Implementing and Testing a Framework for Secure Applications on General Purpose Robots

Corbin Souffrant*
Department of Computer Science
University of Illinois at Urbana-Champaign

Abstract

As technology becomes more advanced, things that are seemingly sci-fi quickly become reality. Personal robots are becoming more of a reality every year, and there has been little research in the area of application development on robots for general use. In this paper, we are showing a new framework that allows an every day person the ability to develop an application on a general purpose robot, and also helps to prevent the developer from being able to break any standards of security or ethics that need to be considered in a robot. Using this framework, we developed a variety of applications that took advantage of capabilities such as microphone, camera, and speaker control, but also used the extra abilities granted from being a robot.

1 Introduction

Robots are also becoming increasingly common in the world, with some even being used for personal purposes. One of the first developed applications of robotics was robots for industrial use. This allowed manufacturers to quickly and efficiently produce products that were uniform at a much faster rate than possible with humans. Another important area of robotics is military use. Developments are being made for airplanes that can pilot themselves, or be able to locate and target enemies, as well as surveillance. Household robots are also starting to make their way into society, with products such as the Roomba, which can automatically vacuum a house, and also toys for children to play with.

Society is getting closer to becoming more capable of acquiring robots. And along with this development, people are also increasingly receptive to these new technologies. For example, only a couple decades ago, it was almost unheard of for a person to have a phone that they could carry around with them. Whereas now, there are a lot of people who not only have cellphones, but cellphones that are small personal computers. They are able to browse the internet, play games, run applications, write documents, and many other things that one would not normally associate with a phone. Having more interaction is not far off from this, leading to the development of personal robotics.

While there have been a few papers that can be applied to these robots, to the author's knowledge there are a lack of papers discussing their security. Previously, we have done research on developing a framework for building applications, but we will further this study by introducing some modifications of the framework, as well as introducing test applications to provide evidence that the framework is in fact secure.

The difficulty with creating this framework is the existence of systems involved in robotics that one would not find in a computer. For example, privacy issues become even more pertinent as robots are able to record, move around, and possibly make their own decisions. They interact with the world in a way that was not previously possible without human intervention, which provides new security problems that one must consider.

*e-mail: souffra2@illinois.edu

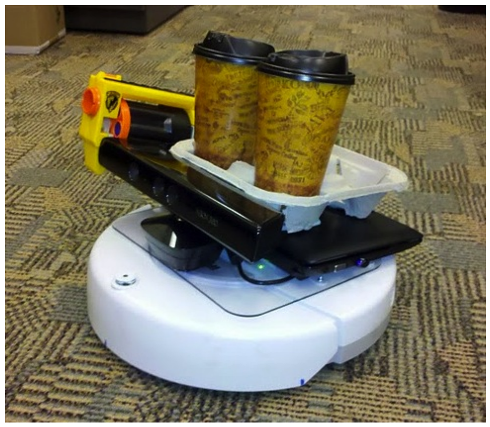

Figure 1: *Figure 1: The robot used for this paper is pictured above*

In this paper, we will address a variety of systems that a robot using the ROS middleware may have, and discuss ways that we were able to provide the ability for people to develop applications that use these systems in a secure manner. The systems that we discuss are the robot's ability to move around, know its location, use the microphone, speak in the speaker, and record things with its camera. We attempt to find a way to make it easy to develop applications that take advantage of these systems, using an API that works with our framework. This API uses an abstraction that makes the more complex architecture of our framework easy to implement in terms of a combination of everyday language and basic object-oriented programming.

The robot (seen in fig. 1) was developed using commercial parts, which extends to the practicality of the robots that we planned on being available for purchase by general consumers. It uses an iRobot system as a base, a netbook in order to control the systems, and a kinect for the camera functions. The framework and applications that we developed can be easily extended and adapted to other robots with various functionalities and purposes, which is an advantage of the way that we developed it.

We have created a comprehensive application that emulates a bartending application in order to test the API and framework that we developed for the robot. This application drove around looking for new users and asked them for drink orders, it then drove back to the bartender, gave him the drink orders, and sent them back to the users.

The contributions are:

-Extending our previous research in the initial development of a secure framework

-Providing an easy to use API for programmers to develop applications

-Testing the framework for the appropriate functionality

We start this paper by discussing related works to our robotics project, and the details of how they contributed to the design and implementation of our secure framework. We then speak about our framework and API in detail, and how it was implemented. We conclude with the results of the test applications we created, and how it showed the functionality of our robot.

