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

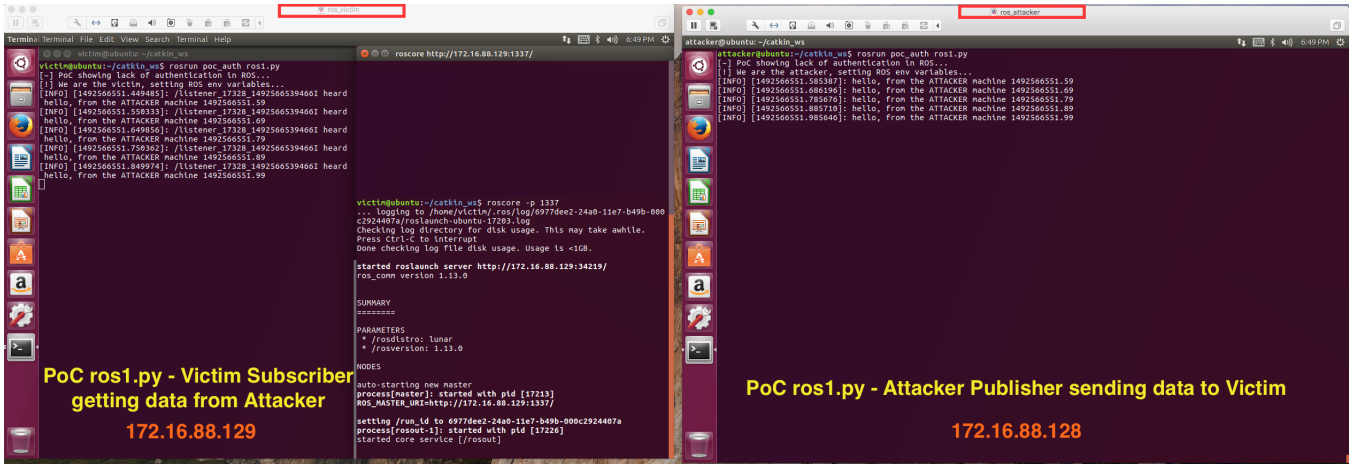


Figure 1: PoC ROS Broken Authentication

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

Below is the first draft of our project whitepaper.

## 3.1 Executive Summary

Our teams purpose was to find vulnerabilities in ROS, the Robot Operating System, and then prove their existence by writing code to exploit them. Our team quickly discovered that ROS is simply is not designed to be secure by most standards and as such that finding vulnerabilities is akin to shooting ducks in a barrel. It also made it hard to follow through on one of our original stretch goals of patching any found vulnerabilities, as they tend to stem from a design choice 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 result has remained the same: showing the existence of vulnerabilities by exploiting them with ROS packages.

## 3.2 Necessity of Project

ROS is known by many users to be insecure, but until now nobody had made any serious effort to document in which ways it is insecure. Our group undertook an exploratory effort to document ways we found it to be insecure as a starting point for what could be a more thorough documentation of the variety of ways it is insecure, for future research or development.

## 3.3 Preexisting Research

This project would have been much more difficult without a lot of documentation and previous research into the security of ROS. One paper that was widely used in the creation of our project basis was Ä Preliminary Cyber-Physical Security Assessment of the Robot Operating System (ROS)∴ [**?**] This article was written by a group of researchers who created a honeypot robotic system and brought it to DefCon 20 to let hackers there find any and all vulnerabilities they could. This lead the researches to be able to recored a range of attack vectors. They also used a variety of their own methods to study what was being done, such as examining the packets being sent and received through wireshark, visualizing the effects with hardware, and using Backtrack Linux. Their work inspired our team to find as many different ways as possible to attacks the system, and to catalog the effectiveness of each different type, relative to the others.

To find inspiration for the different exploits that would be created, a lot of other research papers, journal articles, and security presentations were used.

## 3.4 Packages and Solutions

As mentioned above, the exploits written were meant to cover a lot of ground topic wise. Some of them cluster around similar ideas, while others are a singular representation of an attack vector. Specific packages have been chosen below to represent each of the notable categories of attack. These range from resource consumption, to network sniffing and spoofing, and to OS level exploitation.

