IoT Security Validation

DESIGN DOCUMENT

sdmay21-41

Megan Ryan
Kristin Rozier

KayAnne Bryant
Jacob Conn
Joshua French
Vincent Johnson
Jordan McKillip
Marcus Reecy

sdmay21-41@iastate.edu
https://sdmay21-41.sd.ece.iastate.edu/team.html

Revised: 11-15-2020/0.02

Executive Summary

Development Standards & Practices Used

Software: SV-COMP, Ubuntu 2004, ABET

Waterfall methodology

Summary of Requirements

Ubuntu 2004 VM machine/x86_64-linux to run and test the programs. A memory limit of 15 GB (14.6 GiB) of RAM, a runtime limit of 15 min of CPU time, and a limit to 8 processing units of a CPU conforming to standard SV-COMP guidelines.

Applicable Courses from Iowa State University Curriculum

SE/CPrE 185, COMS227, COMS228, COMS311, SE339, SE329

New Skills/Knowledge acquired that was not taught in courses

Through this project, we learned about some potential security vulnerabilities, SV-COMP, and Linear Temporal Logic formulas.

Table of Contents

List of figures/tables/symbols/definitions (This should be the similar to the project plan)

Library - A set of prebuilt code that allows a developer to call or use some functionality of it. The developer chooses when and where to use the library.

Framework - A set of prebuilt code that allows a developer to add some functionality in preset locations. A framework controls the flow of code and a developer may edit some portions of the framework.

Verification Run - a non-interactive execution of a candidate on a single verification task, in order to check if the following statement is correct: *"The program satisfies the specification."*

Linear Temporal Logic Formula - a modal **temporal logic** with modalities referring to time. In LTL, one can encode **formulae** about the future of paths, e.g., a condition will eventually be true, a condition will be true until another fact becomes true, etc.

Verification Task - consists of a C or Java program and a specification, security property.

Property - a specification to be verified for a program.

IoT Code - any code that can be used to build an IoT device or has functionality that may assist in building an IoT device.

1 Introduction

1.1 Acknowledgement

Our project's purpose is to expand SV-COMP and utilize the repository and tools provided by the non-profit organization ETAPS. We would also like to thank Muhamed Stilic for the work originally completed within developing the benchmarks in C.

1.2 Problem and Project Statement

The Internet of Things (IoT) is becoming more and more a part of people's everyday lives. Devices such as locks, cameras, and smart-speakers are just a very small view of all the ways our lives are going online. With all of these devices having important roles, being located in private places, and gathering loads of information, the security of them is much more prevalent as it would be problematic if it got into the wrong hands.

There are already some ways that the security of the code behind these IoT devices is being tested. However, there are a lot of security properties that aren't being as thoroughly checked. One of the ways is through a program called SV-COMP. SV-COMP is a security validation software that helps compare different software verification tools to help find which tools will suitably satisfy your needs. These tools test a variety of common security failings to ensure that the software can be validated and secure for any developing needs regarding the compatible c softwares. Our project is developing on SV-COMP and expanding it to focus on IoT device code and test different IoT libraries.

The final goal of this project is to have a working version of SV-COMP that is able to test many IoT libraries, and from that be able to confidently verify the security of different IoT libraries. In doing that, we will have IoT code benchmarks for others to use to secure code with their own validation tasks. A further goal, if time and resources persist, is to combine the secure code we find into an IoT library that is trustworthy and reliable. Within the course of this project, we are to modify the existing framework of SV-COMP so that it can test both java and c IoT libraries, a program that can be downloaded and used by any anonymous developers.

1.3 Operational Environment

Ubuntu 2004 VM machine/x86_64-linux, a memory limit of 15 GB (14.6 GiB) of RAM, a runtime limit of 15 min of CPU time, and a limit to 8 processing units of a CPU.

1.4 Requirements

The software can be downloaded, it can be replicated and evaluated.

The software can be archived in a ZIP file, with a directory within.

The software should not require any special software on the competition machines; all necessary libraries and external tools should be contained in the archive.

The software can report its version.

Remains free of unnecessary data, with only the core code and descriptions within the code, free of things such as test files.

