CS Capstone Whitepaper

May 18, 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 holes 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 Introduction

Robotics is still a relatively up and coming field and as most efforts in robotics are in pursuit of furthering capabilities, security has been left largely untouched. Because of this many functioning, deployed robots have limited to no security in their internal systems, making them very vulnerable to attacks. While the community developing robotics is still largely academic and there is little worry of being attacked, the security vulnerabilities still exist. Soon robotics will become ubiquitous in society and hackers will exploit these vulnerabilities for personal gain. There have already been reports of smart appliances being used for botnets, it's only a matter of time before drones and other robotics are similarly abused. [1] Through our work we hope to eliminate some of this abuse before it begins.

2 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. ROS turns out to be very vulnerable by design, and as such we found many vulnerabilities. Since we had an abundance of vulnerabilities, the team decided to not just document them but to also compare them amongst each other as a way of identifying which might be larger priorities for any successors using our work.

3 Problem Statement

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. This paper summarizes our findings.

4 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).̈ [2] 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.

One particularly useful paper was written by Bernhard Dieber et al, which discussed security on the application level of ROS. It provides great insights for the problems with authentication and data integrity that are present. [3] A similar paper entitled R̈OS: an open-source Robotic Operating Systemẅas vastly useful in the planning of this project, as it gave a detailed over view of the system from a research standpoint. While it didn't discuss security explicitly, it helped us to have a better grasp over the system we were attempting to exploit. [4] The last of the papers that were invaluable in the overall planning and implementation of this project was one discussing the SROS project. This project is closely aligned with ours in that they are working to bring better security to ROS in its most vulnerable sectors. The areas in which this project most wanted to provide security were some of the first that this project investigated. [5]

To find inspiration for the different exploits that would be created, a lot of other research papers, journal articles, and security presentations were examined. Rather than listing them out here, they can be found referenced within the context they were used in our requirements and design documentation.

5 Threat Model

A threat model was created to visualize the three categories. It serves as a roadmap for the research and can be looked back on at any time. The model below in Figure 1 is organized as a tree, with the three main branches being the three components of the CIA triad (Confidentiality, Integrity, and Availability). This way one can look at our threat model, decide if their concern is one of confidentiality, integrity, or availability, then examine the vulnerabilities in that category. Not every area from the threat model was successful, some were not possible to breach, but its still good to acknowledge that they are still likely targets.

