events	Allows engineers to monitor API and database activity
Assign Vendor Keys to User Account	Allows vendors to distribute permissions to technicians via access keys
Notify Successful Security Configuration Load	Notifies vendor of successful upload after configuration validation in Lambda

* Developer Note: Additional user registration and authentication/authorization options may exist but have not been included in the prototype design due to time constraints. For example, developers may consider supporting passkeys and FIDO authentication for the user registration function.

Function	Description
Error Handling	After failed upload AWS Lambda file validation returns error messages and deletes bad files from database
Forgot Password/Reset	Account password is recovered/reset via email
Multi-Factor Authentication	Technician/vendor links a workplace email to their account. Through AWS Cognito, login authentication can also be configured to require an SMS text message code or a time-based one-time password
GET Security Configuration	Only permits users with appropriate permissions to GET encrypted data from cloud
Segment Vendor Data	Data on the cloud is organized into groups based on specification similarities

This section details a generic system architecture that can be developed using any cloud service provider and then a sub-section that details a specific AWS-based implementation. Developers can choose to create their own architecture using the generic building blocks or re-create the AWS environment detailed herein.

5.1 Vendor API

The vendor API allows vendors to upload, delete, and manage configurations on the cloud database. These configurations are to be stored only in JSON files, which will be validated against a preexisting vendor-provided JSON schema in order to ensure that each upload to the cloud is appropriate and fits the standards of the vendor. Vendors are only allowed to upload .json and .txt files (containing the JSON-formatted cybersecurity schema data) to the database. All other file formats will not be accepted in order to maximize security and prevent malicious file uploads. The vendor API will pull the existing JSON schema from the cloud in order to validate the file it is attempting to upload. If the file to be uploaded fails validation, the vendor will be notified of the exact parameters that may be missing or incorrect in the file they attempted to upload and validate.

5.2 API Gateway

The API gateway is responsible for managing and deploying APIs used throughout the cloud backend and APIs used in synchronization with frontend elements. The gateway provides developers and administrators with options to both create APIs and edit their features. Security protocol management and permission segmentation need to be provided in order to ensure secure data transmission and API access.

5.3 Message Handling

The message handling system in the back end is the primary means by which the cloud system communicates both internally and externally. This message handling component is responsible for encoding and decoding the messages used in the externally facing API gateway and for translating between the different messages used within the cloud system internally.

5.4 Computing Services

The functionality of the cloud backend places a large emphasis on a computing service that is intuitive to configure and sync with various other components of the environment. Computing services are responsible for executing the logic the APIs are designed in mind with and the code written for their functions. The selected computing service establishes a hub where data in the backend environment is sent for alteration, interpretation, and transportation to other components or locations. These services are managed by administrators and follow least-access principles by restricting communication between components and the service itself depending on user access.

5.5 Data Services

The cloud backend depends largely on a reliable and secure database provider in order to ensure smooth transfer of information between components. The database is responsible for securing information that it receives and distributes through practices that follow least-access principles. Data stored in the database also requires up to date encryption methods to be applied in order to ensure its safety at rest. Previous versions of data should be made available to database operators for use as reference or as backups. Ease of communication between the components and database is imperative to the backend system and requires monitoring and logging.

5.6 Monitoring and Logging

The monitoring and logging systems are responsible for the surveillance of all cloud components and their activities. These systems should be capable of storing access and action records. This entails noting instances where critical changes have been made to components, where access was attempted by unauthorized users, and general suspicious activity. Logging should be able to be automated and configured in a manner that allows for periodic storage into a larger database for future inspection.

5.7 Cryptographic Key Management

The cryptographic key management system is responsible for generating and distributing keys used to secure sensitive information across all cloud components. Privileges and access to key generation and distribution are detailed by these systems in order to provide secure encryption and decryption.

5.8 Authentication

The authentication system is responsible for segmenting user permissions based on groups to ensure that access to backend components is limited. This system is responsible for creating roles that allow actions and access to be distributed to user groups, which users will be placed into upon account creation. Users will then use the account they created in conjunction with this authentication system in order to gain authorized access to specific components and specific actions within those components.

5.9 Identities, Roles, and Privileges

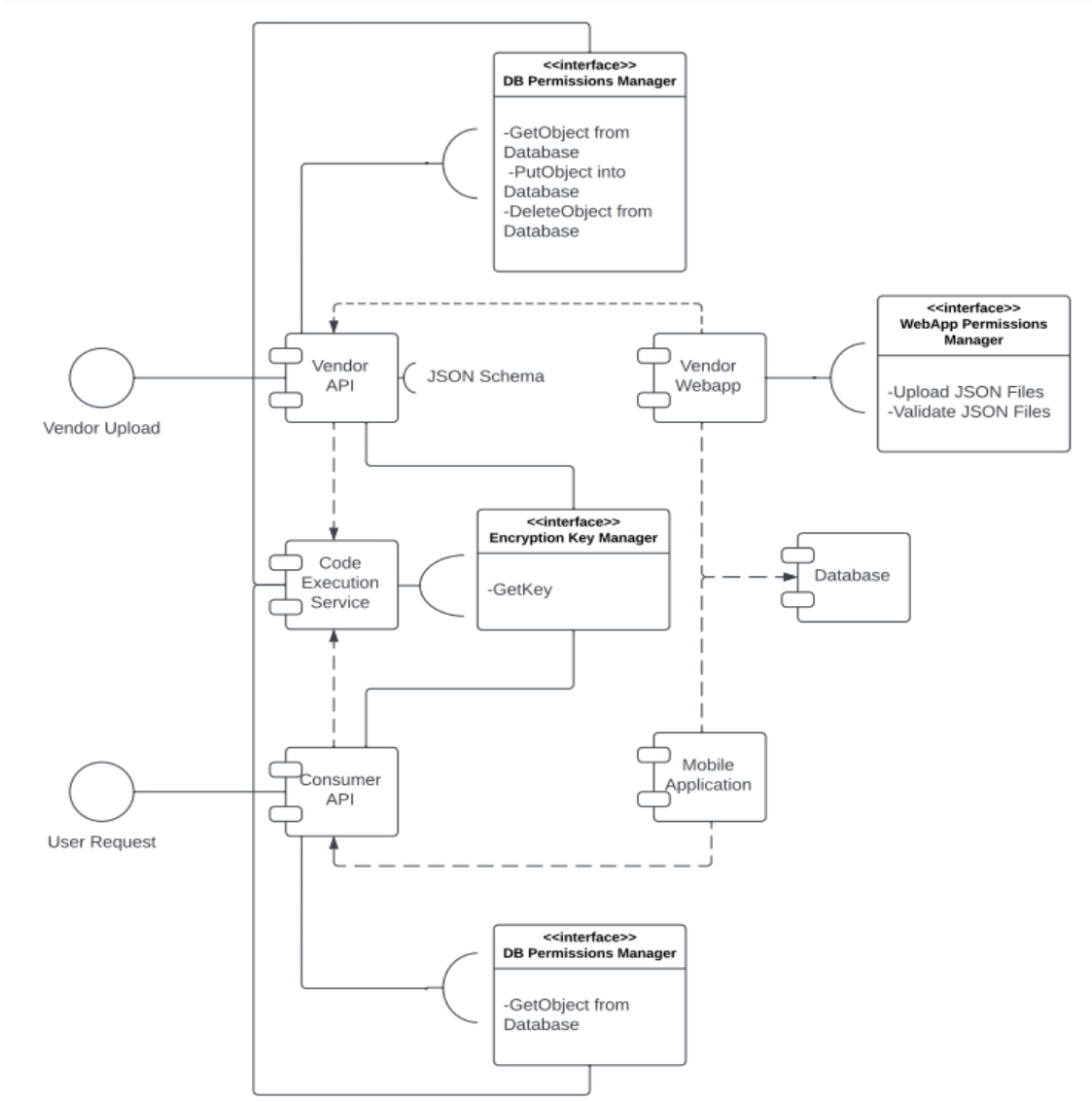
Identity and privilege management are key to backend security in the field device security configuration tool by providing fine-grained control over which users have access to which data. Access to data should be controlled on a per-user, per-vendor, per-device-category basis to enable user (field technician and vendor/manufacturer) to be able to access and/or update the minimal set of configuration items needed to complete their role.