1.1 Research Questions

There are a few different questions that are relevant to the work that we have done for this paper.

-How do you extend current computer framework models to include the additional capabilities of robots?

-How do you implement security into a robot, while still allowing functionality of the extra features?

-How do you make it easy for developers to write applications on this framework, while still providing the security previously discussed?

2 Related Work

Although it is becoming closer to the period when robots may inhabit a household, there has not been a lot of research in the area of sercurity in the application framework of these general purpose robots.

Recently, a team in Munchen, have utilized the PR-2 robot to make pancakes. [Michael Beetz and Tenorth 2011] This was interesting research that had been done in the field of robotics and functionality, as the robot had object recoginition, and without the users help, managed to find the ingredients for pancakes, grab them, and then make the pancakes. One of the more interesting pieces about this paper was the fact that it allowed for alteration in the environment. During the live test demo of this pancake making session, the users would manipulate the environment by moving around objects and making it less straight forward for the robot to be able to just follow a set by set list of commands.

Another paper in the field of general robotics system was done by finding a new way of allowing mistakes to be seen by looking at the logging system of the robot. [Lorenz Mosenlechner 2010] We worked on developing a logging system based off of this research in order to look at mistakes that the robot made and why they made the mistake. This is something important for the field of personal robotics in a secure framework. If an application fails, or makes some sort of mistake, (See the future section on acceptable mistakes.) it is important to look into why it made the mistake, so that someone would be able to fix it.

3 Threat Model

In this paper, we are considering the goal that there is an application store for the robot, where any user can develop an app, and release it into the store for any other user to download. We need our framework to ensure confidentiality, integrity, and availability. Because this is a robot, there is additional concern in the area of privacy and physical harm.

We are using a slightly modified threat model from the paper that this research has extended off of.[Finnicum and King 2011] There are minor variations to this model, but it needs to be restated nonetheless.

3.1 Privileges

When using applications in a current app store, whether it be on android, iphone, or even facebook, one is given a list of permissions that the application is going to need in order to run. The user can choose a default set that they will allow applications to run, but they occasionally will need to make extra choices. While in theory, this could be extended to a robot, the amount of permissions that would be required in order to keep the framework secure would be impractical, and an every day user would not want to use the robot.

For example, if there were an application to see your professor, the user would have to choose if the application is:

-able to use the camera

-able to drive around

-able to have the location of your professor's office

-able to possibly use the speakers

-able to use the GPS

So while this is something that a user could possibly go through, the variety of capabilities a robot has would make this daunting for a normal user. This shows that this kind of abstraction is not practical for a general purpose robot. Our paper has to consider different kinds of low level abstractions that allow for priviliges without having to create too many options for the user.

3.2 Acceptable mistakes

The framework may occasionally allow for mistakes in privacy security, but they should generally be small ones. For example, the robot will not run around broadcasting your location for the past year to any robot within range, but it may accidentally overestimate someones values and tell them where you currently are if they request the information. This is not just an issue with a robot, but an issue you will find in humans as well, so we deemed these to be acceptable mistakes. These are not defined in the security model, but will be more thoroughly researched in future work in security.

4 Experimental Design

In our paper, we used a robot that was created using commercial parts. This allows our experiments to be easily replicated, since there were no special designs required in order to build the robot. We used an irobot for the base, a netbook for our robot systems, and a kinect for the camera.

For our software, we used the middleware Robot Operating System (ROS) to provide the functionality for the robot. ROS uses a publisher-subscriber system that allowed us to create a variety of nodes and services to develop our framework and applications.

The way that we designed applications in ROS, was to set them as individual nodes, as seen in the figure below which could either subscribe or publish messages, depending on the functionality of the application. We abstracted all of our robot's functionalities to services in the underlying framework. This allowed us to define a set of capabilities that our robot had. It also allowed us to easily extend our framework to additional functionalities, since all that would be needed, would be to add an additional services to the list.

When a node would need access to any of the functionalities defined in the services list, they would first need to make sure that they had the appropriate permissions to use them. In order to implement this feature we created a Permissions class which provided the robot with a list of the services and decided wheter or not the app had the

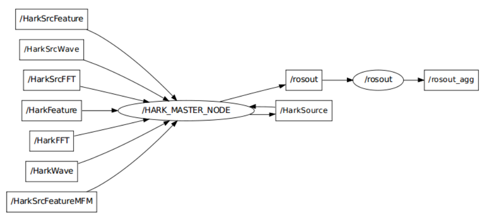

Figure 2: *Figure 2: Pictured above is a graph of an example publisher-subscriber system showing the topics that publish pointing the topics that are subscribing*

appropriate qualifications for the service. If the app failed this test, then it raised an exception that allowed the application to know that it was not allowed to access that section of the robot.