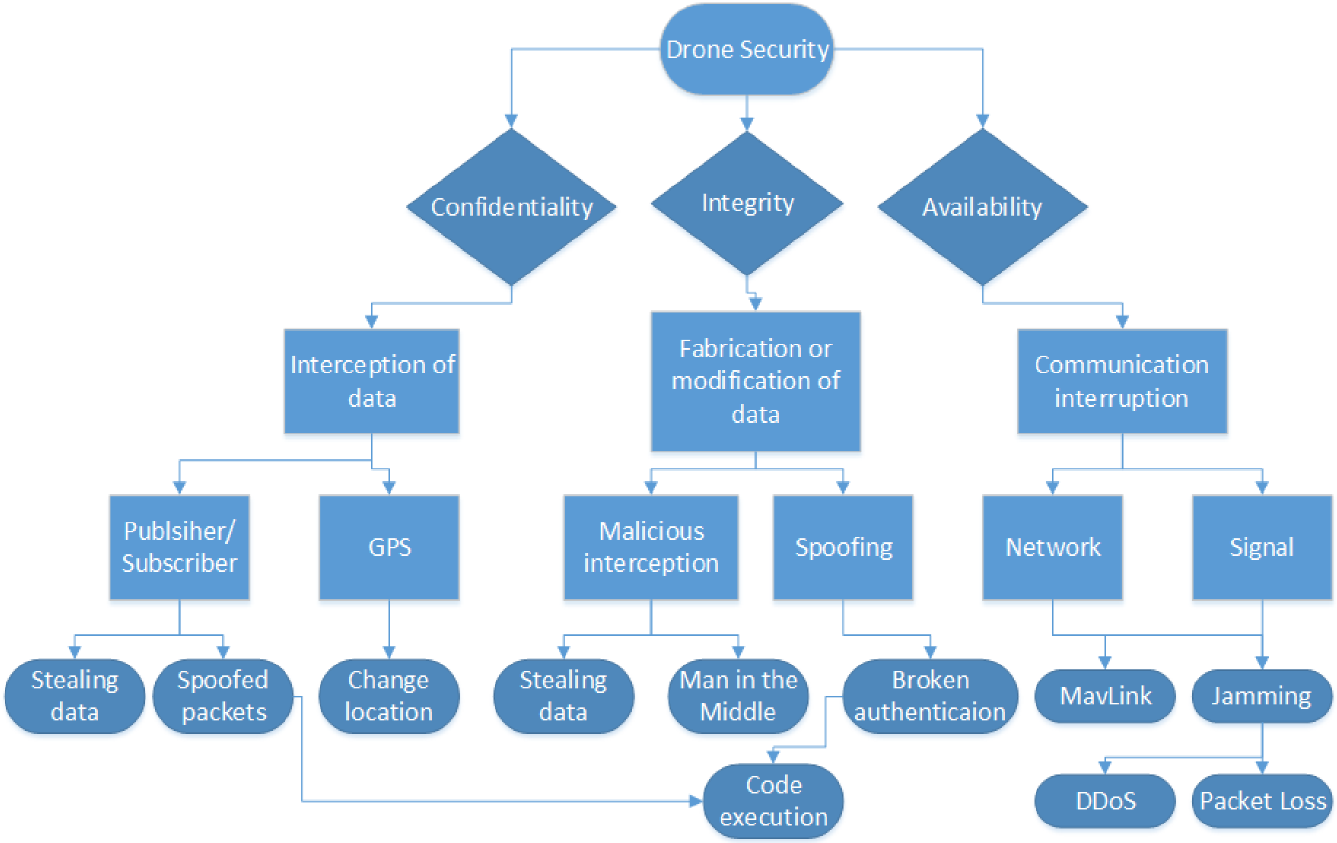


Figure 1: A view of the attacks that are viable against the ROS system

6 Packages and Solutions

As mentioned above, the exploits written were meant to cover a lot of ground topic wise. Some are based on the same vulnerability, while others are more distinct. 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.

6.1 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.

6.2 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.

6.3 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.

6.4 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.

6.5 Malpac 7

This is a package that performs bogo-bogo sort. [6] 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.

6.6 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.

6.7 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.

6.8 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.

6.9 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.

6.10 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[7], it was just a little too much for the team with the time we had.

