RoboTractor: A Secure Signaling and Control System for Remote Management of Agricultural Vehicles using XMPP and REST

Jeremy Wright
Arizona State University
jlwrigh1@asu.edu

Arun Balaji Buduru
Arizona State University
abuduru@asu.edu

David Lucero
Arizona State University
dwlucero@asu.edu

***Abstract*—The need for the automation of vehicles is increasing due to improved efficiency, operation, and reduced operational costs. The scope of this project is to build a signaling and control system with a focus on security, in order to drastically reduce the systems from being compromised. An attack against these systems could allow for the vehicles to be stolen, damaged, hijacked, or cause other losses. The main tasks in this project are to build a secure front-end web interface for users to give instructions, built upon a Representational State Transfer (REST) architecture, set up an Extensible Messaging and Presence Protocol (XMPP) server and client with required authentication, data integrity check mechanisms and encryption, and build a simulated automated vehicle, capable of interfacing with the XMPP components and front-end.**

***Index Terms*—Secure signaling and control, remote management, agricultural vehicles automation, self-driven vehicles**

I. Introduction

Agriculture has a growing problem. Crop input costs such as water, fertilizer, pesticides, and fuel, are increasing. The global population is increasing driving the need for more food. Local, State, and Federal governments are increasing oversight and reporting requirements on growers. Land area available for planting, and harvesting is decreasing globally. All this pushes farmers to lean on technology to improve their efficiency while reducing recurrent costs. Take government over sight requirements [1], growers are required to track where, and how much material is spread into the fields. This can be a considerable overhead for growers, however an automated machine can do this quite easily, and in real-time if desired. This project demonstrates how to securely connect an agricultural machine to an automation server in a secure manner using existing internet technologies. Using existing technologies is a critical point. The market is already stretched thin, and incurring new infrastructure in rural areas can be prohibitively expensive, however leveraging the existing Internet infrastructure we can provide high levels of service in a secure manner.

The most important requirement in developing a secure signal and control system is ensuring authentication of vehicles and data integrity, and securing communication channels. The path plan and location feedback transmission needs to be secured and protected from eavesdroppers, preventing attackers from compromising and/or stealing these vehicles. In this project we use the Django web framework to handle the front end interface and server duties. Django facilitates the use of python libraries and components to interface between our Front-end, server, and client-side vehicle simulators. The end-goal of RoboTractor is to have a comprehensive tool that translates the abstract user inputs into remotely executable actions for agricultural vehicles in a secure manner using advanced web, network, and cryptographic techniques.

II. System Models

A. System Model

RoboTractor will leverage the Django Web framework [2] , to realize the required interfaces. Figure 1 describes the connection of these components. The combination of XMPP and REST in RoboTractor is a demonstration of how to extend the existing HTTP development environment i.e. ”the web of

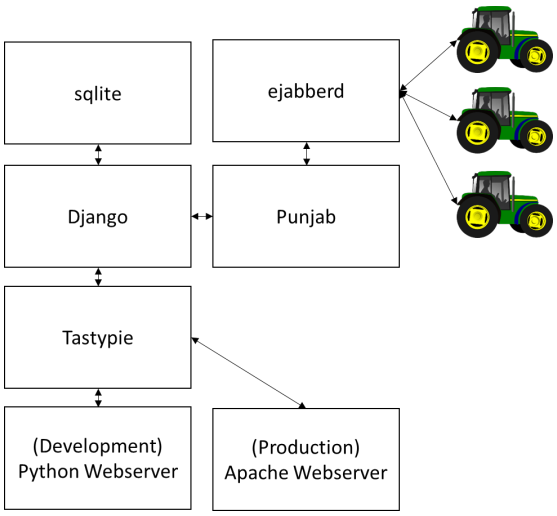

Fig. 1. Software Component Block Diagram

things" into a stateful protocol. HTTP is by design stateless. XMPP on the other hand is a stateful streaming connection between two clients. In this case the Tractor and a command server. To achieve this mesh we will leverage the Bidirectional-streams Over Synchronous HTTP standard protocol [3]. BOSH provides a standard mechanism to operate the streaming XMPP protocol efficiently over an HTTP connection. This is essential for a scalable webservice.