5.10 Schema/API Validation

The schema and API validation systems make sure that only valid or authorized files are uploaded to the database and processed by other components in the backend. All files that are uploaded with the APIs must be validated against a provided schema, which is managed by an administrator. If the file fails validation, this system will remove it from the database and inform the user of the failed upload attempt and why it failed validation. The functionality and logic of this system is handled and executed by the computing service.

5.11 Proof-of-Concept Cloud Architecture

This UML component diagram (depicted in figure 12) serves as a general guideline for any vendors looking to implement the current cloud backend with a different cloud service. The diagram showcases the relationships between each component necessary for the backend to function. Interfaces detail the operations of these components, which helps vendors accurately replicate backend functionality in their implementation.

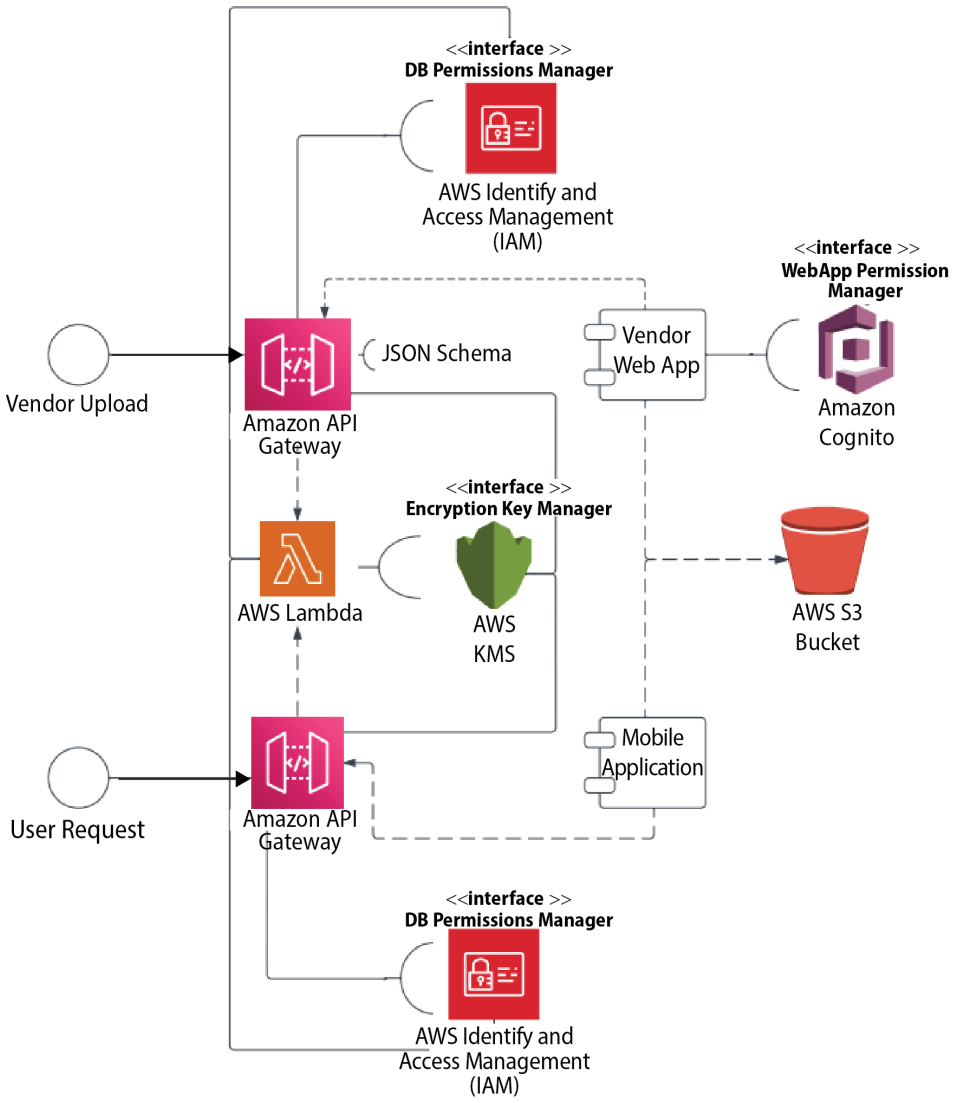


Source: FHWA.

Figure 12. Diagram. Cloud Architecture.

This component diagram (figure 13) provides vendors with examples of how various pieces of technology in a cloud service can act as components described in the generic model. This diagram reflects our current cloud structure which is implemented with AWS. Some examples of mappings to the generic model that this diagram demonstrates are:

- API gateway serves as both the consumer and vendor API components pictured in the previous model.
- AWS S3 serves as the database.
- AWS iAM serves as the DB Permissions Manager interface.



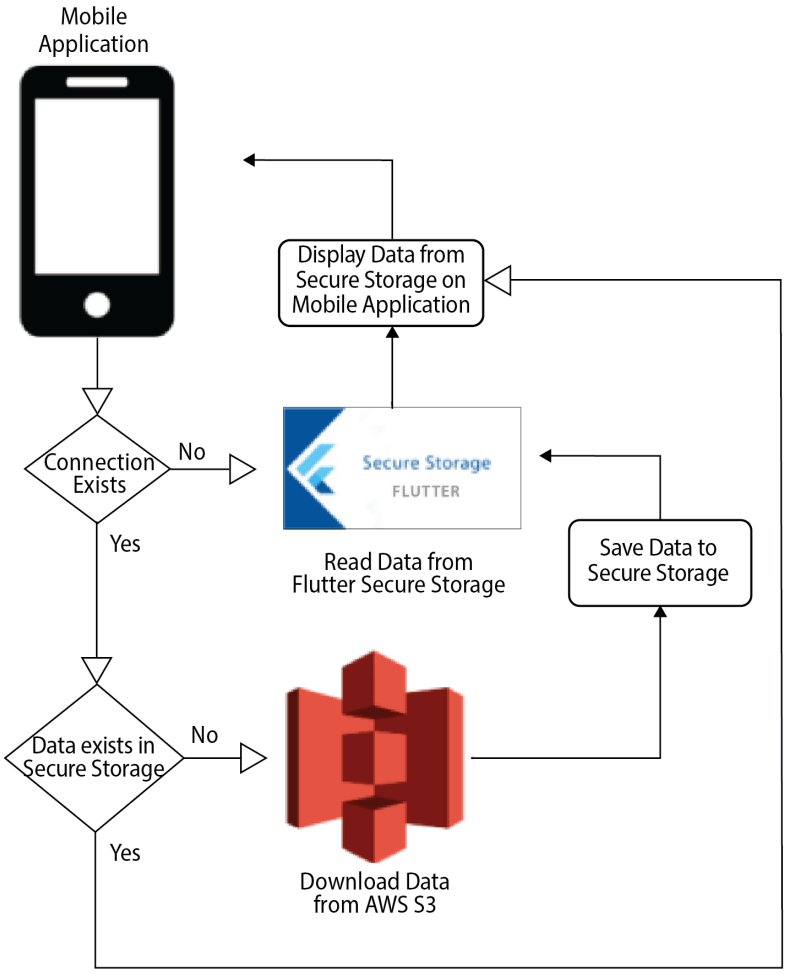
Source: FHWA.

Figure 13. Diagram. AWS Implementation.

The AWS cloud implementation is developed using these languages and frameworks.

- Python: Primary programming language used to develop scripts in AWS Lambda functions and handle API requests in the architecture.
- React: Primary programming language used to develop webapp functionality, features and interface for testing cross-device compatibility in the architecture.
- Dart: Primary programming language used to develop mobile application interface and features in Flutter architecture.