The software can be within java or C programming languages.

1.5 Intended Users and Uses

The intended users of this software will be developers and academia of IoT.

End users will be able to use our developed benchmarks to test their own IoT code through validation tasks in the SV-COMP environment.

1.6 Assumptions and Limitations

Assumptions:

- Only using Java and C IoT code
- BenchExec will run smoothly on a different Ubuntu version
- Verification tasks will run correctly on BenchExec

Limitations:

- The monetary cost to produce the end result shall be zero
- VMs are limited to one core on 8 GB of RAM running Ubuntu
- We are limited to the end of spring semester to finish the project

1.7 Expected End Product and Deliverables

List of several well rounded IoT libraries

> These IoT libraries will all come from open source code on the internet. A well rounded library will be one that does not have a profusion of dependencies in it. Some of these libraries may be slightly modified to make them into well rounded libraries. This list will then be run through IoT verification tasks to see how secure they are.

A running instance of SV-COMP

> This instance of SV-COMP will be set up on VMs provided by the university. The instance of SV-COMP will be able to run the IoT verification tasks on the open source IoT libraries.

Verification tasks to run with C and Java IoT code

> This set of verification tasks will be focused around the security aspect of IoT devices. Ideally, this will include both the C and Java languages. Part of these may come from the SV-COMP benchmarks repository, whereas others will have to be written on our own. These tasks will cover a slew of security issues with IoT devices, and will not focus on any one particular aspect/weakness.

C and Java library consisting of code that has been verified using IoT verification tasks

> This deliverable is a bonus one that we would like to do if time allows it. After testing all of the open source IoT code with the verification tasks by using SV-COMP, the good code will

then be compiled into a library. This will result in separate C and Java libraries of secure IoT code.

2 Project Plan

2.1 Task Decomposition

Below is a list of our planned tasks, and some of the steps required within each task to complete it. Some tasks require specific steps, whereas others require more open-ended research and data collection. Most tasks build upon the knowledge found in the first task of general research.

- General Research
  - Team introductions
  - Vulnerabilities
  - SV-COMP
  - IoT code
  - Code libraries versus frameworks
  - Code verification tools
- Identify Milestones
  - Discuss with advisor and client
  - Have a clear end-goal decided on
  - Develop a Gantt chart
  - Verify milestones with advisor and client
- Identify IoT Libraries for use - the team will be testing many libraries throughout the project
  - Create library benchmarks - based on data found in general research
  - Create IoT code benchmarks - based on data found in general research
  - Research available IoT libraries based on determined benchmarks
  - Choose a select amount of libraries for use
- Identify verification properties to test
  - Research which properties are tested often - knowledge of vulnerabilities from general research used
  - Choose properties not tested as often - partially based on IoT libraries decided upon
- Set up SV-COMP
  - Get access to an ISU virtual machine (each team member)
  - Decide on SV-COMP tools to use for Java - based on knowledge from general research
  - Set up SV-COMP on virtual machines - uses knowledge from general research
- Design Java Verification Tasks
  - Use example verification tasks for guidance
  - Use knowledge of decided upon SV-COMP tools
  - Use knowledge of SV-COMP
  - Create based upon decided security properties to test
- Design C Verification Tasks
  - Use knowledge from created Java verification tasks
- Run Verification Tasks
  - Utilize SV-COMP

    - Plug in IoT libraries previously decided upon
    - Verify libraries based on previously chosen security properties
- Build a Validated IoT Library
    - Using validated IoT Code from our tested benchmarks.

2.2 Risks And Risk Management/Mitigation

For our project we have identified the following risks and risk mitigation plans:

Licensing Issues - a few IoT platforms require capital to use. Our mitigation strategy for this is to use open source IoT code and libraries.

Inadequate Design - a risk associated with a misunderstanding of the project's goals. Our mitigation strategy is to define our project problem and our statement and use those definitions to expand on our design of our project.

Team Dynamics - a risk with any project that has a team. Our mitigation strategy is to have weekly team meetings, address issues as they arise, and be proactive.