B. Software

1) XMPP Server: RoboTractor uses the ejabberd [4] XMPP server as it provides an existing Python interface, to integrate with the rest of our Python based ecosystem.

2) REST Interface: TastyPie provides REST [5] by extending the existing Django Models.

3) BOSH Interchange: Punjab is an Django plugin implementation of BOSH [6]. In addition to BOSH, this library combines the ejabberd Users with Django Users to provide a single authentication and authorization framework. While this demo project will have a single user type, this combination is critical to maintain proper use management and least-privilege authorization.

C. Security Model

RoboTractor deals with two primary security classes: One involving the vehicles and other assets, and one involving the system components and their communication channels. Because the project is web-based and uses PKI, numerous attack models have been created that this project will deal with. Within the class of communication channels, attacks may be present against the authentication systems of both users and assets. Robotractor currently employs advanced two-factor authentication. Usage of TLS and other best common practices will be implemented to prevent against replay attacks on the system, and also for establishing a secure session between assets and control. A dual-encryption scheme has been devised to send encrypted control signals within an encrypted XMPP stanza to ensure high information assurance. Because this project will utilize PKI, key protection is important. To this end the RoboTractor project plans to secure keys on assets with a Trusted Platform Module or relevant simulated facsimile. By making both key-pairs private and secure, RoboTractor provides advanced communication security. On the other end, the server components must be secured as well using best common practices in server defense, firewall, patches, updates, etc. Additionally, the RoboTractor system will implement an advanced GPS location reporting system to provide physical security in case of asset theft. By defending against these attack models, the RoboTractor project presents the realization of an advanced system for signaling and control with high security.

III. Project Description

Implementation is divided into 2 primary phases (Marked as milestones on the attached Gantt chart). The first phase is an integration phase connecting the various libraries and off the shelf components. It is critical that this integration does not weaken, but rather strengthens the security properties provided by each independent component.

The second phase is the extension of these components to provide command and control of agricultural vehicles. For schedule mitigation, David and Arun are scheduled to begin this phase before the framework is assembled. Once the framework is in place, David and Arun's interface designs may be merged with the overall system. As of the creation of this document, all Midterm deliverables have been achieved. The following tasks have been updated to reflect work currently done, and remaining

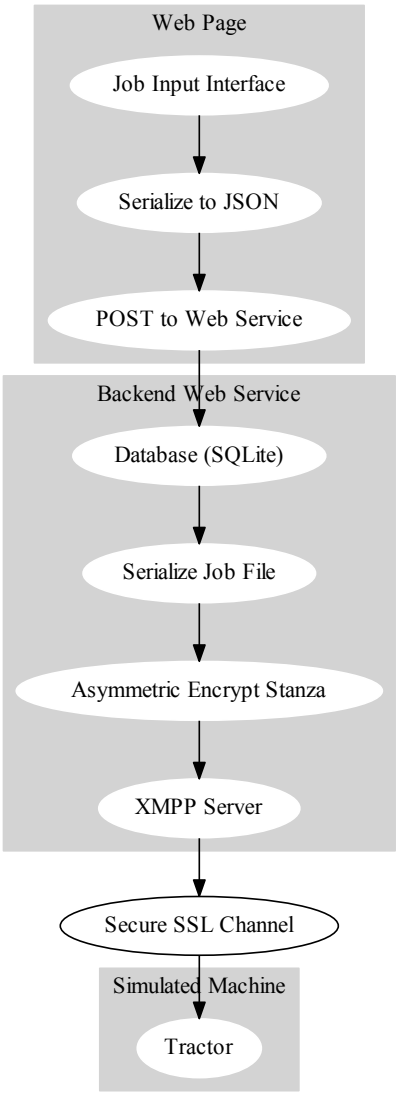

Fig. 2. High Level Signal Path

future work.

A. Task 1 : Development Environment

This task consisted of setting up the development environment for the RoboTractor project. A project homepage was provided to us via the ASU virtual lab project. This home page provides source control in the form of a GIT repo which our project takes advantage of. Additionally, two virtual machines were provided via the ASU virtual lab project to use as our project servers. Setup of these servers

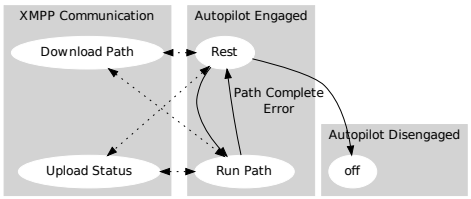

Fig. 3. Tractor Software State Machine

was automated with the creation of a script in order to download all required software packages. Remaining work for this task is to enable proper connectivity between the two assigned Virtual Machines.

