Security for Robotics

Emily Longman, Zach Rogers, and Dominic Giacoppe

CS461 Capstone

11/13/2016

Abstract

Mission My specific work for the project is, at the moment, managing the software components of the drone itself. Most of our attempts to find vulnerabilities will be at the software level, so deciding what exact software we are using is crucial in how we approach our research.

Drone Operating System

The most important component of the drone itself is the operating system, and as I see it our team has 3 major options: an embedded system, a real time operating system, and a non-realtime operating system. For the purposes of our design, the we will consider Nuttx https://bitbucket.org/nuttx/nuttx/src/master/ for our real time operating system, and Ubuntu+ROS http://www.ros.org/ for our non-realtime. The embedded system would have to be handmade by our team, so there isn't an existing one to use as a comparison. Indeed, this is the first major downside for an embedded system: not only would our team have to create one, it would be specific to whatever we are using hardware-wise and not be portable to other drones unless we made it so. The portability is in direct conflict with the overall mission of the project, which is to make robotics software more secure in general. If our fixes only work for one OS, run by exactly one robot, we aren't doing much to better drone security as a whole. As such there is no particular use for an embedded system for this project. It seems a little like a software equivalent of digging a hole just to fill it. As for nuttx, it avoids that problem by being one of the most used real time drone operating systems. I could not find any statistics on that, nobody seems to take polls on this subject yet, but based on how widely it has been ported and that it happens to be the first open-source result on google when you look up real time drone OS I can only assume it is at least popular. That also means that we will likely have a pre-existing port available for whatever board we decide to use, and will not need to do the porting ourselves, which saves us the possibility of introducing security vulnerabilities ourselves. We would still however have to configure and build nuttx itself on the drone, which may or may not be a problem based on the readme, along with making sure our power supply is compatible. And, as every board's version of nuttx is board specific, any fixes we create are only good for that board's version of nuttx. It does have it's benefits: if setup correctly the team can squeeze out as much power as possible from the drone, and that real time operating systems seem to be the way most commercial and military drones are heading, which means our research is more applicable overall. However, ROS+Ubuntu have the distinct advantage in 3 areas: our project was originally written to use ROS, that ROS itself is not build-specific and is in fact portable to run on top of nuttx among other things, and that 2 members of our group (Zach and I) both have prior experience using ROS. ROS also happens to be a very robust project at this point with a large software ecosystem, and our campus has a ROS expert in Prof. Bill Smart who we could tap for insight. Lastly, while the drone community is moving towards real time operating systems, for robotics as a whole that does not seem to be necessarily true, and while we will most likely use a drone for the project the goal overall is to improve security for robotics. ROS has a large market share on robotics, and at the moment seems to have little competition in the generic robotics system category. As such, our team will use ROS for our project.

ROS v SROS

SROS http://wiki.ros.org/SROS is a port of ROS with the specific focus of being a more secure version of ROS; in essence it aims to achieve a more complete version of our team's goal with a ROS focus. We both desire to better the security of robotics, in particular ROS, through adding security minded enhancements and patching potential vulnerabilities inherit to the design of ROS. It also already has some features that our team would have considered adding to ROS anyways, like TLS support for ROS sockets and certificate management. The project also has a small roadmap for it's goals, which we would give us targets for us to investigate. Lastly, it has an install guide for ubuntu systems, so our team shouldn't have too many issues installing SROS compared to regular ROS. It has 3 major drawbacks, however, as compared to focusing on regular ROS. The most major one is that the SROS project, compared to ROS, is very young with very little general support or package support. Also, based on the last edits to SROS's wiki pages, there has been no work on the project since August, all work up until this point has been done by 2 users, and it's wiki pages are covered in TODOs. In fact, the entire documentation for this project comprises a grand total of 10 pages, none with real explicit details on the current differences between ROS and SROS. In addition, this is the official github repo for the SROS project: https://github.com/osrf/sros. It is almost completely empty, with exactly 2 commits dating back to June of this year. This suggests the project has been closeted by it's creators, which in turn suggests that either they lost interest in the project or that was simply not feasible as planned. Neither of those are good signs for our project, and would suggest that to work on SROS is to take control of the project from it's creators. The next issue is that since it is such a small project, the user base is much smaller than ROS'es, if it indeed has one. Any work our team accomplishes would need to be incorporated into the main ROS distribution channel for it to see much use, and hence we are improving robotics security for a much smaller subset of people. For SROS to be fully adopted by the ROS community, it would need to become a fully fledged ROS equivalent with package support, which is rather outside the scope of our project. We are aiming to improve security, not necessarily make ROS completely secure within 4 months. Third, a lot of SROS'es improvements are tied up in the use of containers, which while fine on it's own limits it's usefulness to systems that can support the amount of overhead a container creates. For bare-metal systems this may not be possible and thus limit our potential audience. Also, nobody on our team has any experience with containers, which would be a limiting factor for our team. For the reasons above, unless the SROS project shows signs of life anytime soon, we will be using basic ROS for our project.