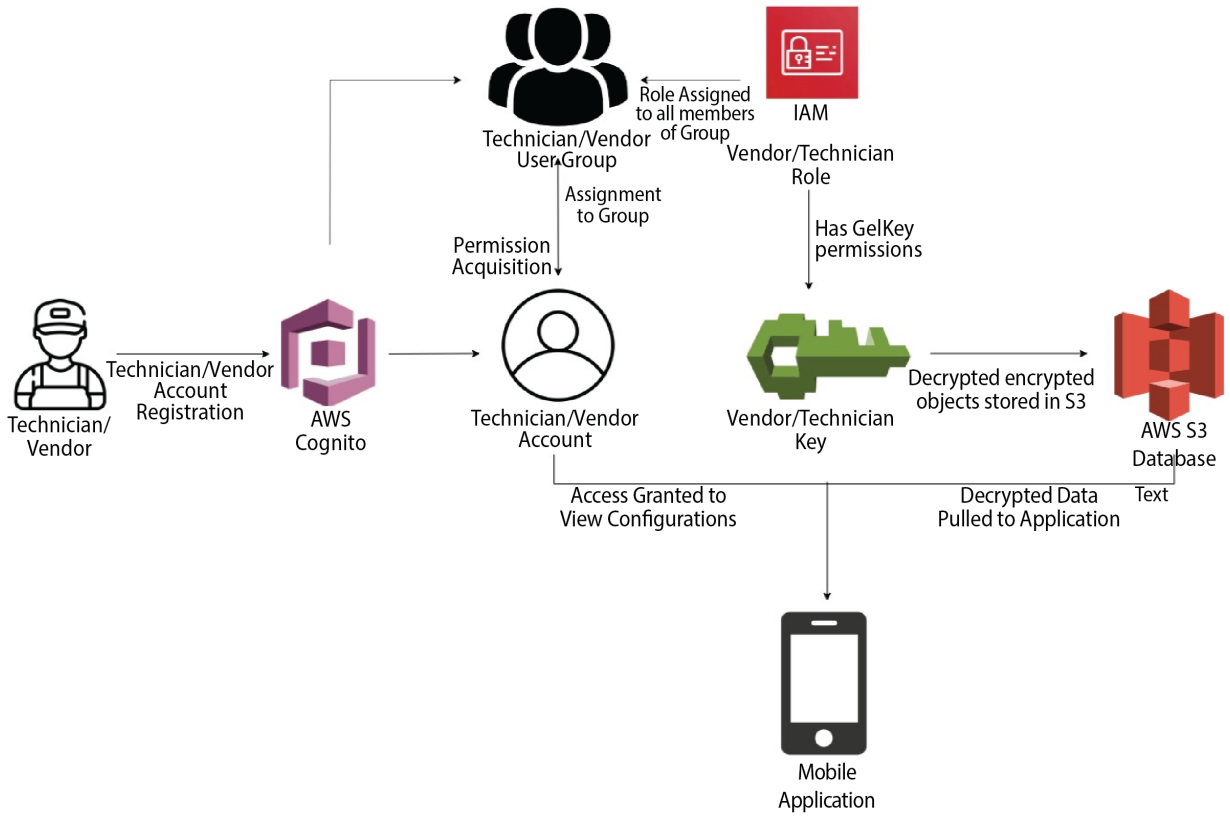
Figure 14 shows the relationship between technicians/vendors and the offline key management system that is employed to secure data that is stored within Flutter Secure Storage. The key components in this system are AWS S3, Flutter Secure Storage, and the mobile application itself. Once the user is authenticated through Cognito and logged into the mobile application, the app will check if a connection exists. If a connection is available to be connected to and there is no data in Flutter Secure Storage, the application will then download the configuration data from AWS S3 and save it to Secure Storage for offline usage. The data will then be pulled from Secure Storage and displayed to the user on the mobile application. If a connection is available and there is already data in Flutter Secure Storage, the application will use the existing data in Secure Storage and display it to the user on the mobile application. If a connection is not available to be connected to, the application will instead read the data directly from Flutter Secure Storage and pull it to the application to be displayed to the user.



Source: FHWA.

Figure 14. Diagram. Offline Encryption Support.

Figure 15 shows the relationship between technicians/vendors and the online AWS-based key management system that is employed to secure data distribution within the cloud architecture. The key components in this system are AWS Cognito, iAM, and the S3 Database. Initially, the technician/vendor registers an account under AWS Cognito, which is then assigned to the corresponding Cognito user group. All members of this user group are assigned a specific role, which is configured with explicit permissions relating to AWS KMS and AWS S3. One of the several permissions that this role grants its users is the ability to get and use encryption keys from a specified key. With this key, technician/vendor accounts can decrypt data pulled to the mobile application from S3 and view its contents.



Source: FHWA.

Figure 15. Diagram. Online Encryption Support.

5.11.1 AWS Lambda

AWS Lambda provides serverless, cost-efficient cloud computing in the form of Lambda functions, which are linked to AWS APIs that have been created to send and receive data. There are two Lambda functions in this system that handle API logic for technicians and vendors. The technician Lambda function, (which is to be used with the technician API) extracts configuration data from the cloud and sends it to the technician API to be displayed to users. The vendor Lambda function handles configuration uploads and updates and is initialized using the vendor API.

5.11.2 AWS API Gateway

AWS API Gateway acts as the deployment and management system for all REST APIs that are used within this system. API Gateway secures all API interactions with other components through AWS provided data encryption in transit to data endpoints. API Gateway also provides authentication methods in order to ensure that initialization can only be initiated by specified users with API keys.

5.11.3 AWS Amplify

AWS Amplify provides the system with simplified full stack web app deployment with little to no overhead for engineers. The provided webapp natively links with other AWS components allowing for effortless prototyping and testing of other features in the architecture. Amplify also provides built-in authentication methods with React and AWS Cognito, which can be used to make sure system permissions work properly across different user account types.

5.11.4 AWS Cognito

AWS Cognito bears the responsibility of segmenting system permissions through user groups and identity pools. These groups and pools authenticate and restrict webapp and mobile application access for its users. User groups follow least-privilege principles by designating roles with specified permissions to each group. Upon authentication with Cognito on either the webapp or mobile application, users are then assigned to the most appropriate group that permits actions only fit for their usage case. For example, an admin user group would have read and write permissions while a regular user group would only have read permissions.

5.11.5 AWS Authentication

AWS authentication provides the application with a streamlined, secure authentication flow. AWS Cognito allows the app to control which accounts can login to the app and restrict users on what files the user can see. With AWS authentication, the app can utilize prebuilt UI components and even social identity provider logins. These prebuilt components also include Account Creation and Forgotten Password features.

5.11.6 AWS SSE KMS

SSE KMS is a key management system that handles the encryption of the data that is transported and stored at several points in this architecture. Keys are generated by this service and assigned to roles in order to give them access to interact with encrypted data across the system. These keys can be used to decrypt secure data when appropriate for its user.

5.11.7 Firebase Database

Firebase provides a NoSQL database that syncs data across all clients. Data is stored as a JSON file which should be created or read by devices. The app should be able to take user feedback and transform it into a new JSON object to store in the Firebase database. From there, authorized users can pull these objects from the database for user feedback analysis.

5.11.8 Vendor API

The actual API logic and functionality is handled by the vendor APIs respective vendor Lambda function, which the API calls on initialization. When the vendor API is called, it allows vendors to upload or update configurations in the cloud. The vendor API will also validate the input file against a preexisting JSON schema in order to ensure that the upload is appropriate and fits standards. If the file does not pass

validation, it will be deleted from the cloud and the vendor will be informed with explicit details, including which parameters in the file caused validation errors.

5.11.9 Technician API

The actual API logic and functionality is handled by the technician APIs respective technician Lambda function, which the API calls on initialization. When the technician API is called, it allows the specified configuration data to first be pulled from the cloud, then sent to the mobile application, where it will be processed and displayed for technicians to view.

5.11.10 Flutter

Flutter takes input from the technician API in the form of configuration data which is stored in a JSON file pulled from the cloud. This framework is responsible for processing and reformatting the JSON file in order to output and display the configuration details on the mobile application interface for technicians.

5.11.11 Python JsonSchema

JsonSchema is a Python implementation of the original JSON Schema library, imported with Lambda layers. This implementation provides JSON file validation against an inputted schema and JSON file. After validating the JSON file, the validation either passes without issue or returns an error message. In this architecture the vendor Lambda function takes the outputted validation results in order to determine whether the inputted JSON file will be deleted or kept on the cloud.

Chapter 6. Cybersecurity Schema

The schema details the available policy groups and policy element that the developer for the vendor can specify for their ITS equipment. The vendor will identify the policy groups and elements applicable to the equipment and add recommendations for each supported element. Policy element recommendations must be uploaded to the policy service in the format described in this section. Constraints are also detailed herein, and should be validated by the policy service prior to allowing the upload from the vendor. Note that the developer for the vendor does not need to implement all of the following schema elements in order for the application to function properly, as the listed elements simply represent a superset of all possible security configuration items.

