Oregon State University
College of Engineering

CS Capstone Midterm Progress Report

May 15, 2017

Security for Robotics

Prepared for

Oregon State University

Vedanth Narayanan

Prepared by

Group 50

RoboSec

Emily Longman
Zach Rogers
Dominic Giacoppe

Abstract

In drones and other networked robotics there is a broad array of security vulnerabilities that can be leveraged in an attack. We will evaluate ROS to find as many of these security vulenrabilities as we can and document them. The different vulnerabilities found will be categorized into malware, sensor hacks, network and control channel attacks, and physical breaches. For some of these exploits we may be able to implement solutions, which will also be documented. These findings and any solutions will be added to an ongoing academic effort to make robotics more secure.

Contents

1 Overview

It has finally come time to harvest the fruits of this project and turn them into a usable research presentation. After a large amount of addition to the code base of this project in the past six weeks, the available data has been broadened with new and different attack types. When the packages cover more ground, better and more generalizable conclusions can be drawn, which is the overarching purpose of this project.

The specific purpose was to find vulnerabilities in ROS, the Robot Operating System, and then prove their existence by writing code to exploit them. It was quickly discovered that ROS is simply is not designed to be secure by most standards and as such finding vulnerabilities is akin to shooting fish in a barrel. This also made it difficult to follow through on one of our original stretch goals of patching any found vulnerabilities, as they tend to stem from design choice rather than from any sort of coding error. Overall the project went from expecting to find one or two vulnerabilities to seeing how many interesting and unique ways we could break ROS, but the end goal has remained the same: create ROS packages to show the existence of vulnerabilities in ROS. In doing so, our team hoped to show what exactly makes ROS insecure and why, so that moving forward other groups working on ROS would have an idea of what they need to change. At very least for those using ROS know what they need to watch out for.

2 Term Progress

The largest portion of progress this term has been made within the codebase. New exploits in the realms of authentication, network protocols, and the operating system have been made, and documentation for all packages has been updated. After changes in the plan for this term were made, the work on the drone was scrapped and production of exploitation code was increased. Some initial dissection of data was also done for the expo poster, which provided a start for the metrics on the final paper. This mainly involved using the FMEA variables to rank the importance of each exploit and compare their failure modes.

Currently there are 30 exploit packages, which range from sniffing packets to messing around in the Linux kernel. Each team member worked on different sectors of these, sometimes overlapping. Most work was individual however, which allowed more ground to be covered with division of labor and subject matter.

Dominic developed multiple packages that exploited the lack of process pre-checking in ROS packages. ROS does not check to ensure that packages operate in a safe manner before running them by design, and as such any package run by ROS can quite easily do unsafe operations. The creators of ROS simply assumed that all packages run in ROS would all be run intentionally and benevolent in nature, but the fact remains that there is no check on packages run by ROS to ensure good behavior. In particular, ROS packages can make system calls to the underlying operating system's shell, which in turn gives them full run of the robot with the right commands. This discovery was made about 2 weeks into winter term, as it is technically intentional functionality and documented; but it is very unsafe. Dominic was able to use packages to dump the entire root directory of the robot to another system, turn off the robot's wireless networking capabilities, kill all ROS processes, and other problematic actions. Dominic also attempted to create a TCP node that would trick a ROS publisher node into publishing it's data to the TCP node, but was unable to due to technical limitations.

Emily worked to create packages focusing on navigation data and ways to abuse it, as well as OS level exploits of the Linux kernel on which ROS sits. The navigation packages focused either on gps or mavlink protocols, and involved sniffing or spoofing their information. The mavlink exploit was finished before the start of this term, but it never got further than proof of concept since there was no reliable way to test its functionality. Instead the documentation for it was updated to better convey how it should work and discuss what caused the inability to continue production on it. Simply put, this was a combination of a lack of means to test, and even less documentation to guide development. Thankfully the gps spoofing packages were easier to create with the resources available, but still were difficult to actually run without hardware. They worked with the same ideas as the mavlink package, in that they worked to first discover the flight data via sniffing, and then pose as a man in the middle, changing the landing information so as to divert the path of the drone. These were able to be more buffed up this term and documented out so that a reader can understand how they work.