Developing wrong software functions - this risk can come from miscommunication or misunderstanding of the project requirements. A mitigation strategy for this risk is to have code peer reviewed and have weekly team meetings to go over functions that are required for our task.

Gold Plating - adding more features to a product that the client did not ask for. Our mitigation strategy for this is to keep within our project plan and use our weekly client meetings to stay within the scope.

Incompatible Libraries - we may run into a library that requires too much time to 'round' off. One way to mitigate this is to look for standard libraries in addition to checking libraries versus library definition.

Incompatible IoT Code - code that is heavily library dependent requires too much time to properly 'round' off to run as a validation task. Our mitigation strategy is to look for good IoT code commonly used in IoT and use that as a comparison tool with other IoT code to validate.

Time Constraints -as we run these verification runs our system will have to do model checking. This takes real time and since we have limited ram it may take a full day or more to run a verification run. Our mitigation strategy for this is to set soft limits for verification runs. For example, a normal run would be considered <24hrs, but anything >24hrs we will consider as an unknown failure.

2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

There are currently seven milestones for this project, and each of them fall under a specific deliverable:

The first milestone falls within the deliverable, "Project Research", and is described as being met once we have sufficiently researched key materials related to our project abstract. The metric is to have all team members aware and understanding the required information needed to advanced to the next milestone.

The second milestone is met once the team has identified both the milestones and timelines of the project. This falls under the "Prepare Project Plan" deliverable. The metric will be designated by a proper timeline given and agreed upon by the project team so that other milestones can be properly accomplished within the required timeframe.

The third milestone, within the "Set-up SV-COMP" deliverable, is met once the team builds the SV-COMP environment, and is able to run verification tasks composed of an IoT library and a chosen security property. The metric is when more than half of team members have created a sustainable environment and successfully ran SV-COMP within their virtual workspace. Once that has been confirmed by other members of the project team will they proceed to the next milestone.

The fourth milestone is achieved once the team has successfully completed designs for verification tasks that are composed of C IoT code and a chosen security property. The metric is to half several designated security compromises and create corresponding verification tasks within C depending on the requirements of said IoT compromises.

The fifth milestone is the same as the fourth except that it will be Java IoT code with a chosen security property. The metric is to half several designated security compromises and create corresponding verification tasks within java depending on the requirements of said IoT compromises.

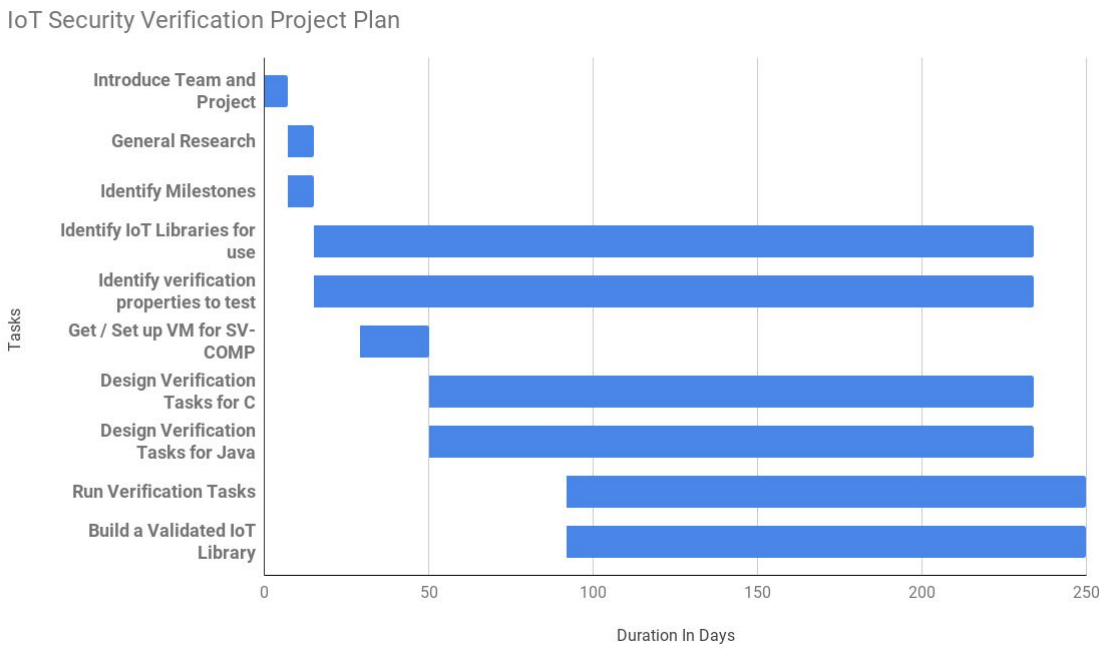
The sixth milestone combines both the C and Java verification tasks completed in the previous two milestones and is used to test common IoT libraries the team finds. The metric will be to have a set of working, sustainable verification tasks based on the framework of SV-COMP in order to