### 3.4.1 Malpac 1

Called Malpac in the repo, this is a very simple package. It has 2 nodes, one that publishes a string every couple seconds, and a subscriber that takes that message and prints it to stdout. There is also a third malicious node which also subscribes to the publisher and prints the same message with a slight modification to stdout as well. While not that malicious in this case, the general principle can be used to receive and change any published data, including sending said data to other computers using the ROS subscriber/publisher system and compromise the original datas integrity. For example, if you had a drone doing aerial reconnaissance of an area and transmitting the images back to a base, this kind of system would be able to intercept those images.

### 3.4.2 Malpac 2

This package is a forkbomb. As stated previously there is no process monitoring or control native to ROS, so this is a very simple forkbomb. All it does is do a little math to slow things down a bit, then call itself 8 times and exit. Eventually there are enough processes to clog the CPU and cause a kernel panic, bringing down the robot. In theory, as long as you dont have some sort of process management, this would kill any robot with enough time, which has obvious use cases.

### 3.4.3 Malpac 3

This exploit temporarily disables the OSes wireless networking capabilities. For a drone dependent on wireless communication for control or navigation, this has some rather obvious consequences. It could also permanently disable the wireless networking with a small tweak, which would make the only fix to physically capture and wire back into the robot to fix the networking.

### 3.4.4 Malpac 6

Consisting of 2 parts, this includes one package which launches a publisher that publishes a string every couple seconds and a subscriber that takes that string and prints it to stdout. The other one runs a package that kills the subscriber from the 1st package, and replaces it with its own subscriber with the same name that prints a modified message to stdout. At the time we thought the node would be indistinguishable from the previous node besides the

different message, but it turns out that ROS silently assigns each publisher and subscriber its own unique ID and is in fact quite noticeable. It might be hard to notice by an unassuming user, assuming they dont notice the slight hiccup where the old subscriber dies and is replaced, so it still has potential to be a threat or phone home.

#### 3.4.5 Malpac 7

This is a package that performs bogo-bogo sort. [**?**] Bogo-bogo sort is an extremely inefficient sorting algorithm that basically performs a random shuffle to sort a list, recursively. While it is not particularly system intensive, the sort for a list with a mere 10 elements can take on the order of days. The goal of this package was not to be directly malicious but more of a lurking threat; it shows that ROS also doesnt check for any particularly long running processes, which allows for a variety of actual threats. The best example is an attack that lies dormant until signaled to do maximum damage, or one that waits for a specific process to kill it.

#### 3.4.6 Malpac 8

This is a package that changes the ROSMASTER URI to an arbitrary one, and starts a new ROS master to match. The URI is basically where packages check for the ROS master node, which is the master in the ROS master/slave system as one might expect. ROS master manages nodes, and the publisher/subscriber system between nodes as its 2 main duties. As such, changing the URI and starting a new ROS master node means that all new nodes will talk to the new ROS master as opposed to the old, which could allow for some interesting node management or fiddling with the publisher/subscriber system. The ROSMASTER URI can also be set to a remote machine if it setup correctly as well, which means you could have a compromised robot respond to a remote ROS master node as well, allowing an attacker to control the robot remotely.

#### 3.4.7 Malpac 9

This simply pkills (process kill, for those unfamiliar) all processes with ROS in the name. This will at least kill ROS master, and maybe some other related ROS processes, but killing ROS master also kills all the nodes currently being managed by ROS master, so it will also kill all ROS processes on the system. This has the obvious effect of bringing the robot to a halt, and has the potential of doing major damage. Just image if a drone carrying an explosive payload fell out of the sky because its control system suddenly stopped.

#### 3.4.8 Malpac 10

This package uses wget to attempt to download all of Wikipedia. It probably doesnt succeed, unless the robot has enough disk space to hold all of Wikipedia, but it will fill the disk up to the limit. This can cause a number of issues depending on what else needs disk space, but it is sure to cause some sort of issue. If nothing else it will prevent any more ROS processes from being launched, as ROS master does some record keeping for each process and will need a little disk space for that purpose.