6.1 Device Information Group

The device information group fields specify additional information about the device the security configuration data pertains to. This information will not be used for configuration purposes but is required to display the device to the field technician on the mobile application user interface after the data file is downloaded.

6.1.1 Device Name

The device name field should contain the human-readable name of the device the configuration pertains to. This data will be used on the user interface to help field technicians select the correct device that matches what they are trying to configure.

- Options: Any string under 50 characters
- Recommendation: N/a
- Format: String
- Constraints: <50 characters

6.1.2 Device Category

The device category is a broad and vendor-definable way to group devices that might be used in a similar context in the field. For example, a vendor might define an intersection category that they could use to group all the field devices that see usage at an intersection or they might define a networking category that might group all the devices that play a role in network operations.

- Options: Any string under 50 characters
- Recommendation: N/a
- Format: String
- Constraints: <50 characters

6.1.3 Device Type

This field is used to specify the exact type of the device that the recommendations pertain to. This might have values such as camera or router, that describe what the device is at a high level.

- Options: Any string under 50 characters
- Recommendation: N/a
- Format: String
- Constraints: <50 characters

6.1.4 Device Subtype

This field is used to further refine the type of the device defined in the above device type field. If a device has type Camera, then subtypes might be Standard, Thermal, Depth-Sensing, Binocular, etc.

- Options: Any string under 50 characters
- Recommendation: N/a
- Format: String
- Constraints: <50 characters

6.1.5 Image URL

This field will be used by the mobile application to download an image to display alongside the device on the home page of the application. This image should clearly show the device in well-lit conditions with a resolution of at least 640x640px. The image must be in JPG format.

- Options: Any string under 200 characters
- Recommendation: N/a
- Format: String
- Constraints: String must be a valid URL pointing to an image in .JPG format

6.2 Password Policy Group

The password policy group includes policy elements that can be configured within the ITS equipment to require strong passwords, limit failed login attempts, enforce device timeouts, enable credential recovery features, set maximum password age, and enforce minimum time between password recovery attempts.

6.2.1 Minimum Password Length

Minimum password length specifies the minimum length that a password can be within ITS equipment registration. It is specified as a number representing minimum password length, passed as an integer.

- Options: Any integer over 4

- Recommendation: 12 characters
- Format: Integer (INT)
- Constraints: >4

6.2.2 Minimum Password Quality

Minimum password quality is an identifier representing the password quality categories, which can be passed as a string or integer depending on how the qualities are implemented.

- Options: [1] Unspecified, [2] String, [3] Numeric, [4] Numeric (complex), [5] Alphabetic, [6] Alphanumeric, [7] Complex
- Recommendation: [7] Complex
- Format: [1] String, [2] List of Strings
- Constraints: None

6.2.3 Device Lock Timeout

Device lock timeout specifies the maximum amount of time that may pass before a device is locked from the software automatically. This time period is set in minutes and represented by a number, passed in as a double.

- Options: Any double greater than 3
- Recommendation: 5 minutes
- Format: Double (double)
- Constraints: None

6.2.4 Maximum Number of Failed Attempts

Maximum number of failed attempts specifies the maximum number of times password entry may be failed by a user, passed in as an integer. Exceeding this number results in the user being barred from attempting to log in to that account and asked to reset their password.

- Options: Any number greater than 3
- Recommendation: 5 attempts
- Format: Integer (INT)
- Constraints: None

6.2.5 Enable Credential Recovery

Credential recovery provides technicians with the option to reset their passwords. This option is passed in as a Boolean which represents the state of the setting (True/False = On/Off).

- Options: True/False

- Recommendation: True
- Format: Boolean (Bool)
- Constraints: None

6.2.6 Maximum Days before Password Expiration

Password expiration mandates password changes in order to maintain consistent credential security across user accounts. The maximum days parameter specifies how many days a password can remain active until it must be changed and is passed in as an Integer.

- Options: Any number greater than 14
- Recommendation: 180 days
- Format: Integer (INT)
- Constraints: None

6.2.7 Minimum Time between Credential Recovery Attempts

Minimum time in this instance represents the minimum number of days that must pass before a password can be recovered again if it has been recently reset. This parameter is passed in as an Integer.

- Options: Any number greater than 2
- Recommendation: 5 days
- Format: Integer (INT)
- Constraints: >2

6.3 Application Restriction Policies

6.3.1 List of Approved Apps

- Options: [1] Alphanumeric, [2] OID
- Recommendation: None
- Format: [1] Alphanumeric String for Application name, [2] OID String for Object ID
- Constraints: None

6.3.2 List of Unapproved Apps

- Options: [1] Alphanumeric, [2] OID
- Recommendation: Discuss with customer for best approach.
- Format: [1] Alphanumeric String for Application name, [2] OID String for Object ID
- Constraints: None

6.4 Physical Security Policies

The physical security policy group includes policy elements that restrict and manage access to physical components interacting with ITS equipment.

6.4.1 Debugging Features

Manages diagnostics, error catching, and print statements that detail operating information for an ITS device, passed in as a Boolean to determine whether these features are enabled.

- Options: True/False
- Recommendation: False, disable debugging features
- Format: Boolean (Bool)
- Constraints: None

6.4.2 USB Enabling

Determines whether connecting to physical USB ports is enabled for ITS equipment. This is determined by a passed in Boolean value.

- Options: True/False
- Recommendation: False, disable USB
- Format: Boolean (Bool)
- Constraints: None

6.4.3 Location Services

Allows for real-time location of ITS equipment to be transmitted, enabled, or disabled by a passed in Boolean value.

- Options: True/False
- Recommendation: False, disable location services
- Format: Boolean (Bool)
- Constraints: None

6.5 User Management Policies

6.5.1 Inactive User Lock

Establishes a maximum number of days for a user account to stay inactive before being disabled to prevent future security vulnerabilities. Number of days is passed in as an Integer.

- Options: Any number greater than 20
- Recommendation: 30 days
- Format: Integer (INT)
- Constraints: Integer cannot be greater than 90

6.5.2 Require Two-Factor Authentication

Mandates two-factor authentication for all technicians logging in to ITS equipment, parameter passed in as Boolean value.

- Options: True/False
- Recommendation: True
- Format: Boolean
- Constraints: None

6.5.3 Enable Multiple Two-Factor Authentication Methods

Allows vendor to enable a variety of methods used to authenticate users. The enabling of these methods is passed in as a list of Booleans, with each index (i) of the list corresponding to an authentication method.

- Options: [1] T/F (Fingerprint), [2] T/F (SMS), [3] T/F (Retina) …. [i]T/F (Auth Method)
- Recommendation: [2] True
- Format: List of Booleans (Bool)
- Constraints: None

6.6 Audit Collection Policies

6.6.1 Account Recovery Logging

Logs attempts to recover account credentials as well as password changes.

- Options: True/False
- Recommendation: True
- Format: Boolean (Bool)
- Constraints: None

6.6.2 Database Interaction Logging

Logs user interactions with database.

- Options: True/False
- Recommendation: True
- Format: Boolean (Bool)
- Constraints: None

6.6.3 Log Storage Frequency

Determines how often, in days, logs will be uploaded to a secure database. Days parameter is passed in as an Integer.

- Options: Any number greater than 5
- Recommendation: 14 days
- Format: Integer (INT)
- Constraints: None

6.6.4 Log Storage Destination

Determines the location where logs will be uploaded for storage and future reference. Location parameter is passed in as a URL.

- Options: Any URL
- Recommendation: None
- Format: URL
- Constraints: URL must be properly entered to fit with standard formatting.

6.7 System Time Policies

6.7.1 Require Network Time

Requires network time to be set before proceeding with operating ITS equipment.

- Options: True/False
- Recommendation: True
- Format: Boolean (Bool)
- Constraints: None

6.7.2 Network Time Protocols List

If network time is required, produce a list of Network Time Protocol servers that are allowable.

- Options: True/False

- Recommendation: True
- Format: Boolean (Bool)
- Constraints: None

6.7.3 Allow Automatic Date and Time

Allow Date & Time to be automatically configured and readjusted according to time-zones and time changes, passed in as a Boolean.

- Options: True/False
- Recommendation: True
- Format: Boolean (Bool)
- Constraints: None

6.7.4 Utilize Three Synchronized Time Sources

Enables three synchronized time sources to maintain consistency in event log timestamps.