RoboTractor on built on the Django. The development environment is best described in [2]. The Development environment consists of 2 Fabric configurations one for development and a second for deployment. This allows development settings to be more relaxed facilitating quick coding cycles, while the deployed configuration is more secure. Separate database also prevent test data from polluting the deployed demo project.

B. Task 2: XMPP Implementation

XMPP Implementation is a configuration task to setup the ejabberd server and connect it into the HTTP Framework. The XMPP client/server components and interfacing with them is not a midterm deliverable, and is to be finished before final submission.

C. Task 3: Demo Tractor

The midterm deliverable for this task was to finalize the design specifications of the tractor simulator. This has been achieved after multiple team discussions, and a barebones state machine vehicle simulator has been written in python. As more tasks are completed, the simulator is to be updated with its required functionality such as XMPP communication, path processing, and status reporting. The state machine consumes the Job File format described in Figure 4.

As described in Figure 2, the job files are stored in the database. When the Tractor is instructed to run a job, the Tractor downloads the Job file from

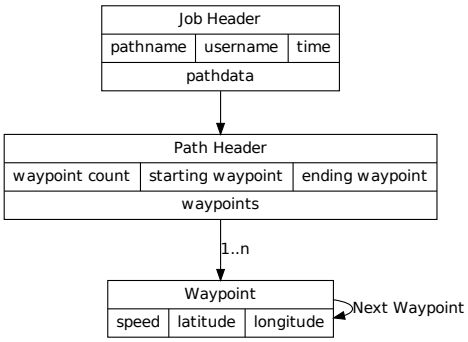

Fig. 4. Job File Format

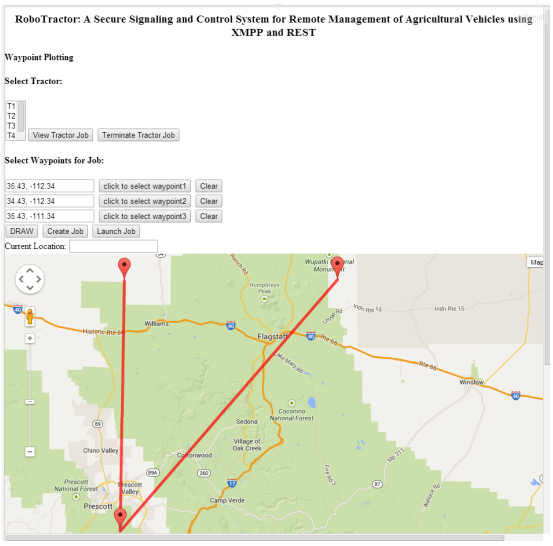

Fig. 5. Job Input Tool

the server over its XMPP connection. The backend server is responsible for translating the database models into JSON encoded Job File.

D. Task 4: Frontend UI

The ”single-page” web application is the modern design methodology to web apps today. Leveraging this design architecture the Frontend will query the REST API to draw the position of all tractors within a Google Map context. Additionally, the frontend user interface is to be highly secure. This task has 2 high level sub tasks:

1) Path generation
2) Map rendering

Path generation is the primary deliverable of this task. The user shall be able to input a path for a given tractor to drive. The Tractor will then receive this path over the REST to XMPP channel. The UI may then periodically query the position of the tractor and update it on the Google Map. Currently, a test-interface has been completed to meet the midterm deliverables, with additional functionality forthcoming.

E. Task 5: Server Backend

The Server Backend is the critical component which connects all components. Extensive knowledge of Django will make this task easier. As it will require linking multiple components together in a orchestrated fashion. We have currently setup and tested our server running on Django, and will continue to update as needed over the course of the project.

F. Task 6: REST Interface

The REST interface will server the primary means of interacting with the site. The Web interface will exist as a ”single-page” app which leverages AJAX principles over this REST API. The REST interface is currently functional, and will be updated to interface further with the frontend interface and backend components as they are completed.

