AVERT C2
Command & Control

WhiteFox Drone Detection

Technical Design Document

October 11, 2022

Project Summary

The purpose of this feature is to integrate to the WhiteFox drone detection system. When the WhiteFox system detects a drone in the area it is monitoring, it will create a flight in its system that contains the locations of the drone, controller, and home position of the drone. We want to show each of these positions on our map and alert the users of the new flight.

Software Design Considerations

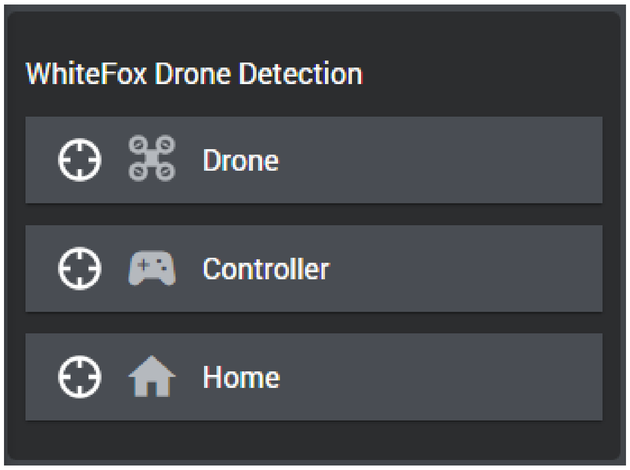
AVERT C2 already supports tracks, so we will utilize this existing functionality to show the positions of the drone, controller, and home position. AVERT C2 also supports alarm activities and rules, so we will utilize these concepts to alert the users of the new flights.

This is the first drone detection integration for AVERT C2, so there are two main additions to the UI that will need to be made. The first is a new widget for the track profile to indicate the relationship between the drone, controller, and home tracks. The second addition is we need to add new map icons to our sprite sheet to support these three new track types.

User Interface Design

Track Association Widget

A new widget will need to be created to show the relationship between the three different tracks for a given flight (drone, controller, and home). Since the widget will be in the profile for one of the tracks, the widget will only show the *other* two tracks for the flight. Each card in the widget should contain a targeting icon, an icon based on the type (drone, controller, home), and the name of the track. The cards should be clickable and clicking on them should open that track's profile. The icon for the drone should be the mdiQuadcopter icon, the icon for the controller should be the mdiController icon, and the icon for the home should be the mdiHome icon, all from the @mdi/js library.



Sprite Sheet Updates
Our current map icon sprite sheets do not contain icons for drones, controllers, or the home track, so we will need to update the sprite sheets to include these. The task for this effort will be to write an init script in ecosystem to replace the existing sprite sheet files in Minio.

User Interface Code Specification

Drone Widget
The new widget should be done in orion-components. In order to determine if this widget should be visible for a user, we will need to add a new “widgets” property to the feedType. This new property will be an array of strings, where the strings correspond to the widget’s ID. In this example, the widget’s ID should be “drone-association”, therefore the feedType’s widgets property should be an array containing “drone-association”. In the track profile, we will need to add the logic to support this new widget condition configuration

This will be the new standard way we support “custom” widgets, which are anything outside of the core widgets (details, files, activities, etc.). At this time, we will **not** go back and update the existing custom widgets to use this method. We will make time to do that later.

Sprite Sheet
We will provide the updates sprite sheet, so the only requirement should be to add an init script in ecosystem to replace the existing sprite sheet files in Minio.

Service Code Specification

Edge Interface
The edge interface for this integration will be a REST Polling interface. We will poll the getDetections endpoint in the WhiteFox API on a configurable interval. Each flight object returned should be treated as an individual message to be published. The API base URL and user ID should be configurable in the external system.

Transform Process
Each flight message received by the transform process should be parsed as a JSON object.

Flight Alarm
We will store the flight ID in Redis with an expiration of 5 minutes so we can keep track of which flights are new flights and which ones are being updated. Once the flight message has been parsed, we will need to check for the flight ID in Redis. If the flight ID exists in Redis, we know we are updating an existing flight, so we should update the expiration of the Redis key to persist for another 5 minutes. If the flight ID does not exist in Redis, we know this is a new flight, so we should publish an alarm activity and then add the flight ID to Redis with a 5-minute expiration.

Tracks

For any flight (new or existing), we will publish the three tracks (drone, controller, and home). It's possible not all tracks will be in every flight message, so we will only publish the tracks that are present in the message. The tracks should be published with a type property of "drone", "drone-controller", or "drone-home". The detections for the flight should be time-ordered, so we should only use the most recent detection for updating tracks. In the following tables, "flight.detection" refers to the most recent detection in the array.

Drone Track Properties

Track Property	Flight Property
entityData.geometry	flight.detection.drone_coordinate (geoJSON)
entityData.properties.disposition	flight.classification
entityData.properties.name	flight.name
entityData.properties.flightId	flight.flight_id
entityData.properties.sourceId	flight.drone_id
entityData.properties.altitude	flight.detection.drone_altitude
entityData.properties.height	flight.detection.drone_height
entityData.properties.speed	flight.detection.drone_speed
entityData.properties.heading	flight.detection.drone_orientation[1] (yaw)
entityData.properties.type	"drone"
entityData.properties.timestamp	flight.detection.timestamp

Controller Track Properties

Track Property	Flight Property
entityData.geometry	flight.detection.pilot_coordinate (geoJSON)
entityData.properties.disposition	flight.classification
entityData.properties.name	flight.name + " Controller"
entityData.properties.flightId	flight.flight_id
entityData.properties.sourceId	flight.detection.pilot_id
entityData.properties.type	"drone-controller"
entityData.properties.timestamp	flight.detection.timestamp

Home Track Properties

Track Property	Flight Property
entityData.geometry	flight.detection.home_coordinate (geoJSON)
entityData.properties.disposition	flight.classification
entityData.properties.name	flight.name + " Home"
entityData.properties.flightId	flight.flight_id
entityData.properties.sourceId	flight.drone_id + "_home"
entityData.properties.type	"drone_home"
entityData.properties.timestamp	flight.detection.timestamp

Database Changes

External system

A new sys_externalSystem record will be necessary to configure the integration. This external system will need to contain all connection information and any additional configuration to support the integration. The external system should be added to the Jira issue as a JSON file.

Feed

A new sys_feedType record will be necessary to support the feed in AVERT C2. This feed type will need to contain all necessary configuration to support the feed, including the new widgets array property. The feed type will also need a mapIconTemplate property to determine which icon to use for each track. This will be provided once we have the updated sprite sheet, but it will use the track's type and disposition properties to determine which icon to use. The feed type should be added to the Jira issue as a JSON file.