6.11 Malpac OS

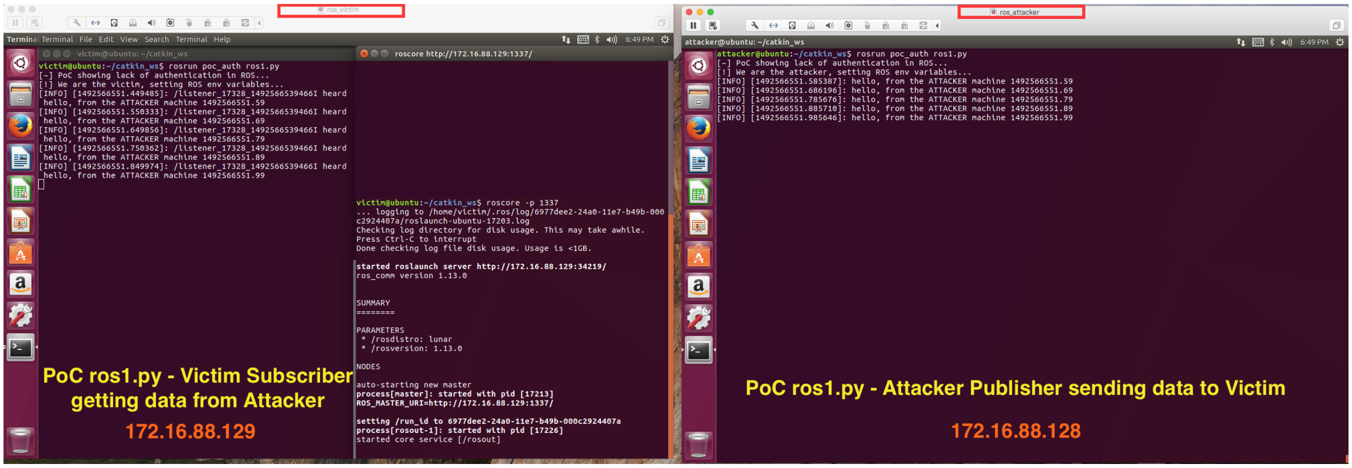
This package contains two scripts, a memory hook and a stack smasher. The first, the memory hook, is actually a Linux kernel exploit, and doesn't use ROS itself, but ROS commonly sits atop Ubuntu or Ubuntu-based linux systems. This package does a simple memory hook into the syscall table, which allows 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, the stack smasher, provides a basic proof of concept (PoC) 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.

6.12 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.

6.13 ROS Broken Authentication PoC


ros_victim
Terminal File Edit View Search Terminal Help
victim@ubuntu: ~/catkin_ws
roscore http://172.16.88.129:1337/
6:49 PM
ros_attacker
attacker@ubuntu: ~/catkin_ws
PoC ros1.py - Victim Subscriber getting data from Attacker
172.16.88.129
PoC ros1.py - Attacker Publisher sending data to Victim
172.16.88.128

```
victim@ubuntu:~/catkin_ws$ rosrun poc_auth ros1.py
[~] PoC showing lack of authentication in ROS...
[!] We are the victim, setting ROS env variables...
[INFO] [1492566551.449485]: /listener_17328_1492566539466I heard
 hello, from the ATTACKER machine 1492566551.59
[INFO] [1492566551.550333]: /listener_17328_1492566539466I heard
 hello, from the ATTACKER machine 1492566551.69
[INFO] [1492566551.649856]: /listener_17328_1492566539466I heard
 hello, from the ATTACKER machine 1492566551.79
[INFO] [1492566551.750362]: /listener_17328_1492566539466I heard
 hello, from the ATTACKER machine 1492566551.89
[INFO] [1492566551.849974]: /listener_17328_1492566539466I heard
 hello, from the ATTACKER machine 1492566551.99
```

