Dynamic Programming Based FPGA Hosted Control-Flow Integrity (CFI)

Design Document

sdmay21-10
Professor Akhilesh Tyagi
Advisers: Ananda Biswas and Zelong Li
Greg Wendt/Meeting Scribe
Cole Schumacher/Meeting Facilitator
Nickolas Mitchell/Chief Engineer
Samuel Henley/Chief Engineer
Maxwell Wangler/Test Engineer
Tristan Duyvejonck/Report Manager
sdmay21-10@iastate.edu
http://sdmay21-10.sd.ece.iastate.edu

Revised: 1/15/2020 Final

Executive Summary

Development Standards & Practices Used

We use the following standards for interfacing with the FPGA:

- FMC (FPGA Mezzanine Card): Most popular and versatile interface standard
- Digital PMOD: A nice, simple alternative to FMC
- SYZYGY: newer standard that provides a middle ground between these two
- Standard design practices:
    - Well-documented
    - Efficient use of resources
    - Using proper IP's and memory cores

We use the following standards in regards to Control-Flow Integrity:

- Current CFI strategies
    - Stack Canaries
        - Non-executable memory
    - Write XOR eXecute
    - Code randomization
- Coarse-grained CFI
- Fine-grained CFI
- We want the best possible security with the lowest possible performance impact

The following IEEE Standards were considered in this design:

- IEEE 1008-1987: IEEE Standard for Software Unit Testing
    - This standard is used to facilitate accurate and consistent unit testing for the software part of the project. This applies as the algorithm the project is built off of must be rigorously tested.
- IEEE 1500-2005: IEEE Standard Testability Method for Embedded Core-based Integrated Circuits
    - This standard is used to facilitate the testing of the hardware side of the project. As the hardware can't be tested in the same methods as the software tests, this standard is applied to the integration between the software and hardware.
- IEEE 15288-2004: Systems and Software Engineering System Life Cycle

Processes
    - This standard is used in the design and development of a framework in which the software can maintain a healthy life cycle. This included a section about how this deals with . This applies to the project as it helped guide the development so the project is stable after development.
- IEEE 1220-2005: IEEE Standard for Application and Management of the Systems Engineering Process
    - This standard is used to create necessary infrastructure for life cycle sustainment. This applies to the project as it helps build out that infrastructure necessary for post development implementation of the project.

Summary of Requirements

A high level overview of the function requirements is given below:

- We will create an algorithm to match two sets of strings
- One string is generated based on program control flow, and is compared to a defined string
- It will use a two dimensional grid with its sized based on the number of basic blocks in the program

A high level overview of the nonfunctional requirements is given below:

- The algorithm must perform fast. The goal is O(nlog(n)).
    - current design is O(nm)

Applicable Courses from Iowa State University Curriculum

The following courses at Iowa State University were applicable to our project:

CPRE 281 - Digital Design

CPRE 381 - Computer Organization and Assembly Level Programming

COM S 228 - Introduction to Data Structures

COM S 309 - Software Development Practices

COM S 311 - Design and Analysis of Algorithms

COM S 321 - Introduction to Computer Architecture and Machine-Level Programming

COM S 327 - Advanced Programming Techniques

ENGL 314 - Technical Communication

S E 329 - Software Project Management

New Skills/Knowledge acquired that was not taught in courses

LLVM
Parallelism with FPGAs
CFI
LLVM Passes
PMU
RaceLogic

Table of Contents

List of figures/tables/symbols/definitions

1 Introduction

1.1 Acknowledgement

Work on this project is taking place along with Dr. Akhilesh Tyagi, and graduate assistants Ananda Biswas and Zelong Li. All of which have a significant expertise in the computer engineering and software engineering field. Our team is working closely with these assistants to familiarize ourselves with the project and the problems we need to solve.

1.2 Problem and Project Statement

General Problem Statement:

We are trying to determine if a break exists within the control flow of a program. When a program is compiled, it is split into sets of instructions called basic blocks. The computer jumps between basic blocks during execution. A vulnerability exists where a malicious user could inject some code which causes the program to jump to an unexpected block. In this case, the Control Flow Integrity has been breached.

General Solution Approach:

To do this, we will develop an algorithm for comparing the expected control flow of a program to the given control flow. The algorithm will determine if the difference between control flows constitutes a breach in Control Flow Integrity. The algorithm will implement concepts of race logic and will run on an FPGA board.

1.3 Operational Environment

The Operational Environment for this program will be lab rooms. No extreme conditions are expected.