- Options: True/False
- Recommendation: True
- Format: Boolean (Bool)
- Constraints: None

6.8 Apache Tomcat Policies

6.8.1 Enable SSL

Implements data encryption and certificate-based authentication for client-server communications.

- Options: True/False
- Recommendation: True
- Format: Boolean (Bool)
- Constraints: None

6.8.2 Enable Detailed Logging

Logs detailed information such as event source, date of event, users involved in event, event timestamp, source, and destination address in communication, and much more.

- Options: True/False
- Recommendation: True

- Format: Boolean (Bool)
- Constraints: None

6.9 System File Restriction Policies

6.9.1 Enable File Versioning

Enable storage of older versions of ITS device manuals or data for future reference.

- Options: True/False
- Recommendation: True
- Format: Boolean (Bool)
- Constraints: None

6.9.2 Enable Integrity Protection

Identifies and reports changes to ITS device system files.

- Options: True/False
- Recommendation: True
- Format: Boolean (Bool)
- Constraints: None

6.9.3 Enable File Encryption

Enable secure encryption of files on ITS devices.

- Options: True/False
- Recommendation: True
- Format: Boolean (Bool)
- Constraints: None

6.9.4 Generate Access Restrictions

Generate and apply restrictions to ITS device users to limit read, write, and execute access to device files. Permissions are segmented through creation and management of user roles such as admin, auditor, and user. The first parameter determines which role the user wishes to manage, while the second parameter details the permissions that will be assigned to that role. Both parameters are passed in as Strings.

- Options: [1] admin, auditor, user; [2] Any Permissions

- Recommendation: Whitelist roles with restrictions of limit, read, write, and execute
- Format: [1] String, [2] String
- Constraints: None

6.10 Secure Boot Load Policies

6.10.1 Read/Write Only Permissions

Only allow sudo users to read/write boot parameters to prevent vulnerability exploitation by non-root users.

- Options: True/False
- Recommendation: True, set only root users to read/write boot parameters
- Format: Boolean (Bool)
- Constraints: None

6.10.2 Require Bootloader Password

Prevents unauthorized users from entering boot parameters and weakening security.

- Options: True/False
- Recommendation: True, prevent unauthorized users from changing boot parameters.
- Format: Boolean (Bool)
- Constraints: None

6.10.3 Require Authentication for Single-User Mode

Require authentication when booting into single-user mode to prevent unauthorized users from gaining root access through repeated rebooting.

- Options: True/False
- Recommendation: True
- Format: Boolean (Bool)
- Constraints: None

6.11 Encrypted Storage Policies

6.11.1 Automounting

Allows any USB drive or disc to be attached to the system and have its contents transferred into the ITS device. Option to enable this is passed through as a Boolean value.

- Options: True/False
- Recommendation: False, do not enable Automounting
- Format: Boolean (Bool)
- Constraints: None

6.11.2 Routine Cloud-Based Backups

Periodically backup hard drive contents to cloud-based storage system in case of system failure. The period between each routine backup is passed through as an Integer representing the number of days between backups. The destination endpoint is passed through as a String representing the destination URL.

- Options: Any integer greater than 5 | URL
- Recommendation: 30 days | None
- Format: Integer (INT) | String
- Constraints: None

6.11.3 Routine Defragmentation

Periodically defragment hard drive, in order to reorganize files and improve performance. The period between each routine defragmentation is passed through as an Integer representing the number of days between defragmentation.

- Options: Any integer greater than 5
- Recommendation: 30 days
- Format: Integer (INT)
- Constraints: None

6.11.4 Limit Hard Drive Storage Capacity

Limit maximum storage capacity of hard drive to a specified percentage of actual capacity to minimize risk of data corruption and performance loss. Percentage is specified through input passed as double.

- Options: Any double greater than 0
- Recommendation: 0.80 size of actual capacity

- Format: Double
- Constraints: Double must be less than 1

6.11.5 Generate Encryption Recovery Tokens

Generate fallback tokens in the case that encryption key credentials are lost. Note that vendor administration will be managing the encryption recovery token system and will be able to set up permissions at their discretion.

- Options: True/False
- Recommendation: True
- Format: Boolean (Bool)
- Constraints: None

6.12 Data at Rest Policies

6.12.1 Enable Data at Rest Encryption

Encrypts sensitive data at rest on servers, applications, and databases.

- Options: True/False
- Recommendation: True
- Format: Boolean (Bool)
- Constraints: None

6.12.2 Enable Data Loss Protection Solutions

Enable DLP in the cloud with tools such as Google Cloud DLP, Barracuda backup, etc., in order to further secure data at rest and minimize data leaks.

- Options: True/False
- Recommendation: True
- Format: Boolean (Bool)
- Constraints: None

6.12.3 Data Classification

Classify data at rest by tag in order to apply appropriate security and defense measures in case of exposure. Tags are passed in as String(s) and will be used to categorize data.

- Options: Tag(s)

- Recommendation: None
- Format: List (String (Str)), List of strings
- Constraints: None

6.12.4 Routine Data Backup

Data at rest will be backed up routinely in the case of mishandled data transfers or power outages. How frequently data will be backed up is determined by input passed in as an Integer (INT) representing days.

- Options: Any number greater than 5
- Recommendation: 14 days
- Format: Integer (INT)
- Constraints: None

6.12.5 Data Tokenization

Substitutes sensitive data with non-sensitive tokens that will not be able to be exploited if data is stolen or leaked.

- Options: True/False
- Recommendation: True
- Format: Boolean (Bool)
- Constraints: None

6.13 Audit Management Policies

6.13.1 Data Retention Capacity

Sets the max size (in bytes) of data to be held by audit logs, size and age are passed in as a double.

- Options: Any number greater than 1
- Recommendation: None (Dependent on preference)
- Format: Double
- Constraints: None

6.13.2 Verify auditid

Verify that auditd (daemon that records system events, provides information regarding unauthorized access instances) is installed.

- Options: True

- Recommendation: True, scans and checks if auditid is provided.
- Format: Boolean (Bool)
- Constraints: Linux operating systems

6.13.3 Automatic Audit Log Deletion by Capacity

Delete audit logs if they have reached maximum file size capacity.

- Options: [1] [True, *size of capacity*], [2] False
- Recommendation: Set to True and delete when file size reaches a certain capacity.
- Format: [Boolean, Double], [Boolean]
- Constraints: Capacity must be below 100 GB, Linux operating systems

6.13.4 Automatic Audit Log Deletion by Age

Delete audit logs if they have reached a maximum age

- Options: [1] [True, *age*], [2] False
- Recommendation: Set to True and delete when file size reaches a certain age
- Format: [Boolean, Double], [Boolean]

6.14 Integrity Protection Policies

6.14.1 Enable AIDE

AIDE takes a snapshot of the current filesystem's state noting register hashes, modification times, and other data defined by administrators. A database is then built based on this snapshot and is stored to use in future integrity checks against the system's status. AIDE will automatically report any discrepancies between versions found during integrity checks to administrators.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: Linux operating systems

6.14.2 Routine Integrity Checking

AIDE will be activated routinely in order to determine if critical file changes have occurred. The frequency of AIDE's snapshots will be determined by input passed as an Integer, representing days between each activation.

- Options: Any number greater than 3

- Recommendation: 7 days
- Format: Integer (INT)
- Constraints: Unix-like operating systems

6.14.3 Enable Version Control

Allows for data recovery or rollback in case of breaches or data loss incidents.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: Version control software (i.e., Git)

6.15 Privilege Management Policies

6.15.1 Generate Sudo Log File

Simplifies auditing of sudo commands.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: Linux operating systems

6.15.2 Reauthentication for Privilege Escalation

Users will be required to reauthenticate to access higher-privilege resources or tasks. Prevent automated processes from being able to utilize elevated privileges. This option is enabled/disabled by an argument that is passed in as a Boolean.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: Linux operating systems

6.15.3 Sudo Authentication Timeout

Establishes a timeout that reduces the window of opportunity for unauthorized privileged access by unauthorized users. Maximum time before timeout is represented in minutes and passed in as a double.

- Options: Double value

- Recommendation: 15 minutes
- Format: Double
- Constraints: Linux operating systems

6.16 Software Update Policies

6.16.1 Automate Software Patch Management

Deploy automated software update tools in order to ensure that third-party software on all systems is running the most recent security updates.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: None

6.16.2 Automate Operating System Patch Management

Ensure that operating systems are running the most recent security updates.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: operating system

6.16.3 Disable Mandated System Updates

Prevents system updates from being forcefully executed, allow for evaluation of updates and patches before executing. Input responsible for enabling this feature is passed in as a Boolean.