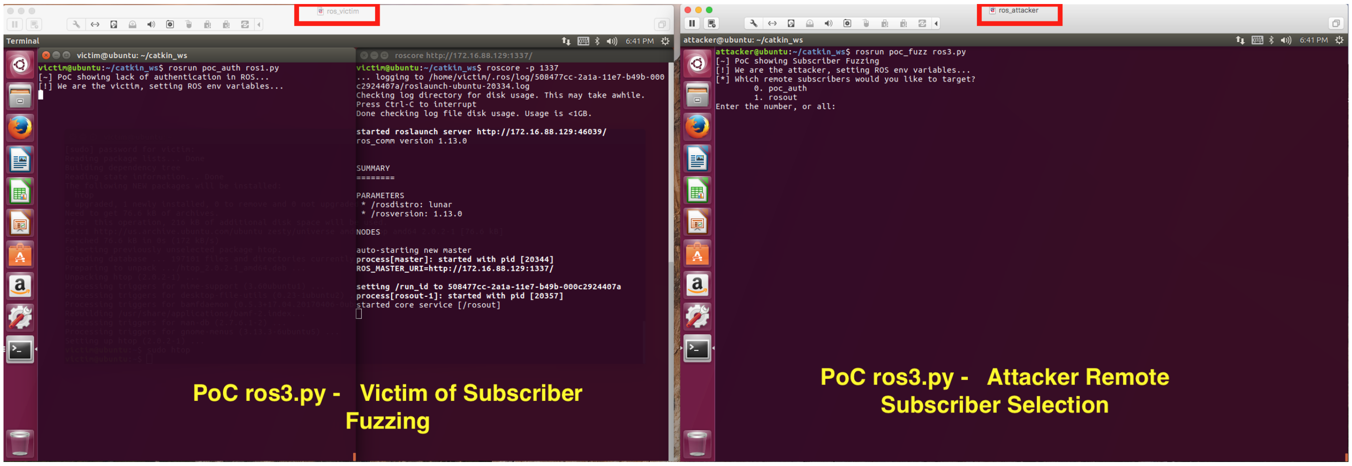
```
victim@ubuntu:~/catkin_ws$ roscore -p 1337
... logging to /home/victim/.ros/log/6977dee2-24a0-11e7-b49b-000
c2924407a/roslaunch-ubuntu-17203.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://172.16.88.129:34219/
ros_comm version 1.13.0


SUMMARY
========

PARAMETERS
 * /rosdistro: lunar
 * /rosversion: 1.13.0

NODES

auto-starting new master
process[master]: started with pid [17213]
ROS_MASTER_URI=http://172.16.88.129:1337/

setting /run_id to 6977dee2-24a0-11e7-b49b-000c2924407a
process[rosout-1]: started with pid [17226]
started core service [/rosout]
```

```
attacker@ubuntu:~/catkin_ws$ rosrun poc_auth ros1.py
[~] PoC showing lack of authentication in ROS...
[!] We are the attacker, setting ROS env variables...
[INFO] [1492566551.585387]: hello, from the ATTACKER machine 1492566551.59
[INFO] [1492566551.686196]: hello, from the ATTACKER machine 1492566551.69
[INFO] [1492566551.785676]: hello, from the ATTACKER machine 1492566551.79
[INFO] [1492566551.885710]: hello, from the ATTACKER machine 1492566551.89
[INFO] [1492566551.985646]: hello, from the ATTACKER machine 1492566551.99
```

Figure 2: PoC ROS Broken Authentication

Two PoC packages were created that show ROS has zero authentication measures, by leveraging the publisher subscriber model. These simple yet effective PoCs are 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 any way. 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. This complete PoC package extends upon Malpac 8 and 15.

6.14 ROS Process Communication


ros_victim
Terminal
victim@ubuntu: ~/catkin_ws
roscore http://172.16.88.129:1337/
6:41 PM
ros_attacker
attacker@ubuntu: ~/catkin_ws
PoC ros3.py - Victim of Subscriber Fuzzing
PoC ros3.py - Attacker Remote Subscriber Selection