Finally, the seventh milestone will be completed once we compile all of the benchmarks made from the previous milestone and construct a verifiably secure IoT library. The metric will be the creation of the IoT library where each of the verification tasks are able to run consistently and successfully.

The first two milestones are evaluated by all of the team members, and are considered completed once we are satisfied with the work put into them. The remaining five milestones will also be evaluated by all team members, but will rely more heavily on the functionality of our designs.

2.4 Project Timeline/Schedule

Our Gantt chart is still a work in progress as we continue to discover new requirements and difficulties. This being said, our schedule plans long periods of time for the tasks that we think may uncover new difficulties and will take the longest. These long periods of time will allow us to break them down into smaller tasks. This way the project end date will not be delayed.



Introduce Team and Project (8/24/2020 - 8/31-2020) - In this section we introduced ourselves to our team, client, and advisor. We also were given a summary of what our project would consist of. As a team, we decided when meetings would be and set internal roles and responsibilities.

General Research (8/31/2020 - 9/8/2020) - We were assigned research topics by our advisor to understand our project in a further manner. We created slides containing our research and additional questions we had. These questions turned into further research.

Identify Milestones (8/31/2020 - 9/8/2020) - After being given an overview of the project we brainstormed general milestones that our team could use to guide our project. Though we came up with a set of milestones, we all agreed that this would continue when new goals arose.

Identify IoT Libraries for use (9/8/2020 - 4/15/2021) - One major portion of our project is to identify IoT libraries that we may use for verification tasks. This milestone started by identifying criteria that we may consider before picking a library. It continues to 4/15/2020 because we will continue to pick and choose new libraries as we continually design verification tasks.

Identify verification properties to test (9/8/2020 - 4/15/2021) - Verification properties are the properties that we test in a library to determine if it meets certain criteria (in our case, security). Verification properties will continually be edited as we consider new libraries. Because we are also considering new libraries until 4/15/2021, we must also identify new verification properties until this same time.

Get / Set up SV-COMP (9/22/2020 - 10/13/2020) - Our team must request and set up virtual machines such that SV-COMP and other testing tools may run. This time period is 3 weeks because we must wait on the ISU IT department to give us access to these VMs.

Design Verification Tasks for Java (10/13/2020 - 4/15/2021) - Our team must use the chosen libraries to design verification tasks to run in SV-COMP. These verification tasks will be written on libraries specifically in Java. As we pick new libraries we will also need to write new verification

tasks for these libraries. Since we will be considering libraries until 4/15/2021, we must also write new verification tasks until this point.

Design Verification Tasks for C(10/13/2020 - 4/15/2021) - Our team must use the chosen libraries to design verification tasks to run in SV-COMP. These verification tasks will be written on libraries specifically in C. As we pick new libraries we will also need to write new verification tasks for these libraries. Since we will be considering libraries until 4/15/2021, we must also write new verification tasks until this point.

Run Verification Tasks (11/24/2020 - 5/1/2021) - As our team writes verification tasks we will continually be running and testing them to make sure of completion. This process has a potential to take a long time and thus takes up most of our allotted schedule. After 4/15/2021 (When no new libraries will be chosen by our team), we will continually develop with the libraries that we have in our hands at that time. By 5/1/2021 we wish to have all verification tasks for the libraries in a complete state.