1.4 Requirements

A high level overview of the function requirements is given below:

- We will create an algorithm to compare two sets of strings
- One string is generated based on program control flow, and is compared to a defined string
- It will use a two dimensional grid with its sized based on the number of basic blocks in the program
- Standardized solution to DE2-115 FPGA board.

A high level overview of the nonfunctional requirements is given below:

- The algorithm must perform fast. The goal is O(nlog(n)).

1.5 Intended Users and Uses

Intended users of this project include IT personnel and any company concerned with cyber security.

One of the core use cases is a scenario where a malicious user attacks a computer server by attempting to alter the control flow of a program. Our program will notify the system of a control flow breach in an effort to mitigate the attack. In this scenario, the user would be the personnel managing the computer server.

1.6 Assumptions and Limitations

Assumptions:

- The solution for DNA sequencing can be applied to this problem.

Limitations:

- We must use dynamic programming to determine if we lost control flow integrity.
- We must use a FPGA board to test our solution.
- The FPGA board shouldn't cost more than $100.

1.7 Expected End Product and Deliverables

The deliverable for this project is an FPGA board with a working program. The program shall compare two strings: one generated based on program control flow and a defined one. The expected delivery date for this is April 30, 2021.

The expected delivery dates are as follows:

- Ideate solutions: October 26, 2020
- Test solution in higher-level programming language: November 20, 2020
- Implement race conditions logic: February 12, 2021
- Implement solution in FPGA: April 30, 2021

2 Project Plan

2.1 Task Decomposition

1. Develop solution (Meant for understanding race logic/cfg)
    a. Write pseudocode for solution
    b. Develop solution in higher-level programming language
    c. Test solution
    d. Look at binary/cfg (LLVM)
2. Implement solution in FPGA board
    a. Implement using FPGA emulator
    b. Implement on actual FPGA

3. Features (Extra tasks to consider after our base project is complete)
    a. Controller for managing all components of the system
    b. Create visual UI

2.2 Risks And Risk Management/Mitigation

- Possibly override code of other team members
    - The use of git prevents this to a high degree.

2.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Our first key milestone includes, completing development on the software solution which will detect whether there is a significant breach in Control Flow Integrity with above 50% accuracy. Note, the target accuracy is subject to change based on testing. Our team's last milestone will be completing implementation on hardware with accuracy similar to our software solution.

2.4 Project Timeline/Schedule

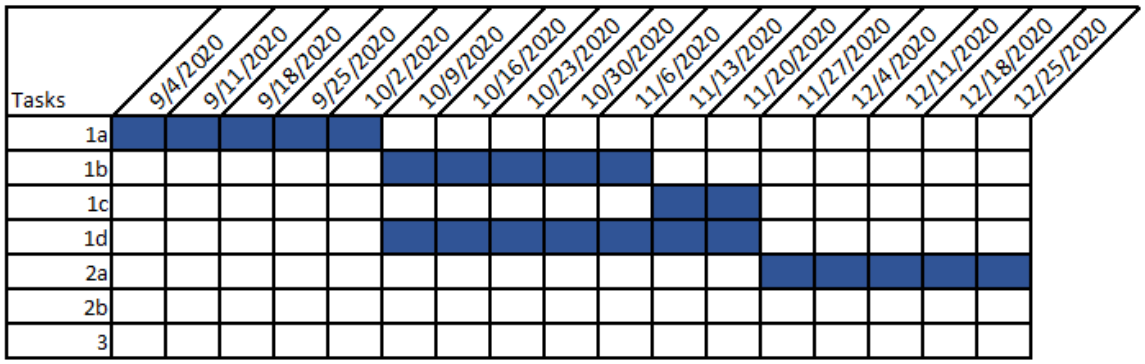

Table 2.1: Gantt Chart for tasks (2020)

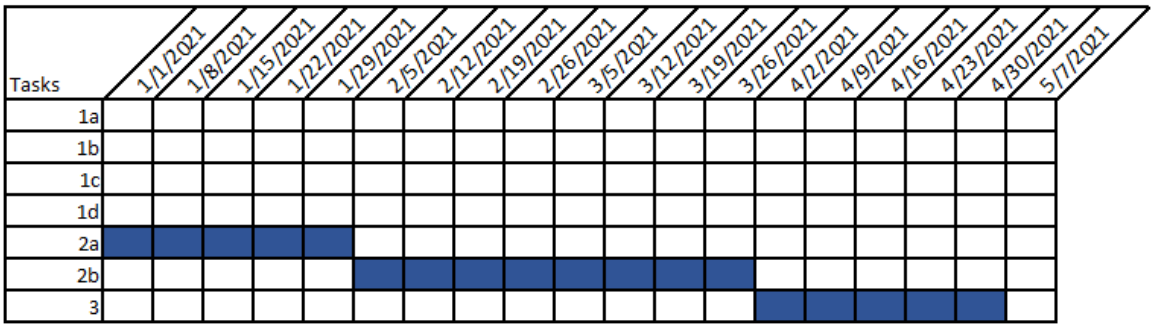

Table 2.2: Gantt Chart for tasks (2021)