The permissions system first looked at the applications id, which is initiated when the app first starts up, and is then stored into a database. The permissions then decides whether the application has the ability to run these apps based off of the user defined settings of what they want to allow.

If the application has the appropriate permissions to continue, the app is then registered into the database, and initializes any properties it needs whether it needs to start subscribing to certain topics, or publishing messages of its own to control different functionalities of the robot.

While desigining our framework, we kept into consideration that the future may have different capabilities then we currently had on our robot. Additionally, there may be other models of robots produced that don't have certain features that we do. Because of this, we had to ensure that the framework we designed was adaptable and extendable to many different kinds of robots. This gives developers the flexibility to choose what kind of functions their robot needs, without having to rewrite a security framework each time they want to implement something new.

4.1 Secure Data Storage

In order to provide appropriate security in message handling, we abstracted the messages to a backend API. This backend API stored any information that the robot needed into a database that would pass only IDs through the messages. This prevents applications from being able to look up details (i.e. precise locations or details about different users in the database), which provides privacy to the user and any location that they store.

4.2 Robot API

Handling messages in a publisher-subscriber system would be difficult for a developer, because they would have to know the specific names of different topics, message types, and how to write callback functions appropriately. This would alienate all but more experienced programmers from developing applications for our robot, thus defeating the purpose of making it a general purpose framework.

In order to compensate for this, we wrote an API that handled a lot of the messages for the developer, and made it a lot simpler to program with only a small amount of object oriented programming abilities needed.

In order to write the API, we first created functions that allowed the user to publish and subscribe to the different functionalities of the robot. We made these class methods, so that they user would not have to initiate a class for each of the different services that they wanted to use.

After this we created other functions that spoke to our backend object model, which stored and retrieved different things in a database. For this paper, we only had 2 different tables, which stored different users and locations.

For application writing, we created an empty class which the user could extend in order to create their application. This included an on_startup() function, which the user is forced to overwrite, which allowed them to define what initialization functions would be required when the robot starts up in order for it to work.

The empty class also included a run() function, which the user would call after they initialized the class. The reason we wrote it like this is because of the permissions system. When the class is initialized it checks the permissions and if it isn't passed it raises an exception. If it does pass the object is created, and the developer can then use run() to launch the application.

We also created an optional on_shutdown() which allowed the user to provide any code to handle the app closing. This could involve clearing any data that the app created, among other needed things.

4.3 Task Priority

Our robot implemented a task priority system in it, that allowed it to choose what kind of tasks were more important than the other. Because of the publisher-subscriber system in our robot, when a node published a message to a topic, any node subscribed to that topic would receive that message. Deciding which nodes had priority to take control of the robots features over the others was a case we had to consider.

We chose to design a middle node, which we considered the concierge node. This would intercept any message passed and would assign priorities to the different capabilities of our robot. It then would choose which node was allowed to have access to it first. The concierge was an implementation that is intended to be easy to extend to future applications in which there are different capabilities that our robot might have. All that a developer would need to do, would be to add the capability to the list, or remove any that their robot did not have access to.

4.4 Logging System

In our threat model, we defined that our robot's system would have a tolerance of acceptable mistakes. This would include giving a person a little bit too much trust and telling them something that the user might not have wanted them to know. We claimed that this was acceptable for minor mistakes, because its a level of detail that even humans would make. We had to account for what kind of acceptable mistakes the robot might make, and have a way of showing why the robot made these mistakes. In order to do that we implemented a logging system into our robot.

The logging system was a way so that a user could trace any mistakes that the robot made, and find out why the robot did that. Was it a case of a vulnerability of the system? Or was it just the assumed reason that the robot gave too much trust to another human. Our logging system is relatively simple, but is extremely powerful in the fact that it can help debug any acceptable mistakes made.

The logging system is handled by the robot logging any message sent through the system. Recall that the robot is run by a publisher-subscriber model. This allows for the logging system to take any message that is published and store it in the hardware, it can also grab which nodes had subscribed to it, which allows one to see what kind of things were happening around the time that the robot made the mistake.