Build a Validated IoT Library (11/24/2020 - 5/1/2021) - Validated IoT Code from our benchmarks will then be compiled into a validated IoT library for future users to utilize.

2.5 Project Tracking Procedures

The group has decided to utilize a waterfall method and a gantt chart to track the progress of the project in its various stages. The Waterfall model will be used to make sure that all members are aware of what parts of the project are being worked on at what time. Should additional information force the group to focus on one specific project within the project, additional time will be allocated within the waterfall model so that everything is kept on track for our deliverable due date.

The Gantt chart will give a detailed breakdown of what parts of the project have been completed, and how much longer the group must work on that particular area of the project. The chart will be updated during every meeting, and will be an accurate depiction of the progress that has been made.

Messages between members will be communicated through Slack. Any vocal calls will be done through either Discord or Zoom at designated meeting times. Code and all progress can be evaluated and checked within a shared Git repository.

2.6 Personnel Effort Requirements

Task	Person Hours
General Research (8/31/2020 - 9/8/2020)	200-300
Identify IoT Libraries for use (9/8/2020 - 4/15/2021)	60-100
Identify verification properties to test (9/8/2020 - 4/15/2021)	60-100
Get / Set up SV-COMP (9/22/2020 - 10/13/2020)	250-400
Design Verification Tasks for Java (10/13/2020 - 4/15/2021)	500+
Design Verification Tasks for C(10/13/2020 - 4/15/2021)	500+
Run Verification Tasks (11/24/2020 - 5/1/2021)	400+

2.7 Other Resource Requirements

We will be using virtual machines provided by the university to use SV-COMP to build and run our verification tasks.

2.8 Financial Requirements

All code and virtual machines are provided via the university, and all code is personally made or publically available. As such, currently, the project requires no financial support in order to proceed.

3 Design

3.1 Previous Work And Literature

CWE [1]
CWE is a website that contains all of the most common software weaknesses and vulnerabilities. Each year the list is updated with a new set of common weaknesses. Our project will involve taking some of these weaknesses and determining if a new security property should be developed.

SV-COMP [2]
SV-COMP is a competition whose goal is to publicly efficiently test and verify software. SV-COMP works to create and maintain a set of programs and security properties. These properties are made publicly available for researchers and developers. Our project revolves around using these properties (and making new ones) in order to build a list of security tested libraries.

3.2 Design Thinking

There are a few aspects that shaped our design:

Find libraries and/or code to implement into verification tasks. Our team needs to put together a list of IoT libraries that we are able to run verification tasks on. A library is defined as a set of prebuilt code that allows a developer to call or use some functionality of. A library is different from a framework. A framework is defined as prebuilt code that controls the flow of the program. A library differs from this such that a developer can use it whenever they choose. Iot code is defined as any code that can be used to build an IoT device or has functionality that may assist in building an IoT application.

After considering these design aspects we were able to make a few more design choices that came up during the 'ideate' phase:

Find IoT related security properties. Our team needs to expand on current properties listed in SV-COMP for both C and Java that relate to both security and IoT.

Create Verification tasks. Our team needs to create and expand on current verification tasks listed in SV-COMP. We may use the following definition for a verification task: A set of C or Java programs and a specification, a security property.

What languages are our libraries in? The IoT libraries that we will be focusing on are in C and Java. This decision was made because SV-COMP only has categories for these two languages. In the future, as SV-COMP expands, we may also expand our language selection.

What security properties will we cover? There are many security properties that are already listed in SV-COMP. Our team will look at the Common Weakness Enumeration (CWE) website and compare the weaknesses listed there that might relate to security. In the end, we will have a comprehensive list of properties that encompass SV-COMP and CWE.

3.3 Proposed Design

Through our research in the SV-COMP repository and its property files used in validation runs for BenchExec. We have discovered that we can use SV-COMP and its tools provided to test properties that align with our chosen security standards.

Our proposed design is as follows:

SV-COMP is mainly used to validate tools against predefined programs and predefined properties and gives each tool a score. With the information a tools score for a particular property, we can leverage SV-COMP and the chosen tool with the highest score for a property to validate IoT Code.