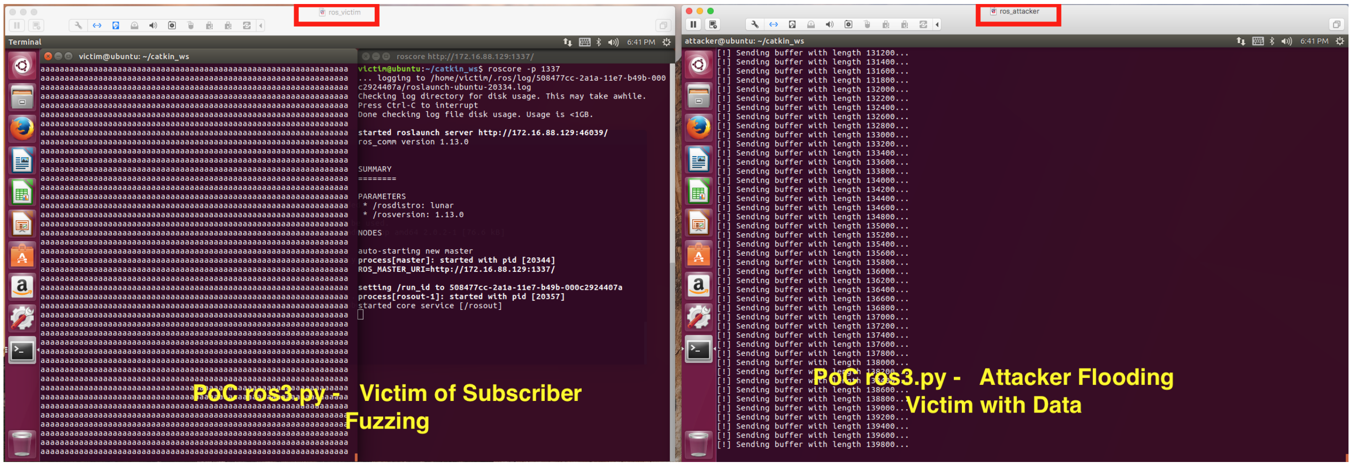
```
victim@ubuntu:~/catkin_ws$ rosrun poc_auth ros1.py
[~] PoC showing lack of authentication in ROS...
[!] We are the victim, setting ROS env variables...
```

```
victim@ubuntu:~/catkin_ws$ roscore -p 1337
... logging to /home/victim/.ros/log/508477cc-2a1a-11e7-b49b-000
c2924407a/roslaunch-ubuntu-20334.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://172.16.88.129:46039/
ros_comm version 1.13.0


SUMMARY
========

PARAMETERS
 * /rosdistro: lunar
 * /rosversion: 1.13.0

NODES

auto-starting new master
process[master]: started with pid [20344]
ROS_MASTER_URI=http://172.16.88.129:1337/

setting /run_id to 508477cc-2a1a-11e7-b49b-000c2924407a
process[rosout-1]: started with pid [20357]
started core service [/rosout]
```

```
attacker@ubuntu:~/catkin_ws$ rosrun poc_fuzz ros3.py
[~] PoC showing Subscriber Fuzzing
[!] We are the attacker, setting ROS env variables...
[*] Which remote subscribers would you like to target?
        0. poc_auth
        1. rosout
Enter the number, or all:
```

Figure 3: PoC ROS Fuzzer Selection

Two additional packages were created, 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.


ros_victim
Terminal
6:41 PM
victim@ubuntu: ~/catkin_ws
roscore http://172.16.88.129:1337/
ros_attacker
attacker@ubuntu: ~/catkin_ws
6:41 PM
PoC ros3.py - Victim of Subscriber Fuzzing
PoC ros3.py - Attacker Flooding Victim with Data

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

```
victim@ubuntu:~/catkin_ws$ roscore -p 1337
... logging to /home/victim/.ros/log/508477cc-2a1a-11e7-b49b-000
c2924407a/roslaunch-ubuntu-20334.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://172.16.88.129:46039/
ros_comm version 1.13.0


SUMMARY
========

PARAMETERS
 * /rosdistro: lunar
 * /rosversion: 1.13.0

NODES

auto-starting new master
process[master]: started with pid [20344]
ROS_MASTER_URI=http://172.16.88.129:1337/

setting /run_id to 508477cc-2a1a-11e7-b49b-000c2924407a
process[rosout-1]: started with pid [20357]
started core service [/rosout]
```

```
[!] Sending buffer with length 131200...
[!] Sending buffer with length 131400...
[!] Sending buffer with length 131600...
[!] Sending buffer with length 131800...
[!] Sending buffer with length 132000...
[!] Sending buffer with length 132200...
[!] Sending buffer with length 132400...
[!] Sending buffer with length 132600...
[!] Sending buffer with length 132800...
[!] Sending buffer with length 133000...
[!] Sending buffer with length 133200...
[!] Sending buffer with length 133400...
[!] Sending buffer with length 133600...
[!] Sending buffer with length 133800...
[!] Sending buffer with length 134000...
[!] Sending buffer with length 134200...
[!] Sending buffer with length 134400...
[!] Sending buffer with length 134600...
[!] Sending buffer with length 134800...
[!] Sending buffer with length 135000...
[!] Sending buffer with length 135200...
[!] Sending buffer with length 135400...
[!] Sending buffer with length 135600...
[!] Sending buffer with length 135800...
[!] Sending buffer with length 136000...
[!] Sending buffer with length 136200...
[!] Sending buffer with length 136400...
[!] Sending buffer with length 136600...
[!] Sending buffer with length 136800...
[!] Sending buffer with length 137000...
[!] Sending buffer with length 137200...
[!] Sending buffer with length 137400...
[!] Sending buffer with length 137600...
[!] Sending buffer with length 137800...
[!] Sending buffer with length 138000...
[!] Sending buffer with length 138200...
[!] Sending buffer with length 138400...
[!] Sending buffer with length 138600...
[!] Sending buffer with length 138800...
[!] Sending buffer with length 139000...
[!] Sending buffer with length 139200...
[!] Sending buffer with length 139400...
[!] Sending buffer with length 139600...
[!] Sending buffer with length 139800...
```