- Options: True/False
- Recommendation: True
- Format: Boolean (Bool)
- Constraints: None

6.17 Firewall Settings Policies

6.17.1 Allow Firewall Management

Manage a host-based firewall or port-filtering tool on end-user devices, with a default-deny rule that drops all traffic except those services and ports that are explicitly allowed.

- Options: [1] [True, ports], [2] False
- Recommendation: False
- Format: [1] Tuple with a Boolean True and “ports” as a set. [2] Boolean
- Constraints: [1] Must be a set of valid ports

6.17.2 Apply Host-Based Firewalls/Port Filtering

Apply host-based firewalls or port-filtering tools on end systems with a default-deny rule that drops all traffic except those services and ports that are explicitly allowed.

- Options: True, False
- Recommendation: None
- Format: Boolean
- Constraints: None

6.17.3 Apply Firewall Rules on New Port Instances

Apply default firewall rules on new port connections, ensuring that data traffic is consistently regulated as connections are established.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: None

6.18 Data Execution Prevention Policies

6.18.1 Enable Anti-exploitation Features

Enable anti-exploitation features where possible, such as Microsoft Data Execution Prevention, Windows Defender Exploit Guard, or Apple System Integrity Protection and Gatekeeper.

- Options: True, False
- Recommendation: True

- Format: Boolean
- Constraints: None

6.18.2 Whitelist

Only allows trusted code to run on the ITS device and blocks any other programs that attempt to run. Identification tags for code segments that would be permitted to be run are passed in as String(s).

- Options: Any string
- Recommendation: None
- Format: String (Str)
- Constraints: None

6.18.3 Enable Intrusion Detection System

Monitors system state and reports unauthorized changes to data.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: None

6.19 Address Space Layout Randomization (ASLR) Policies

6.19.1 Enable ASLR

Obfuscates virtual memory regions such as the stack, heap, library, and executable positions by rearranging address spaces of the environment. Helps prevent the effectiveness of buffer overflow attacks as the location of executables will be constantly shifting. Applicable to Windows, Linux, and MacOS systems.

- Options: True/False
- Recommendation: True
- Format: Boolean (Bool)
- Constraints: None

6.19.2 SAF Authorization

When ASLR is enabled, SAF authorization can be used to exempt selected address spaces from ASLR selection.

- Options: True/False

- Recommendation: True
- Format: Boolean (Bool)
- Constraints: IBM z/OS Systems

6.19.3 Configure VMA Space

Virtual Memory Area (VMA) space is where randomization occurs, generally dedicating a larger space allows for greater obfuscation. The size of this space is determined by input passed in as a double.

- Options: Any double greater than 1
- Recommendation: None
- Format: Double
- Constraints: Linux operating systems

6.20 Secure Key Management Policies

6.20.1 Routine Key Rotation

An over-used key can encrypt too much data which makes it vulnerable to cracking, especially using old symmetric algorithms. High volumes of data can be exposed in the event of a key becoming compromised. This can be countered with frequent key rotations, with the frequency being passed in through input as an Integer representing the number of days between rotations.

- Options: INT value
- Recommendation: 30 days
- Format: Int
- Constraints: None

6.20.2 Audit Key Logging

Key lifecycle is logged to identify instances of key compromise.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: None

6.20.3 Multi-Factor Authentication Key Access

Require multi-factor authentication to access keys. Important for keys that have high privilege such as admin.

- Options: True, False
- Recommendation: Boolean
- Format: True
- Constraints: None

6.20.4 Configure Key Size

Larger keys offer longer protection at the cost of performance. Configuration is based on input passed in as an integer which determines the bit size of the key.

- Options: Any integer greater than 1
- Recommendation: 2048 bits for Rivest Shamir Adleman and P-256 for Elliptic Curve Digital Signature Algorithm
- Format: Integer
- Constraints: None

6.21 Remote Configuration Policies

6.21.1 Enable Remote Access on a Per-Group Basis

Permit remote access to predetermined groups of users.

- Options: Select a group of users
- Recommendation: None
- Format: String[]
- Constraints: Strings within the list should be user group names

6.21.2 Enable Remote Access on a Per-User Basis

Provide permission to remote access based on authorized users.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: None

6.21.3 Authorize Remote Access Based on Connection Type

Authorize users based on the connection type provided.

- Options: Wi-Fi, Ethernet,Cellular

- Recommendation: None
- Format: Selection
- Constraints: None

6.21.4 Authorize Remote Access Based on Time of Day

Authorize remote access for users based on the time of day. Ex: Certain users will only have remote access during an organization's business/working hours.

- Options: Time
- Recommendation: None
- Format: Standard Time UTC
- Constraints: Time cannot be set to below 1 or above 24

6.21.5 Authorize and Verify Caller ID

Specify the telephone number that the user must be calling from in order to establish a successful connection. Requires hardware to detect the number that the user is calling from.

- Options: Caller ID
- Recommendation: None
- Format: Country code + area code + number, number-only, name + number
- Constraints: None

6.21.6 Callback Options

Enables users to connect remotely and without use of callback.

- Options: True, False
- Recommendation: False
- Format: Boolean
- Constraints: None

6.21.7 Assign Static IP Address

Assign a unique IP address to each user that connects to an ITS remotely.

- Options: IP address
- Recommendation: Whitelist trusted IP addresses
- Format: 128-bit number
- Constraints: None

6.21.8 Assign Duration of Session

Disconnect users after a specified amount of idle time has passed. Maximum idle time is measured in minutes and determined by a parameter which is passed in as an integer.

- Options: Any integer greater than 1
- Recommendation: Less than 45 minutes
- Format: Integer
- Constraints: Less than 60 minutes

6.21.9 Assign Maximum Session

Disconnect users after an allotted time. Maximum allotted time is measured in minutes and determined by a parameter that is passed in as an integer.

- Options: Any integer greater than 1
- Recommendation: 60 minutes
- Format: Integer
- Constraints: Less than 1440 minutes

6.21.10 Configure Encryption Parameters

Permit or define VPN connections to the remote server.

- Options: True, False
- Recommendation: False
- Format: Boolean
- Constraints: None

6.22 Data In Transit Policies

6.22.1 Enable End-to-End Encryption

Encrypt data before sending to destinations endpoints and only decrypt data using a public-private key pair.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: None

6.22.2 Enable Authentication at Endpoints

Access to data endpoints should be configured to require authentication in order to further secure data transportation.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: None

6.22.3 Enable Secure Protocols

Provide a secure protocol to perform encrypted communications. SSL, TLS, HTTPs. HTTPs are the most appropriate protocol to implement for this usage case, but other options can be selected based on administrator discretion. Arguments are passed in as a list of True/False Booleans, with each index evaluating to one of the above encrypted communication forms. List[0] would evaluate to SSL, List[1] to TLS.... etc.

- Options: List of acceptable encrypted communication protocols
- Recommendation: [“SSL”, “TLS”, “HTTPS”]
- Format: String[]
- Constraints: Entries in the list should be encryption protocols supported by the system

6.22.4 Enable DLP Solutions (Data Loss Prevention)

Enable DLP services in order to strengthen data security in transit and minimize leakage.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: None

6.23 SELinux Policies

6.23.1 Enable Permissive Mode

When permissive mode is enabled, SELinux policy is not enforced. System remains operational and SELinux does not deny any operations but only logs AVC messages, which can be used for troubleshooting, debugging and SELinux policy improvements.

- Options: True, False
- Recommendation: False

- Format: Boolean
- Constraints: Linux operating systems

6.23.2 Enable Enforcing Mode

When enforcing mode is enabled, SELinux policy is enforced and denies access based on SELinux policy rules.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: Linux operating systems

6.23.3 Enable SELinux on Systems That Previously Had it Disabled

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: Linux operating systems

6.23.4 Enable and Authorize SELinux Roles

Provide special roles for users with elevated permissions or deny access

- Options: Select user(s)
- Recommendation: Authorize trusted users
- Format: User(s)
- Constraints: Linux operating systems

6.23.5 Disable SELinux

Disable SELinux, policy is not loaded and not enforced and AVC messages are not logged.