C IoT:
We can build a validation task, within SV-COMP which consists of a property and a program. The property of the validation task is a defined Linear Temporal Formula. Whereas the program is the code that will be validated against the property. SV-COMP already has pre-defined properties for C those of which, valid-memsafety, valid-memcleanup, no-overflow we chose to be security related. Once a validation task has been compiled the BenchExec, a runnable, will then execute the validation task in a validation run. The validation run will then give a result of "True", "False", or "Unknown".

Java IoT:
Validating Java IoT with SV-COMP will be similar to the above with C, however because SV-COMP only has two predefined property files we will have to include our own. Currently we have defined input_validation, integer_overflow, improper_authentication has the security properties we wish to validate IoT code against. This will include creating our own LTL formulas, implementing them into our property files and then modifying some of the wrappers in the tools to be able to successfully run validation runs.

Through preliminary testing we have found that it is possible to implement IoT code, in this case a DIY Thermostat program, into a verification task and run with a security-related property by the BenchExec.

Through this preliminary test, we were able to meet the following requirements:

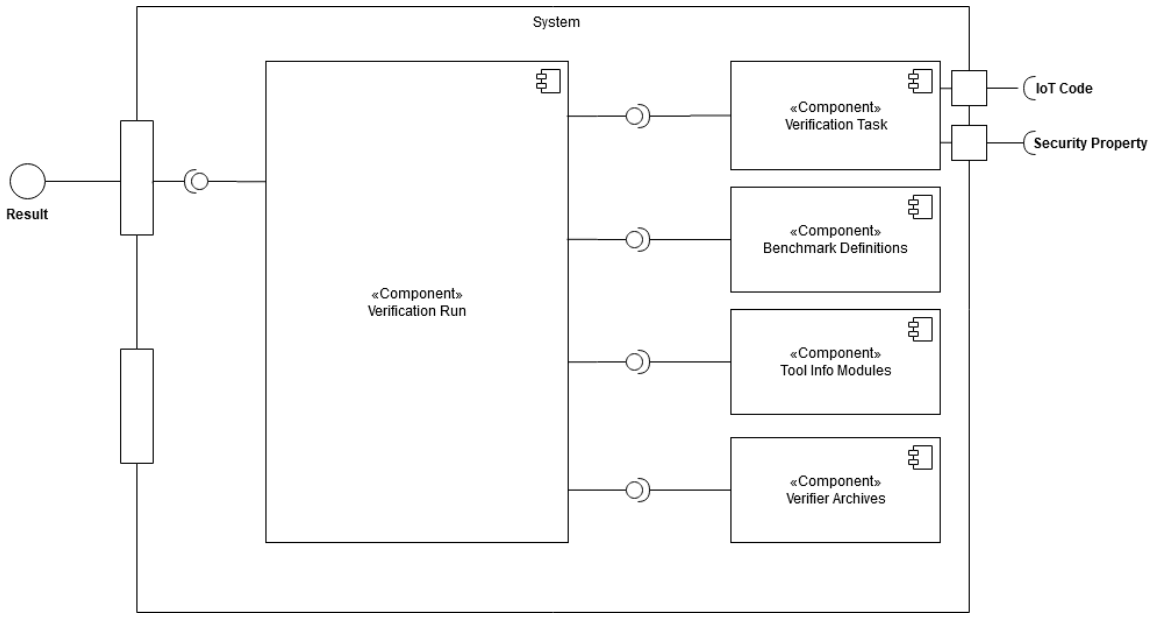
- Report validation result of the security property (Functional)
- Report its version (Functional)
- Can be downloaded, replicated and validated (Non-Functional)
- File Package contained all libraries and tools (Non-Functional)
- File Package contained no unnecessary data, code, etc (Non-Functional)
- Was written in C

In addition, our proposed design will need to meet the following IEEE standards:

- IEEE 802.11ai-2016 - initial setup methods and their security.
- IEEE 1012-2016 - verification and validation of systems, software, and hardware life cycles
- IEEE P2933 - creating a framework for IoT data and device interoperability in the clinical region that incorporates the values of TIPPSS

These standards will need to be met when we are modifying IoT Code and/or selecting IoT libraries to test.