Tables 2.1 and 2.2 show the proposed schedule for completing tasks laid out in Section 2.1.

2.5 Project Tracking Procedures

- Our group is using Slack to communicate with the team
- Our group is using a Google Drive to store team documents.
- Our group is using Github to store files relating to the project.
- Our group is a combination of GMAIL, Webex, and ZOOM, to communicate with our clients.

2.6 Personnel Effort Requirements

Task	Hours Per Person	Team members assigned	Total person-hours
1a	4	1	4
1b	5	2	10
1c	5	3	15
1d	15	3	45
2a	10	3	30
2b	15	2	30
3	15 (estimate)	6 (estimate)	90 (estimate)

Table 2.3: Effort Estimate by task

Our team will expect all team members to complete the tasks assigned to them by a predetermined deadline determined by the team. Our team expects total weekly hours per member to not exceed around 40 hours in week given several tasks to be completed in said week. In general, our team expects all members to give an appropriate number of hours to provide quality while still time managed work. Also please note, task three involves features not critical to our project, therefore these hours are estimates if we are able to complete task three. If we are unable to begin task three, these hours will not be required.

2.7 Other Resource Requirements

Our team plans to use a DE2-115 FPGA board due to its accessibility from Iowa State's ETG department. The DE2-115 FPGA board also offers features important to our project such as JTAG and Active Serial programming use over USB, various LEDs including red and green LEDs to receive real time visual feedback, and more than enough memory to support our solution.

2.8 Financial Requirements

Our team plans on no financial requirements being necessary.

3 Design

3.1 Previous Work And Literature

A. Madhavan, T. Sherwood and D. Strukov, "Race Logic: Abusing Hardware Race Conditions to Perform Useful Computation," in *IEEE Micro*, vol. 35, no. 3, pp. 48-57, May-June 2015, doi: 10.1109/MM.2015.43.

Gomez-Martinez, G. (n.d.). UART TRANSMISSION ON THE DE2-115 BOARD. Retrieved April 1, 2021, from http://cmosedu.com/jbaker/students/gerardo/Documents/UARTonFPGA.pdf

The *Race Logic: Abusing Hardware Race Conditions to Perform Useful Computation* paper by Madhavan, Sherwood, and Strukov was useful in understanding the algorithm we use in the system. The authors applied race conditions to solve a string-matching problem related to protein sequencing. We adapted this algorithm to analyze control flow integrity. The *UART TRANSMISSION ON THE DE2-115 BOARD* article by Gomez-Martinez goes over the details of communication between a FPGA board and computer over UART. This idea was used to guide us in how we set up communication between our FPGA board and computer.

3.2 Design Thinking

To define the problem that shaped our design, we worked closely with the professor to understand the needs and core issues of monitoring Control Flow Integrity. Our final goal is to implement the solution on an FPGA board, but first we wanted to focus on developing a solution that could be implemented and tested in a high-level environment.

When we ideated solutions for the problem, we discussed ideas with our graduate assistants. We focused on one solution and built on it until we could determine if the solution was feasible.

3.3 Proposed Design

We are not using any particular design standards at this time. Our first step was to design pseudo code and come up with ideas that theoretically solve the problem. After we discuss the pseudocode with the client, we want to try to implement the pseudo code in a higher-level programming language and test its functionality. Our final proposed design includes implementing our software solution on an FPGA board.

One of our project goals is to implement race logic in the design of our string comparison algorithm. We believe that abusing race conditions and delay operations will result in more efficient execution of our program. Figure 3.2 represents a two-dimensional grid of how the publication by Madhavan, Sherwood, and Strukov solved race logic using a dynamic programming approach.