Figure 4: PoC ROS Fuzzer Execution

7 Testing Methods

Failure Mode Effects Analysis (FMEA) is a procedure used in a variety of fields which creates an empirical system for testing and logging any failures, which can be applied to all of our attacks. Somewhat akin to risk analysis, FMEA involves defining severity, occurrence, and detection rating scales, usually from 1 to 10. After these scales have been defined one can use them to calculate a risk priority number (RPN) and a criticality number with which the found failures can be ranked in order of importance. You can see this represented in figure 5 below.

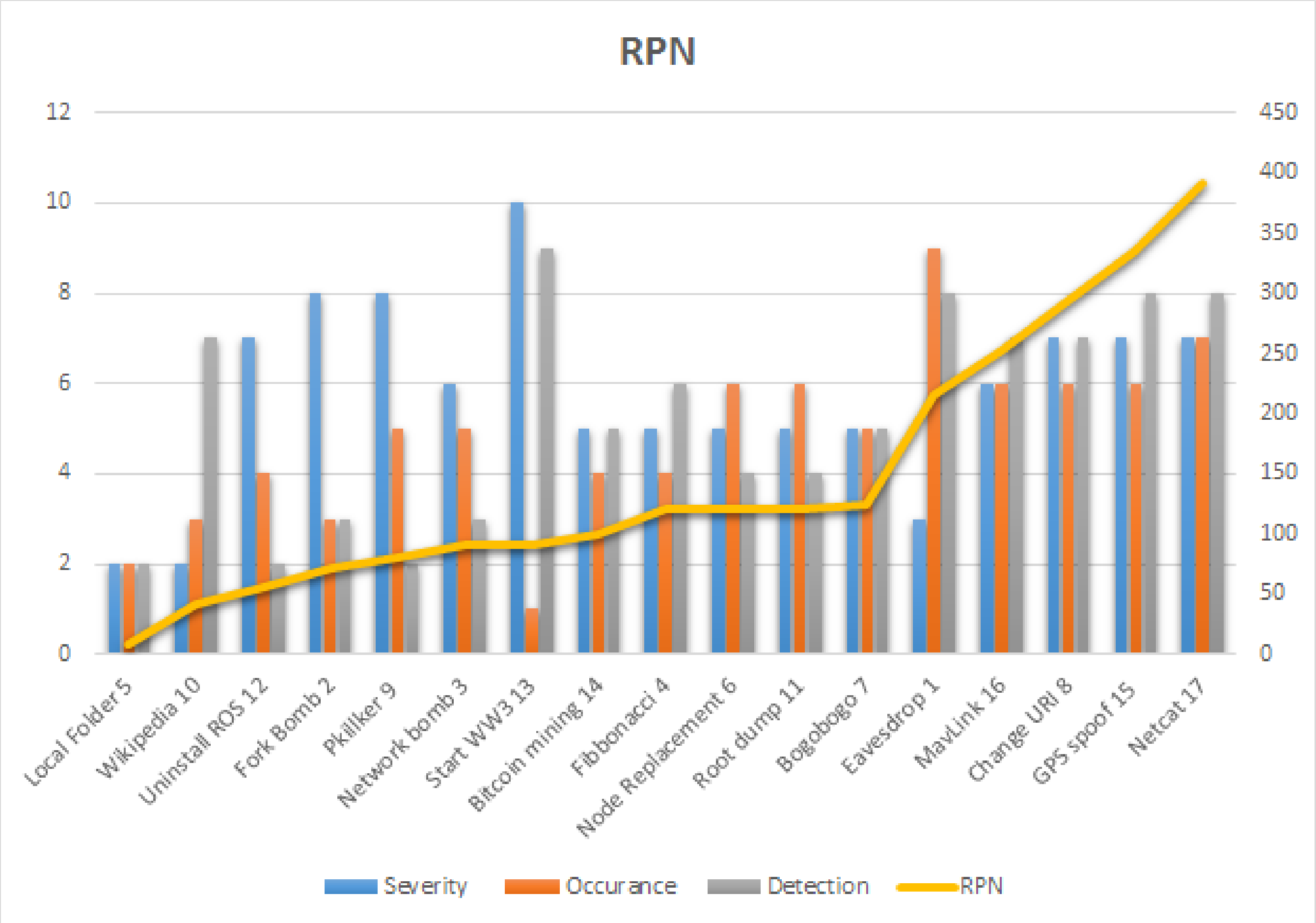


Figure 5: The increasing criticality ratings of packages, based on the three combined numbers

8 Findings

Just as suspected, ROS has wide range of vulnerabilities that this project exploits with a success rate of nearly 100%. Most of these exploits involved either entering the system via the publisher/subscriber system that ROS uses to communicate, or demonstrated what malicious actions could take place on the system once inside. Those that interrupted the availability of the system were most prevalent, since it's somewhat trivial to break subsystems once inside. The next most common type of exploit was those that tamper with the integrity of data, usually in network communication. The least common were those that broke the confidentiality of data, often through authentication or sniffing.

We also found that our project evolved as we went along. We had initially intended to do more hardware based attacks, but they proved to be too cumbersome and difficult to generalize. There are also a board range of physical attacks ranging anywhere from an EMP type gun to an eagle taking down an airborne drone.