Other things that the logging system kept track of were the video. The robot is able to take in the images that it sees, which allows it to recall them if needed. Our current robot had the capabilities of storing 30 images in a second, as that is the rate of the video feed that the robot is outputting. Another similar thing that it is able to log is any voice command given to it. It would store these files in an ogg format, which is a high bitrate audio format that has significantly more compression than a wav.

4.5 Simple Example App

This application is a simple example that we created to test the features of the API. It creates a new user object, and goes to its location. It then says “Hey, Sam”, 3 times through the speaker, and drives back to its starting location.

As seen by the figure below, it is simple to read for someone who understands object oriented programming, and develop code of that nature, and the application never has access to any of the user information or the locations that it is at, as it is all stored as IDs in the database.

5 Testing

In order to ensure that our framework was working correctly, we developed a test application that used all of the services that our robot had the capabilities to perform. This included the use of the microphone, driving, speakers, and camera.

The application that we developed was a bartending application. We will discuss how it worked stepped by step through the application creation and implementation process.

When the app is first initialized, the robot first checks to see if it has the correct permissions to use all of the functions. If it did, the app was able to run, if not then the app through an exception, and it was not ran.

The next step was for the application to find new users. It did this by driving around and identifying the users to see if they were found in the database. In order to provide user identification we used our previously created CLASS system, which while not currently accurate for large sample sizes, we believed that in the future when the robot is able to be produced for the public, that the identification would be able to be more precise.

When it found a new user, the robot would announce through its speakers asking the user to make a choice on the drink that he wanted. (In our demo we had it ask from a few different kinds of sodas)

The user could then say what kind of soda he wanted, and the robot would store it in a class array that the app had created. After finding enough users, or after a defined duration of time, the robot then headed back to the bar to give the bartender the orders.

The robot would announce all of the orders through the speakers to the bartender, after which it would wait until the bartender placed the drinks on top of the robot. The robot would then drive back to the users, and deliver them the appropriate drinks.

While the application was successfully ran in a small group setting, it would be difficult to have it run in a large group setting, since the user identification would be a difficult task, but since this is just a problem of technology not being advanced enough, we decided

```
#!/usr/bin/env python

import roslib; roslib.load_manifest('annoy_sam')
import rospy

from ObjectModel import Location, User, Speaker

from Permission import Permission
from Permissions import Permissions

from FrakApp import FrakApp

class AnnoySam(FrakApp):
    app_name = "Annoy Sam"
    required_permissions =
    [Permissions.SPEAKERS.num,
    Permissions.MICROPHONE.num,
    Permissions.DRIVE.num]

    def on_startup(self):
        self.annoy_sam()

    def annoy_sam(self):
        sam = User.lookup_by_name("Sam")
        start = Location.cur_pos()

        sam.office.drive_to()

        while not sam.office.near(1.0):
            rospy.sleep(1)

        Speaker.speak(
        "Hey Sam! Hey Sam! Hey Sam!")

        start.drive_to()

        while not start.near(1.0):
            rospy.sleep(1)

        Speaker.speak("I'm back!")

        return

if __name__ == '__main__':
    try:
        app = AnnoySam()
        app.run()
    except rospy.ROSInterruptException: pass
```

that we would factor it out of consideration for our evidence of a working framework.

During the entire process, the application was never able to pull any specific details about who the user was or any of the locations it travelled, while it still was able to succesfully accomplish its goals. This is a good thing because it did not invade the privacy of any of us who did not give it permission to access our information, but it was still able to interact with us nonetheless. Overall, we believe that the test application was a success. It was able to function in a manner that was not against what we wanted the robot to do, and no external information was viewable while running the application.

6 Discussion

Our experiment allowed us to create a test application on a robot built on commercial parts that took advantage of the different systems that our robot had access to. It ensured that each of the different functionalities was secure in that a user could not develop anything that would easily break any of the defined issues in our threat model.

In designing the framework, and throughout the paper, we made a few assumptions, that while seemingly large, we believed to in fact be reasonable. We assumed that in the future technology would be more advanced. While we do not know the exact state of technology in society when these robots are produced, we can assume that they will be significantly larger than our current state. This would allow them to execute tasks in a faster manner, while also being able to store significantly more information.