- Options: True, False
- Recommendation: False
- Format: Boolean
- Constraints: Linux operating systems

6.24 Firmware Update Settings Policies

6.24.1 Enable Secure Boot

Ensures only trusted firmware can be loaded and ran on the device. Checks the digital signature of the firmware before it is allowed to run.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: Devices (i.e., Windows machines, Linux machines, Android phones, Apple phones)

6.24.2 Support Rollback Updates

Allow for recovery of firmware when problems are encountered.

- Options: True, False
- Recommendation: False
- Format: Boolean
- Constraints: None

6.24.3 Require Key to Manage Firmware Update

When evaluating new hardware and vendors there is a key requirement to permit changes to the firmware.

- Options: [1] [True, Key], [1] False
- Recommendation: True
- Format: [Boolean, Key] pair or Boolean
- Constraints: Key must be reasonable

6.24.4 Update Firmware Automatically

Updates the firmware to ensure device is always running latest firmware.

- Options: True, False
- Recommendation: False
- Format: Boolean
- Constraints: None

6.24.5 Update Firmware Manually

Manually update firmware instead of automatically. Helps to test and validate updates before reaching production.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: None

6.25 Integrity Measurement Architecture (IMA) Policies

6.25.1 Enable IMA Appraisal

Establish local integrity validation for file hash values before files are read or executed.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: Linux operating systems

6.25.2 Enable IMA Measurement

Log IMA activity and hash file history.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: Linux operating systems

6.25.3 Generate IMA Keyring

Generate keys that provide permissions to alter/enable IMA appraisal functionality.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: Linux operating systems

6.25.4 Enable IMA Audit

Includes file hashes in audit logs, which provide system security analytics and metrics.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: Linux operating systems

6.26 Privacy Policies

6.26.1 Enact Fair Information Practice Principles

Agencies cannot disclose records without individual consent or request.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: None

6.26.2 Third-Party Transparency

Disclose to end-user what personal information they have provided may be shared.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: None

6.26.3 Disable/Enable Public Networks

Connecting to local public networks provides another point of vulnerability for network attacks. This can be easily mitigated with manual connection to a private, company-managed network.

- Options: True, False
- Recommendation: False
- Format: Boolean
- Constraints: None

6.26.4 Configure Pseudonymization for Data at Rest

Replace components of data that would allow for personal identification with randomly generated artificial placeholders to retain user privacy.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: None

6.27 Server Port Settings Policies

6.27.1 Disable FTP Ports

FTP ports are outdated and are therefore especially vulnerable to attacks.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: None

6.27.2 Disable Telnet Ports

Telnet ports are outdated and are therefore especially vulnerable to attacks.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: None

6.27.3 Routine Port Scanning

Enable routine port scanning through tools such as Netstat and Network Mapper every 14 days to ensure that ports are secured and closed.

- Options: INT
- Recommendation: 14 days
- Format: INT value in days
- Constraints: None

6.27.4 Log port configuration changes

Log and monitor port configuration changes to keep track of possible vulnerabilities.

- Options: True, False
- Recommendation: True
- Format: Boolean
- Constraints: None

6.28 Error Handling

Errors that occur when validating JSON files are caught using *jsonschema's* (which is the same library used for validation) built-in error handling. This library informs vendors of the nature of the errors and the location of their occurrences by sending detailed, appropriate messages. Table 4 shows the error handling schema.

Table 4. Schema Error Handling.

Error	Details	Handler Message
NoSuchKey	Catches and handles all errors where the JSON file does not exist in the database	"The object to be validated does not exist in this database."
Failed Validating 'Required'	Catches and handles all errors where the JSON file is missing a property required by the schema in its contents	"Component is a required property." Failed validating 'Component' in schema [item][item_subclass][item_number]"

Chapter 7. Verification and Validation Plan

This chapter is intended to document the plan for Verification and Validation testing approach of the prototype Cybersecurity for ITS system as initially developed. This test plan (along with documentation of the test results) will provide future users and adopters of the Cybersecurity for ITS system with an understanding of the final state of the developed prototype. With the state of the developed prototype in mind, it is anticipated that future users and adopters of the system would assess the results of the protype testing against their needs and deployment plans and determine their own development and deployment roadmap.

In addition to the verification of the implementation of individual features laid out in this document this test plan will also validate that the application satisfies the stakeholder needs laid out in the project vision document. Each test case in the test plan will list the features it verifies and the needs validated by the test.

7.1 Testing Process

The standard engineering process involves several levels of testing, which help to verify correctness of the system. The first level is automated unit testing individual modules, often with mocked up interfaces to enable testing in isolation. Unit tests enable rapid testing of specific units of functionality within the application.

The next level of tests is component integration testing. This typically involves all the software for a given system (mobile application, server backend, server frontend) being executed together, either in the simulation environment or on a live device. It exercises the real-time interactions among the various software components with all the real internal interfaces in place. Sometimes, component integration testing is applied to a subset of the full system to better study a few key interactions. It also makes it easier to isolate issues earlier in the process, which makes future full scenario integration testing go more smoothly.

Once integration testing proves to be successful, and a complete set of functionality is ready to be bundled into a release candidate, all of that code is branched into a candidate release branch in GitHub to isolate it from ongoing feature development. This code is then packaged as candidate distributable application packages. At this point, the team performs a formal system-level verification test, which may involve multiple systems interacting (mobile application, server backend, server frontend). Only independently controlled (by configuration management processes and staff) software builds are used for this level of testing. Testing follows a formal test plan, and all results are documented. Issues discovered in testing are logged as Github tickets and tracked by testing team and project management for triage. Issues will be assigned a severity based on their impact to software functionality and prioritized by the project team. Verification is not considered complete until all essential anomalies are resolved and retested. After all essential issues have been resolved, as agreed upon by project team, any remaining minor issues not prioritized to be addressed will be documented in the release notes as known issues

with descriptions of their scope and impact. Upon acceptance, the release candidate becomes a formal release with a numbered version ID.

7.2 Features to be Verified

The table below enumerates the list of features described in the earlier sections of this document. "Feature IDs" are assigned to each for ease of traceability between test cases and features to ensure that verification covers all designed features.

Table 5 Features to be Tested

Feature ID	System	Feature
FE-1	Frontend	Register User Account
FE-2	Frontend	Multi-Factor Authentication
FE-3	Frontend	Login
FE-4	Frontend	Forgot Password
FE-5	Frontend	Scan ITS Equipment QR Code
FE-6	Frontend	Mode Selection
FE-7	Frontend	Wizard Walk Through of Security Configurations
FE-8	Frontend	Expert Mode Configuration
FE-9	Frontend	Search
FE-10	Frontend	Device Selection
FE-11	Frontend	Vendor/Model Selection
FE-12	Frontend	Report an Issue
FE-13	Frontend	Voice Recognition
FE-14	Frontend	Get Configuration
FE-15	Frontend	Logout
FE-16	Frontend	Screen Timeout
FE-17	Frontend	Encrypted Data in Transit
FE-18	Frontend	Encrypted Data at Rest
FE-19	Frontend	Re-verification for security configuration
BE-1	Backend	Register Organization Account
BE-2	Backend	Manage User Account
BE-3	Backend	Manage Organization Account
BE-4	Backend	Manage User Account Permissions
BE-5	Backend	Maintain Schema
BE-6	Backend	Filter Submissions (schema alignment)
BE-7	Backend	Password Configuration Management
BE-8	Backend	Login
BE-9	Backend	Forgot Password
BE-10	Backend	Logout

Feature ID	System	Feature
BE-11	Backend	Screen Timeout
BE-12	Backend	Encrypted Data in Transit
BE-13	Backend	Re-verification for security configuration

7.3 Stakeholder Needs to be Validated

In the vision document for the Cybersecurity for ITS system several stakeholder needs were originally identified for the two major stakeholder groups: ITS Equipment Developers and ITS Equipment Technicians. The needs identified for these user groups were as follows:

Vendors:

Modular: The application design should incorporate modular components to allow for re-use.

Secure: The confidentiality, integrity, and availability of vendor security configuration recommendations should be enforced throughout application design and operation in order to preserve the Intellectual Property (IP) of vendors.

Cost-Effective: The application design should integrate free and open-source libraries whenever possible.

Portable: The application design needs to work on both Android and Apple mobile devices. A single codebase should be required, to ensure that vendors can efficiently develop and deploy the application across their customer community.