Furthermore, based on the above Design Plan we designed the following UML diagram for our design.



3.4 Technology Considerations

IoT devices are currently being produced at a very significant rate due to the high demand of developing the so-called “smart” houses and lifestyle. This results in fast production of devices without considering the quality of security incorporated. Our project can help validate some security properties without slowing down the development process too much and thus be applicable to a wide range of users and a potential large amount of code.

3.5 Design Analysis

This design has been proactively tested. The IoT code was successfully implemented into a verification task and had a successful verification run on the specified machines. Through the use of constant revisions within the code, it was able to be imported into the VM’s and tested against the use cases. As the project progresses, additional iterations will be made through revisions within the code so that additional cases can be tested and used.

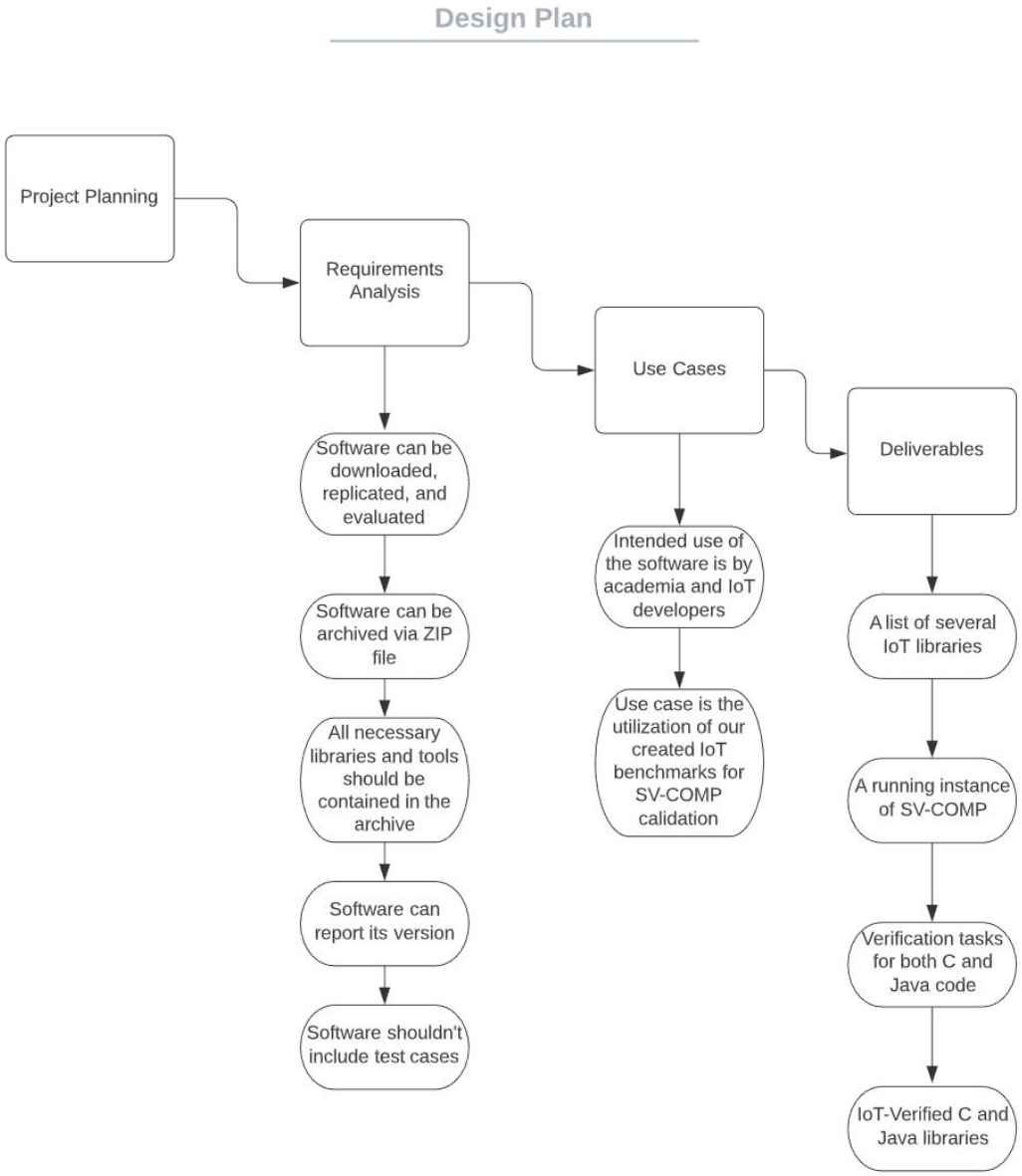
3.6 Development Process

Our team will be using the waterfall model for our development process for this project. This is largely due to the fact that we are having to slowly learn/alter our project goals as we move forward. It is important for us to fully check in with our client at each step of the way so as to ensure that we are on the right track. Through the use of systems such as Slack, Discord, and Git, we are able to keep each other informed of any developments. Any advancements with code are stored within a branch with the git repository, and after it is verified, merged within the main branch. All work is documented for others to use, and progress for each team member is updated during designated meeting times using Discord or Zoom.

3.7 Design Plan

Our Design Plan follows four overall steps which can each contain substeps. The four main steps are as follows: project planning, analysing requirements, identifying use cases, and producing

deliverables. Three out of the four main steps are made up of varying amounts of substeps. These steps and substeps are outlined in the diagram found on the next page.



4 Testing

4.1 Unit Testing

Once we have created our verification tasks, each of the security property tasks for Java and C will need to be validated using verification runs. So far we have chosen the following security properties to test: Improper Input Validation, Out-of-bounds Read and Write, Integer Overflow, Improper Authentication. For each of the properties, we will need unit tests for C and Java to make sure each IoT code successfully runs according to its original design. Another part of our project is setting up an environment of SV-COMP. Some testing will be required to make sure that the environment is set up correctly. Since we also will have to round out some of the IoT libraries that we pull, testing will be conducted to show that the original functionality of the libraries remains unchanged.