Emily's OS exploits were much different from the previous navigation ones, since they actually affect the Linux kernel rather than ROS itself. They work off the same idea as some of Dominic's in that they are more of a secondary exploit that would be run within the system after a more primary exploit had gained them access. This is the same concept as a bacteriophage latching onto a cell and injecting its genomes, with the primary exploit being one that gains entry to the system, and an OS level exploit being the injected genome. The two OS exploits that Emily created were a classic stack smasher, and a syscall memory hook. These both work within the kernel level, with the stack smasher being a way to get into that space by breaking down the stack barrier, and the memory hook sifting through the stack to find the memory addresses of the syscall table it hopes to rewrite. Even though these aren't

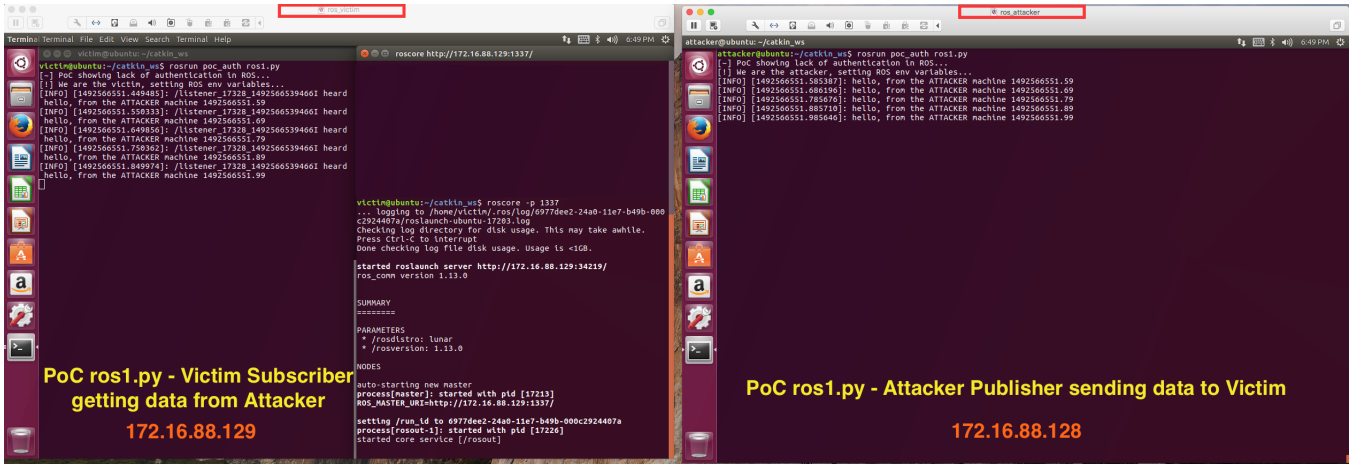

Figure 1: PoC ROS Broken Authentication

within ROS specifically they still would have a great, and very insidious, effect on a system running ROS since it's middleware that sits atop the Linux kernel.

Zach worked on some proof of concept (PoC) packages that involved showing that ROS has zero authentication measures by leveraging the publisher subscriber model. This simple yet effective PoC is the basis for remote ROS exploitation. If you know the IP address of a remote ROS machine, you can leverage it as you wish, without the need to authenticate with the machine in anyway. This should not be taken lightly, as this opens up ROS to any device that has a network connection. The PoC proving this to be true connects to a remote "victim" machine, sending data to a ROS subscriber process, as seen in figure 1.

Zach also worked on PoC packages exploring exploitation of the process communication model that ROS uses. These PoC packages act as ROS security tools, one of which is a subscriber fuzzing tool, and the other a publisher data capture tool. The subscriber fuzzer allows an attacker to target remote ROS processes by flooding them with large amounts of malformed data. This can be used to expose issues with ROS processes on a real world device, like a drone. The remote publisher data capture tool uses a ROS tool called rosbag to create a saved instance of a given ROS process. This PoC uses that data to preform what's known as a "ROS Bag Replay Attack", which can cause devices running ROS to carry out a given operation at the will of the attacker. For example, an attacker could capture what happens with a drone flight control process turns on the motors to begin flight. The attacker could then "replay" that action remotely, causing the drone to fly. It is also possible to modify this captured data before replaying it via ROS, so an attacker could replay a malicious payload quite easily using this method

The majority of progress this term has been in refining preexisting work and in expanding on ideas that had more room for exploration. There are a few more ideas which will likely be implemented as packages after expo, and these can be included into the final research findings. At the moment there is a whitepaper which documents the above work in a more collected fashion and serves as an approachable information source for the concepts of this project, which can be seen in the following section.

3 Research Documentation

4 Conclusion

At this point close to the end of the project there is a broad codebase, a whitepaper detailing it, and the basis of a more formal research paper examining how and why the exploits specifically work. As described above the final exploits cover both breadth and depth of the system and can be analyzed in a variety of ways. There were many problems along the way with the development of all this, and many adjustments were made accordingly. When the drone had eaten up too much time, focus shifted to broadening the software packages and focusing more on that way of breaking into the system. There had been initial plans to include more physical attacks, but they proved impractical to test with out damage to hardware and weren't particularly technical.

Moving more to expanding the exploit packages was a good direction for the project as it prevented a lot of scope creep and could be examined and compared more easily. With the final data there is a more coherent whitepaper and statistical assessments are much easier to correlate. The greatest hope for the future of this project is that practical

solutions can be found to prevent many of these exploits being used in the wild.

References