Performance Metrics

wireless/wired/no comms A major factor in how we approach our research is what kind of connection our team will have between the drone and the control station, the control station being the computer the user controls the drone from. We have 3 options: a wired connection, probably some sort of Ethernet cable, a wireless connection using an ad-hoc network between the control station and the drone, and forgoing the control station entirely in favor of a fully autonomous drone. There is also the possibility of using both wired and wireless connections at different points in the project, but a fully autonomous drone would require a very specific drone setup that would be difficult to revert to and from during the course of our project and hence will be considered separately. The major benefit of a wired connection would be that it would have the least possible connection problems and the highest connection latency. The downside is the our current probable drone model flies using propellers, which means that cables could have the possibility of getting caught in them. It also limits the distance we can fly the drone, and means that we have to maintain a line of sight between our control station and the drone at all times, lest we break the cord or disconnect the drone from the control station which would probably result in a crash. Also, while a wired connection offers the most stable and fast connection possible, for the purposes of our project connection speed will not be a major factor in our results. While it would be nice, we are not trying to do anything with the drone per-say and have no intention of doing any operations where milliseconds are the difference between success and failure. What a wired connection would be especially good for in this project would be bootstrapping and flashing the drone as we attempt to modify the drone's OS. Wireless connections require more involved drivers traditionally than wired ones, and the less bootstrapping we have to do lessens the probability of user error. Wireless, however, allows us to have more range on where we can fly the drone, along with allowing the drone controller station to remain completely stationary, and eliminates any concerns about tangling or catching the cord on random objects. However, the project also doesn't really need a large range of flight on the drone, and probably won't go more than a foot or two off the ground for our purposes, so the extra range may not be necessary. Wireless has some potential to drop packets and have signal delays however, which may result the drone doing things it wasn't supposed to do stall in midair, or crash. It does. Also, for both wired and wireless connections, we will have to setup some sort of listener protocol on the drone to allow for communications. Assuming we use ROS this is already made for us, but we will still have to configure our control station to communicate properly with the drone. Lastly, a fully autonomous drone has the distinct benefit that we would not need to create a controller station and pilot the drone at all, which frees up another member from piloting duty. The 2 major downsides are that we would need to create our own autonomous routine, which is a lot of extra work up front for us to produce that is limited to our drone in particular and may not reflect the codebase for autonomous drones as a whole. Additionally, most drones in use today are not fully autonomous but require a human pilot, which means our work may or not be as applicable to drone security as a whole. The only fully autonomous drones I can find in use currently tend to be military drones that perform reconnaissance missions and return, mostly UAV drones. As most of these drones are the products of DARPA and other state agencies, any advances we make in drone security will probably have little affect on autonomous drone security as a whole, unless DARPA decides to start using ROS. As such, we will probably use a combination of wired and wireless communication systems with a control station. Wired when we need to update drone software, and wireless for piloting the drone.