Standardized: The application design should allow vendors to report device security configuration setting recommendations in a standardized manner for all of the vendor's devices.

IOO Equipment Technicians

Usable: The application needs to be simple to use even during conditions such as nighttime and cold weather.

Customizable: The application should allow users to configure personalized settings that persist across application closures and restarts.

Intuitive: The application should provide options for locating the device type being configured, using either a QR code scanner, a search function, or a browse function.

Comprehensive: Technicians should have a choice as to whether they would like an expert mode that allows the user to navigate to specific configuration settings, or a wizard mode that walks the technician through the entire set of security configuration settings.

7.4 Test Cases

Below are high-level descriptions for each of the test cases that cover the features described in the table above. The test cases list their prerequisites to indicate the flow of testing; the tests assume the application to be in the state resulting from the execution of their prerequisite test cases, e.g. if a test involves entering data into a database then tests dependent on that may assume said data to be present as a pre-condition. Practically, these test cases may be executed in batches (assuming all prerequisites are satisfied) or individually, this is left to the discretion of the lead test engineer.

Test Case 1 Account Registration and Provisioning

The test engineer will register two accounts, one as a Field Technician user and another as a Vendor user via the AWS backend for the application. The test engineer will use the vendor account to upload a test device configuration data set defined in the test plan. The test engineer, acting as a Vendor user, will configure proper access permissions for the registered field technician user account to access the newly uploaded data set. The test engineer, acting in the capacity of a system administrator user will access the AWS Cognito backend to confirm that the accounts are all properly initialized and the permissions defined earlier in the test are properly stored in the database.

Verified Features: FE-1, BE-1, BE-2, BE-3, BE-4, BE-5, BE-6, BE-8
Validated Stakeholder Needs: Secure

Test Case 2 Field Technician Data Download

Prerequisites: Test Case 1

The test engineer, acting as a Field technician user, will log in to mobile application with the account and credentials defined in Test Case 1. The mobile application automatically downloads all device configuration data for which the user is authorized (as configured in Test Case 1). The test engineer will confirm that data is encrypted in transit and in storage locally. They will also confirm that user account has access to devices they are authorized for and does not have access to devices they are not authorized for.

Verified Features: FE-3, FE-14, FE-17, FE-18, BE-12
Validated Stakeholder Needs: Standardized, Secure

Test Case 3 Field Technician Configuration Display

Prerequisites: Test Case 1, Test Case 2

The test engineer, acting as a Field technician user, will open the application and log in as specified in Test Case 1 and Test Case 2. The test engineer will confirm that the downloaded sample device data is rendered as an option on main page of application. The test engineer will confirm that the main page

user interface matches the description outlined in the System Requirements and Specification document. The test engineer should then search for a sample device and open its configuration recommendation. The test engineer should then use both the Field Technician application and Vendor user backend together will confirm that the displayed recommendations match those configured on the server.

Verified Features: FE-9, FE-10, FE-11
Validated Features: Usable, Intuitive

Test Case 4 Issue Reporting

Prerequisites: Test Case 1, Test Case 2

The test engineer, acting as a Field Technician, will log in (as defined in Test Case 1 and Test Case 2) and navigate from the main screen to the issue reporting screen. The test engineer will report an issue using test-plan defined values for the issue type selection and text report fields. The test engineer will then log-in to the backend system using a System Administrator user account to confirm that issue report is visible on backend dashboard and the data matches report.

Verified Features: FE-12

Test Case 5 Field User Logout

Prerequisites: Test Case1

The test engineer, acting as a Field Technician, will log in (as defined in Test Case1 and Test Case2) and then close the app. The test engineer will then leave the device idle for a fixed amount of time defined by the test plan. The test engineer should verify that their log in session has expired after the time has elapsed. The test engineer user should also verify that they are no longer able to access data after logout, and that timeout occurs at specified interval. The session expiration interval may be configured to be shorter for the purpose of this test as normal 30 days interval makes testing of timeout impractical.

Verified Features: FE-15, FE-16, FE-19, FE-4, FE-2, FE-3
Validated Stakeholder Needs: Secure

Test Case 6 Vendor User Logout

Prerequisites: Test Case1

The test engineer, acting as a Vendor user, will log in (as defined in Test Case 1) to the backend web-service and then close backend webpage. The test engineer will then leave their device idle for a fixed amount of time defined by the test plan. The test engineer should verify that their log in session has expired after the time has elapsed. The test engineer should also verify that they are no longer able to access data after logout, and that timeout occurs at specified interval. The session expiration interval may be configured to be shorter for the purpose of this test as normal 30 days interval makes testing of timeout impractical.

Verified Features: FE-15, FE-16, FE-19, FE-4, FE-2, FE-3, BE-9, BE-10, BE-11, BE-13
Validated Stakeholder Needs: Secure

Test Case 7 Expert Mode

Prerequisites: Test Case 1, Test Case 2

The test engineer, acting a as a Field Technician user, navigates through the main screen of the application to select a device. The test engineer confirms the default state of the application is to present a wizard/walkthrough overview of the non-expert mode configuration options for the device. The test engineer then backs out to the setting page of the application and enables expert mode. Navigating again to the same device, the user confirms that all options are now displayed in a fully-browsable set consistent with the expert mode description in the System Requirements document.

Verified Features: FE-6, FE-7, FE-8
Validated Stakeholder Needs: Intuitive, Comprehensive, Customizable

Test Case 8 Mobile App Advanced Features

Prerequisites: Test Case 1, Test Case 2

The test engineer, acting as a Field Technician user, will test voice input for text input to search system. Voice input may be handled by OS voice recognition functionality. The test engineer user should confirm that all text-input fields in the application support voice input and that voice input functions as expected per mobile operating system implementation. Additionally, the test engineer should test the QR code device lookup feature. The test engineer should scan a QR code with an embedded link to a set of device configuration recommendations. The test engineer should confirm that the mobile application properly decodes the QR code and opens the appropriate device information.

Verified Features: FE-5, FE-13
Validated Stakeholder Needs: Intuitive, Customizable

Test Case 9 Password Management

Prerequisites: Test Case 1

The test engineer should confirm the functionality for password management for the mobile application and server backend. Each system should support the following operations:

- Password update – Changing password while logged in to the application
- Password recovery – Changing password while not logged in to the application, via email authentication
- Two Factor Authentication – Turning on two factor authentication and confirming that authentication prompts are asked for prior to log in and that the authentication functions properly.

Verified Features: BE-7
Validated Stakeholder Needs: Secure

Test Case 10 Source Code and Mobile Application Inspection

The test engineer should examine the system source code and compiled application to ensure that stakeholder needs have been met. The test engineer should examine the source code to confirm that the system is sufficiently modular to enable further development and re-use of components. Additionally, they should confirm that, where possible, open-source libraries and tools have been leveraged to implement functionality in the system.

In addition to the source code inspection, the test engineer should also compile the mobile application from source for both iOS and Android devices. It should be confirmed by inspection that the mobile application functions on both platforms according to the typical user workflows identified in this document. This should include confirmation that the user can log in, download device data, and view device configuration information on both mobile platforms.

Validated Stakeholder Needs: Modular, Cost-effective, Portable

7.5 App Store Submission

This application is intended to serve as a functional prototype to demonstrate its utility and to provide a base upon which a final product may be developed by a vendor. Once a vendor completes a production-ready version of the application, it would typically be submitted to one or more of the major app stores (iOS App Store and Google Play app store) to enable easy distribution to end-users. These two major app stores are “walled gardens” where acceptance into the app store is based on approval by the moderation teams within Apple and Google, respectively. The full guidelines and criteria are defined online at https://developer.apple.com/app-store/review/guidelines/ for the iOS store and https://play.google/developer-content-policy/ for the Google Play store. These guidelines touch many areas of the application, the backend systems, and the functionality around it including design, performance, bug/defects, user-submitted content, and user safety. The prototype application has been designed and implemented with these guidelines in mind as of the writing of this report. Given that the prototype has not been submitted for formal review to either app store, some small changes may be required to address comments from the app store review process for the prototype to be a production-ready application.

U.S. Department of Transportation
ITS Joint Program Office – HOIT
1200 New Jersey Avenue, SE
Washington, DC 20590

Toll-Free “Help Line” 866-367-7487

www.its.dot.gov

FHWA-JPO-24-139

U.S. Department of Transportation