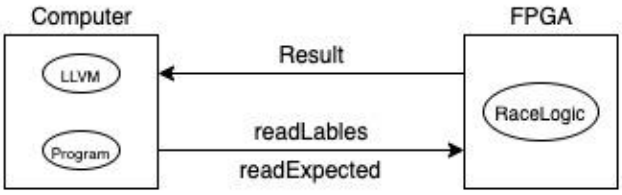

Figure3.1: Overall Proposed Design

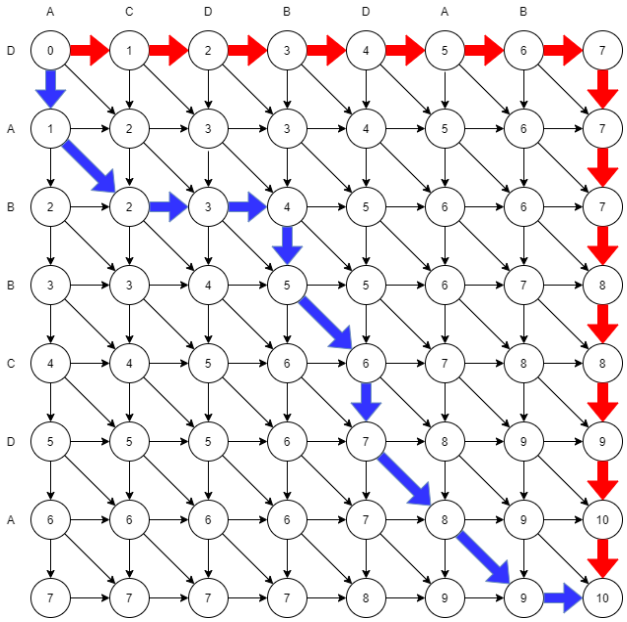

Figure 3.2: Race Logic

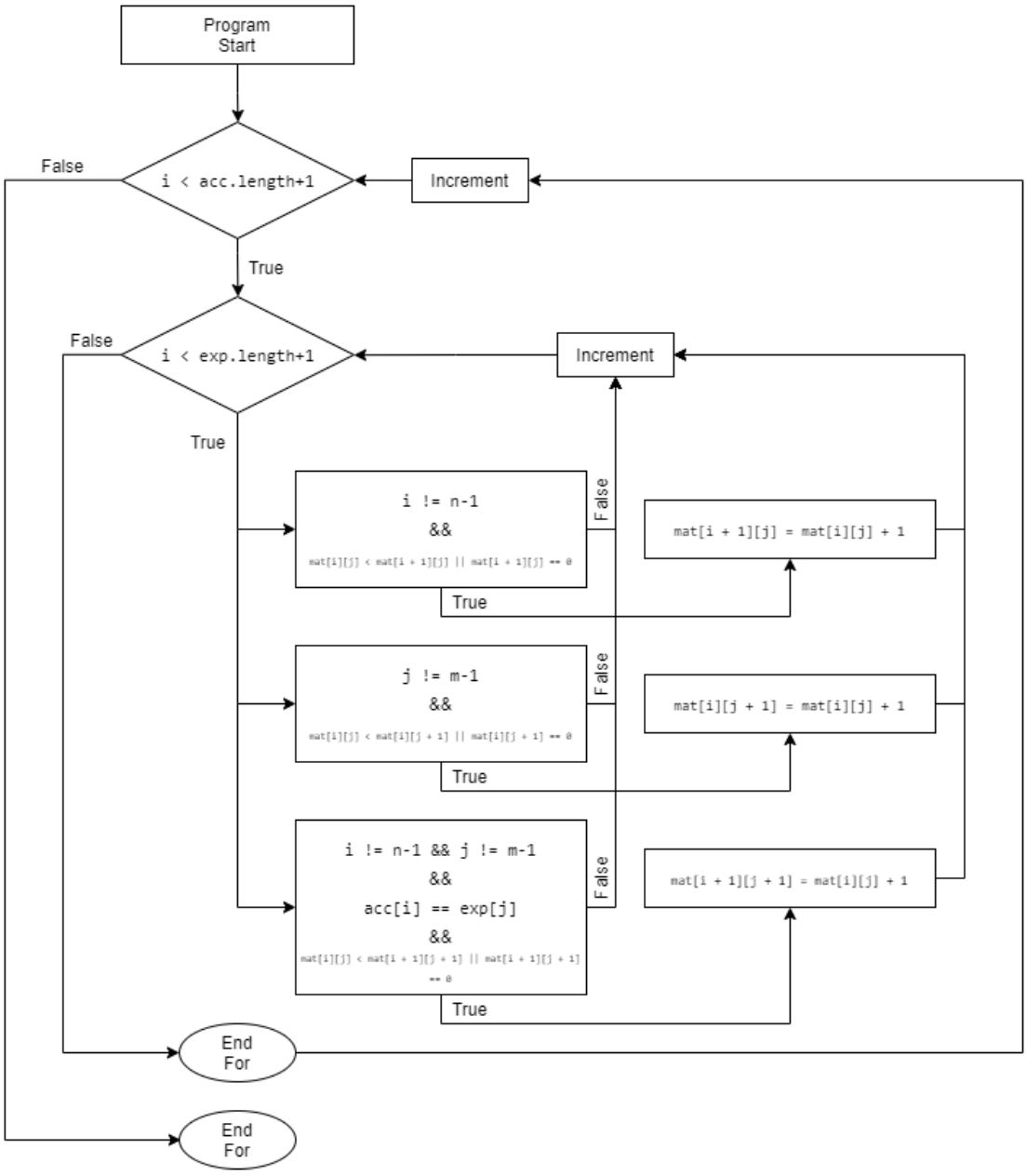

Figure 3.3: Race Logic Flowchart

Our logic works by first taking the length of two strings. If any string is of length zero, the logic returns -1 to show there was a problem. A matrix is then created using these two lengths for its height and width. Two nested for loops go through each tile in the matrix and checks if the tile ahead is of value zero or greater than the current one. If so, the next tile is set to the value of the current tile plus one. The next tile is determined both vertically and horizontally. If the index of both for loops results in an equal character in the strings, the tile to the bottom right of the current tile is increased by one. The value of the bottom right tile is returned as a value showing how different the actual and expected strings are.

Along with developing the string comparison algorithm, we have begun experimenting with LLVM, a set of tools for analyzing and manipulating compiled programs. LLVM will be useful for injecting functions into another program. LLVM can also insert static labels into a program.

Furthermore, we hope to take advantage of LLVM passes to further analyze control flow integrity. LLVM passes are a powerful tool of LLVM which allows for deep analysis of programs down to assembly language. For example, we could keep track of the number of times a certain opcode is run or we could keep track of the number of basic blocks per loop. It is also important to note, LLVM passes can be chained together. Our plan is to produce two LLVM passes:

1) BlockLabelsPass: This pass iterates over each Basic Block in the program. When the pass encounters a Branch instruction, it outputs a unique identifier associated with the current Basic Block. This pass is used to generate the static "expected" sequence of Basic Blocks encountered before the program is executed.
