

PROJECT PROPOSAL

FORMULA SAE PROPOSAL

A proposal for a cybersecurity focused Formula SAE project

Master of Science
in
Cybersecurity

Submitted By,

Daniele Giachetto

647820

Department of Information Engineering

Edoardo

694202

Something

Under the guidance of,

STEFANO CHESSA

PROFESSOR & CHAIR OF THE M.SC. IN CYBERSECURITY,
COMPUTER SCIENCE



DEPARTMENT OF INFORMATION ENGINEERING

Università degli studi di Pisa

Accademic Year: 2022-2023

Contents

1 The project	2
1.1 Project scope and goals	2
1.2 Driverless code structure	2
1.3 Issues found	2
1.4 Objectives	2
1.4.1 Developer guidelines	2
1.4.2 Build environment	3
1.4.3 Tests	3
1.4.4 CI/CD	3

THE PROJECT

1.1 PROJECT SCOPE AND GOALS

The current objective of the team is to develop an autonomous vehicle. This project is also called “Driverless” and needs to meet high standards of reliability and security in order for the car to compete and the project to be deemed a success. This directly lead to this project scope and goal: allow the team to pursue the goal to increase the security of the codebase, mainly targeting development, test and build phases of the software.

1.2 DRIVERLESS CODE STRUCTURE

Before delving in the issues and solutions, we need to look at the code structure to better comprehend the many problems that the team is facing. The main programming languages used are C, C++ known for many vulnerabilities linked with memory issues and **Python** known for vulnerabilities linked with duck typing. Changing languages is not an easy tasks mainly because developers are not proficient with other programming languages and libraries and framework used are only available in those three programming languages.

1.3 ISSUES FOUND

- **Lack of documentation:** results in fragmented codebase and obscure code;
- **Lack of shared code style:** results in high amount of bugs, security issues and bad practices;
- **Different development environments:** every developer has a different environment, without standardization and a clear guide;
- **Lack of tests:** there are no tests, and everything is tested manually;
- **Lack of versioning:** the code is versioned, but the releases are not. This leads to bugs and security hazard when different components interact;

1.4 OBJECTIVES

1.4.1 DEVELOPER GUIDELINES

Documentation is the first step to help a developer ensure high quality code. Those guidelines will contain installation guides and visualization of the “Driverless” project structure.

1.4.2 BUILD ENVIRONMENT

The “Driverless” project builds are complicated to say the least. Every developer needs to install in its own machine the ROS framework and all of the dependencies that the current project has. Right now this leads to builds effected by clashing libraries versions and in the production environment there could be libraries missing or with wrong versions.

To solve this problem we need to create an enviroment containing all the libraries required, that can be easily installed, replicated and in which development and builds can be done without issues. Further research on the subject is required before deciding the technology to be used.

1.4.3 TESTS

Currently, all the tests are done by running the code in the production environment and manually checking the car behaviour. This process is dangerous, inefficient, and behaviours cannot be reproduced consistently. It could lead to security issues regarding the production environment, human errors and more commonly edge cases not tested. To reduce the vulnerabilities that can be found in the code we aim at introducing three different test methodologies:

- **Unit testing:** testing the smallest component that can be isolated, usually only the body of a function and mocking the external function calls inside of it;
- **Integration testing:** testing a functionality, from the input to the output mocking only calls to external software (like build dependencies);
- **End to end testing:** testing the complete flow of the program, checking if all the components can work together.

1.4.4 CI/CD

Planning a well-tought out continous integration and delivery pipeline will be beneficial to solve all of the previously discussed issues. Currently those are the phases that will be executed:

- **Linting:** used to enforce rules on the codebase. It will be used to statically analyze the code, avoid bugs and possibly introduce rigid static type checkers to scripting languages;
- **Code formatting:** used to automatically format the code following pre-established rules (ex: line length);
- **Test:** unit tests, integration tests and end to end tests will be automatically executed to ensure code quality;
- **Build:** using the standardized build environment, we can automate the building process. Doing so will also allow us to introduce versioning to the releases.