G. Project Task Allocation

From the tasks outlined in section III, the breakdown of work is as follows: Jeremy Wright has configured the development environments and the REST Interface. Jeremy is also looking into modelling trust zones which would allow us to simulate trusted hardware in actual vehicles. He is operating as the project lead due to his experience in Python-based Web Applications. Arun Balaji Buduru is working on the Server Backend and Frontend UI components. He has completed a front-end test interface and is continuing to update it to achieve all required functionality. David Lucero is working primarily on the Tractor simulation agents and also XMPP implementation. A barebones tractor simulation has been completed and will be continuously updated as the project progresses. Each group member has approximately 30% workload of the project with the remaining 10% shared due to the interconnected nature of all components. A detailed task breakdown can be seen in the attached Timeline.

H. Deliverables

Midterm Deliverables:

1) Functioning REST Interface (completed)
2) Front-end Test Interface (completed, with additional two factor-authentication)
3) API documentation for using the REST interface as an external service. (completed)
4) API documentation for using the XMPP interface to act as a vehicle. (moved to final deliverable)
5) Finalized Design of Vehicle simulator (completed, with additional python test implementation)
6) Interim Progress Report (completed)
7) Updated Project Schedule (completed)
8) Interim Project Powerpoint Presentation (completed)

Final Deliverables:

1) Finalized back-end server configuration including easy installation script(s)
2) Completed, secure front-end interface including two-factor auth and SSL/TLS.
3) Vehicle Simulator with working component interfaces and simulated trust zone functionality
4) Final Project Report
5) Final Project Powerpoint Presentation
6) User-guide covering front-end interface, and including API documentation for REST and XMPP interfaces

I. Project Timeline

The attached project timeline (generated from Microsoft Project) describes the overall tasks of the project.

IV. Risk Management of the project

Several potential issues have been identified that may pose risk to successful completion of this project. These risks have been identified in Table I. Along with the description, ratings have been assigned to each risk identified, and potential mitigation strategies are described.

V. Conclusion

In this proposal we intend to build a remote control system leveraging existing internet technologies such as XMPP and REST. Future work based on this approach may include asset management for corporate farm management.

References

[1] (). Growers now have fertilizer, diesel cost information available | cotton content from southeast farm press, [Online]. Available: http://southeastfarmpress.com/cotton/north-carolina-report-will-track-fertilizer-diesel-costs-growers (visited on 03/08/2014).

[2] (). Django, GitHub, [Online]. Available: https://github.com/django (visited on 02/06/2014).

[3] I. Paterson, D. Smith, P. Saint-Andre, and J. Moffitt. (Jul. 2, 2010). Bidirectional-streams over synchronous HTTP (BOSH), [Online]. Available: http://xmpp.org/extensions/xep-0124.html (visited on 02/06/2014).

[4] (). Ejabberd community site | the erlang Jabber/XMPP daemon, [Online]. Available: http://www.ejabberd.im/ (visited on 02/06/2014).

[5] (). Toastdriven/django-tastypie, GitHub, [Online]. Available: https://github.com/toastdriven/django-tastypie (visited on 02/06/2014).

[6] (). Twonds/punjab, GitHub, [Online]. Available: https://github.com/twonds/punjab (visited on 02/06/2014).

TABLE I
RISK MANAGEMENT

Risk Description	Risk of Failure	Consequence of Failure	Mitigation Strategy
Connections between components must be secure	Low	Product would still function, but would lack security information assurance will suffer	Plan to include security on component connections ahead of time. Follow best practices (SSL/TLS)
Evaluation of Project depends on proper vehicle agents	Medium	System is unable to be tested if vehicle agents are inoperable or incorrect	Vehicle agents will be designed first to ensure compatibility with all other components. Thorough review and testing of component to ensure proper functionality
Improper configuration of components	Medium	Components do not communicate properly with each other. System functionality is void	Thorough reviews and testing of components to ensure proper functionality and communication
Incorrect PKI implementation and/or configuration	Medium	Puts all identified security domains at risk. PKI used becomes useless. System becomes vulnerable to outside attacks i.e. MITM	Follow best practices and standards. No re-invention of the wheel
Service Uptime (web reliably)	High	Product is unusable if network connections are not operable	Redundancy. Cloud hosting to alleviate hardware reliance