9 Limitations

The main limitation our project faced was not being able to do testing on actual robotic hardware. If this project were continued, the next steps would be to setup a ROS enabled drone and see how our exploit packages effect the drone's functionality. Testing on real world hardware will drive home the real-world implications of using an insecure middleware on robotic devices with open communication channels. Our research is not just limited to drones; any device running ROS can be used to further test our methods. While our testing was done on ROS enabled machines, those machines were not real-world robotic devices, thus the real-world impact of our packages could not be evaluated.

10 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.

References

[1] S. Sharma, S. Garg, A. Karodiya, and H. Gupta, "Distributed denial of service attack."

[2] J. McClean, C. Stull, C. Farrar, and D. Mascareas, "A preliminary cyber-physical security assessment of the robot operating system (ros)," pp. 874 110–874 110–8, 2013. [Online]. Available: http://dx.doi.org/10.1117/12.2016189

[3] S. R. Bernhard Dieber, Severin Kacianka and P. Schartner, "Application-level security for ros-based applications."

[4] M. Q. et al, "Ros: an open-source robot operating system."

[5] S. R. Bernhard Dieber, Severin Kacianka and P. Schartner, "Application-level security for ros-based applications."

[6] D. Morgan-Mar. Bogobogosort. [Online]. Available: http://www.dangermouse.net/esoteric/bogobogosort.html

[7] O. S. R. Foundation. Ros/ technical overview. [Online]. Available: http://wiki.ros.org/ROS/Technical%20Overview