#### 3.4.9 Malpac 14

attempts to install a bitcoin mining service on the robot. It does currently need sudo permissions for this, as it does need to use apt-get the way it is set up, which means that ROSCORE needed to be run as a superuser so that this child process would have those permissions as well. If the install succeeds however, then you have a bitcoin miner using CPU time for someone elses monetary gain, and with a little more work could be further configured to have the miner run at boot as a system process, but for the purposes of the package it was good enough as is.

#### 3.4.10 Malpac 15

Here was an attempt to abuse how ROS subscribers and publishers really worked outside of ROS. All subscribers and publishers actually communicate through a shared TCP/UDP socket controlled by ROS master, and the general idea was to make a non-ROS socket to communicate with the ROS socket, subscribe to a known publisher on the ROS machine, and print whatever they published as plain text to stdout. In short, something like malpac 1, but going around ROS to do so. Unfortunately ROS uses TCP/UDP with a special header encoding that makes this difficult, as to communicate any publisher or subscriber you need their unique ID, which is assigned to them by ROS master at their launch. And to get said ID, you need to do some handshaking with ROS master, which turned out to be a bit beyond our technological capacity and time constraints. We believe it is still possible to achieve this, but to do so would require a better working knowledge of sockets and more investigation into the handshake protocol

used by ROS master. While the documentation is available online, specifically on the technical overview page of the official ROS site[**?**], it was just a little too much for the team with the time we had.

#### 3.4.11 Malpac OS

This package contains two scripts, a memory hook and a stack smasher. The first, the memory hook, is actually a Linux kernel exploit, and doesn't use ROS itself, but ROS commonly sits atop Ubuntu or Ubuntu-based linux systems. This package does a simple memory hook into the syscall table, which allows the script to make changes to whichever syscalls it wants once it finds the syscall table memory address. This is done by changing the pointer to the function of an existing syscall and replacing it with a pointer to a malicious function. This is the kind of script that would likely be secondary to one of the network attacks. One of those might be used to get into the system, and could then install LKMs like this one. Since it operates at the kernel level in Linux, ROS would have no way of ever knowing something was amiss, which renders this a very insidious threat.

The second script, which does stack smashing, provides a basic proof of concept for stack smashing. This is a form of buffer overflow attack which has been used reliably for a very long time to gain unauthorized access to a system. It works by leveraging the overflow to push its way into privileged space and then execute some form of malicious code once it has entered that area of the stack. This specific script is a small scale buffer so better demonstrate the idea. To use a simple approach like this, one needs access to the environment variables of the system so it's not perfect.

#### 3.4.12 Malpac GPS

The exploits in the category work with the GPS or Mavlink protocol. Specifically the GPS package uses both spoofing and man in the middle to trick a drone into thinking it is the home base, and can misguide it. This is done by sniffing for drone network information, and then leveraging that to send spoofed landing data via UDP. The Mavlink script follows the same idea, but it was never able to reach a point of development beyond simply sniffing for the information and positioning itself as a node in the middle. With better documentation and a system with which to reliably test it the Mavlink script could grow to the point of the GPS one.

### 3.5 Conclusion

ROS is vulnerable in at least the ways detailed above, and as such is insecure. There are also vulnerabilities whose existence we are confident of, but were unable to write code to exploit in the given time. In particular, due to our inability to repair our test drone in time, we were unable to attempt any exploits involving hardware. Most of the vulnerabilities come from basic design choices and philosophy in ROS itself, and to attempt to fix said vulnerabilities would take fundamental changes and re-designs in ROS. There are indeed already undertakings to do just that, specifically SROS and ROS2, but overall it is probably better to design drone security around the fact that ROS is insecure, if one chooses to use ROS, than attempt to fix ROS itself. Many of our vulnerability exploits come from having access to ROS itself or by abusing lax communications security to get into ROS, so just ensuring that the ROS is only accessible by trusted users is enough to prevent most exploits that we have found, although that in itself is a rather big task.

## 4 Midterm Progress 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