One important thing we had to keep in mind while designing the framework was that each robot may have different functionalities and capabilities, and even more possibly, they may have a different underlying middleware to control their robots functionalities. Because of this, we wanted to create a framework that was extendable to additional capabilities, and flexible enough to account for different middleware systems. In order to do this, we had a level of abstraction in our API that allowed the user to develop for the robot, using general terms for the actions that they wanted them to perform. For example, instead of manually handling messages between topics, the user would only have to call a function such as Drive.driveTo(). This would robot producers to rewrite the underlying functions with however tehy want the robot to communicate between applications, while still being able to be ensured that the framework would be successful. This is different from other programming languages that have been written for robots because of the adaptability of our system.

Security was an important concern in the design of the framework for the robot, as it is an area where little research has been done before. We handled security in a simple way, that allowed the robot to be adaptable in what kind of permissions it allowed applications to have on the robot. The permissions system was easily flexible to different capabilities in robots, and allowed developers to handle them in an easy to define way.

We also allowed for developers to be able to easily create applications for our robot. Because we were designing the framework for what would be a general purpose robot with an application store, we would need to ensure that it was easy to develop the applications for it, unlike, say a language like prolog, where you would need a lot of understanding to be able to write a message. Our robot was able to be programmed in python, and we had developed an API for it that allowed a developer to not have to understand the architecture of the robot in order to develop applications for it.

6.1 Social Network

There were a few assumptions that were made when designing the framework. This had to include considerations for the future and different technological capabilities that are likely to happen but not currently in place in society. The first of these is a social network system that may be included into the majority of robots.

The internet is available on a variety of applications, which makes it easier for objects to communicate with each other. This lets us discuss the ability of having a social network with the robot. Some people have different levels of privacy than others, so we need to ensure that our robot will account for those individual levels of privacy.

A professor in a university is a good example of a case of social privacy. They need to be more open then the average person, since there are other professors, students, and even their friends/family that may wish to interact with their robot in some way. Even if the professor thinks out a list of all the possible people who may want to interact with his robot, there may be other appropriate examples that do not fit into this list, such as a professor from another school for instance.

This shows that the social network idea of allowing the user to set groups for themselves to choose who can and cannot interact with their robot is not a good use. Instead, the robot framework will use a feedback and learning mechanism to decide whether or not the person is appropriate to interact with it. For instance, you can have a set of predefined relationships such as sibling, friend, and friend of friend. The robot can the choose how appropriate it is for a person in one of these fields to talk to it, as well as extending it to adjacent scenarios.

Privacy will also need to be in both directions. As an example with the professor, if their robot was kept in their office and a user came to see if the professor was in his office, the robot would not want to give the information to someone that is completely untrusted, but on the other side, the user might not want the robot to log all the information. Not many users would want their professors robot to say that they came to see if they were there 30 different times.

6.2 Secure Identification

When identifying users, an image is needed to check to see if the user is the one that the robot is looking for. This causes a privacy concern in the way the identification system works. If the robot is storing too much information about each of the users, then it is acting too much like a surveillance robot, and can not ethically be allowed to run in public, where you could locate someone who does not want to give you access to know that they are there. This could lead to someone creating apps that indirectly stalked people, which is a huge privacy violation.

7 Conclusions and Future Work

The robotic framework that we have developed for this project is something that has much needed future development. There are a lot of things that could not be tested because of either lack of time or technology. I believe that by developing the framework now, that it will prevent us from having problems in the future that might otherwise arise from lack of securities in extended capabilities from robotics development.

A lot of the future work should be done in the field of general robotics development, in allowing an easy to program robot system as opposed to some of the more complicated languages that are available today. Other systems required for robots are also important, such as better visioning techniques in identifying users. The future of personal robotics is not that far away, so the sooner research is done in this field, the safer we can ensure the environment is for allowing robots.

Acknowledgments

I'd like to thank Murph Finnicum (PhD Student), Aleksandar Kravchenko (MS Student), and Joe Leong (MS Student) for their help in developing the robot framework, and Sam King for providing with the materials needed to create the project. I'd also like to thank Tarek Abdelzaher for allowing us to use his lab space for the robots.

References

FINNICUM, M., AND KING, S. 2011. Building secure robot applications. *Usenix Workshop on Hot Topics in Security*.

LORENZ MOSENLECHNER, NIKOLAUS DEMMEL, M. B. 2010. Becoming action-aware through reasoning about logged plan execution traces. *IEEE Internation Conference on Intellegent Robots and Systems*.

MICHAEL BEETZ, ULRICH KLANK, I. K. A. M. L. M. D. P. T. R., AND TENORTH, M. 2011. Robotic roomates making pancakes. *Intellegent Autonomous Systems*.