4.2 Interface Testing

BenchExec is the main interface that is used in our project. It is an interface that is not developed by us, but is simply being used to run simulations. Being that it was not developed by us, we will not be testing it. BenchExec will be the program that is used to test our tools as well as our IoT code.

4.3 Acceptance Testing

In order to demonstrate that the design requirements are being met, we will conduct black-box testing. This way we can show through the user perspective if there are any discrepancies based on the specifications. It also helps with having an objective perspective and avoids developer bias. By conducting acceptance testing through the black box method, it will be in terms that are quickly understood by everyone, including the client.

4.4 Results

Because it has been proactively tested, we know that the design can work. However, at our current state, we are unable to make any further tests due to incomplete benchmarks. Once these benchmarks have been completed, we will have additional information to make changes as needed.

5 Implementation

We will be creating and testing the tasks for Java and C. This process has begun and is expected to be completed and published by the end of next semester.

6 Closing Material

6.1 Conclusion

So far this semester we have investigated SV-COMP and security properties commonly associated with IoT devices. After the group obtained a general understanding, we have defined a new set of properties that we may add to the already existing properties of SV-COMP. In order to test and run these properties in the future we have requested, obtained, and set up virtual machines. Lastly, our group has split into two teams - Java and C - in order to cover the multiple code bases of SV-COMP.

The goal by the end of this project is to have an extensive list of properties that we may test Java and C IoT libraries for. The final goal is to have a public list of libraries that meet our security requirements.

Our group has split into two teams in order to cover Java and C IoT libraries. The Java tests in SV-COMP are less developed and thus must be caught up. Once the Java group is caught up to where the C group is, the C group can help guide the Java group to progress. This is optimal because it allows progress to be made on both fronts. Finally, testing of the libraries against our properties will occur, and be added to the list.

6.2 References

[1] Cwe.mitre.org. 2020. *CWE - Common Weakness Enumeration*. [online] Available at: <https://cwe.mitre.org/> [Accessed 25 October 2020].

[2] sv-comp.sosy-lab.org. 2020. *SV-COMP 2021*. [online] Available at: <https://sv-comp.sosy-lab.org/2021> [Accessed 25 October 2020]