2) InjectFunctionPass: This pass iterates over each Basic Block in the program. When the pass encounters a Branch instruction, it inserts a call to a separate function, passing the unique identifier associated with the current Basic Block as a parameter. This separate function is used to output the current Basic Block when the program is executed. This pass is used to generate the dynamic "actual" sequence of Basic Blocks encountered during execution.

The FPGA solution to our proposed design will include three major modules. First is the memory module which will hold the values from ReadLables and ReadExpected. ReadLabeles and ReadExpeteced will be discussed in more detail in section 3.7. The next module, Race Logic, will take an FPGA implementation of our race logic software solution and use ReadLables and ReadExpected to determine a delay. This result will then be sent to the classify module where we hope to implement a way of interpreting the result received from the race logic algorithm. As it stands, the classify module is not crucial to our overall solution and therefore it is considered to be a feature we will tackle after completion of all priorly mentioned modules.

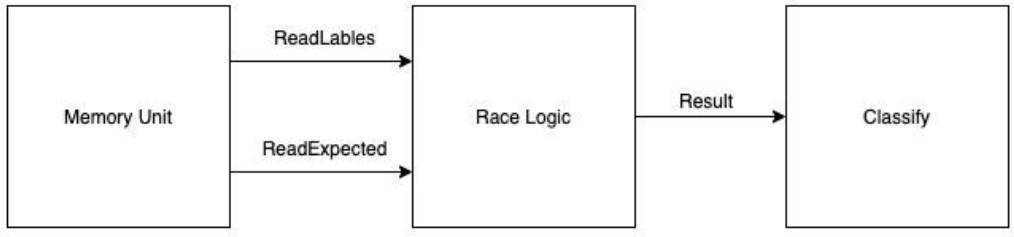

Figure 3.4: FPGA Design schematic

As an additional feature, we would like to develop an environment that combines the three modules: compiles source code and invokes passes, sends the data to be processed by the race logic algorithm, and receives the output from the classification algorithm. This feature will include a refined user interface. It will display sample 8-bit strings that have been run through our race logic algorithm, and it will also display their race logic score. It will also display the 8 by 8 grid that the race logic used to calculate the race logic score.

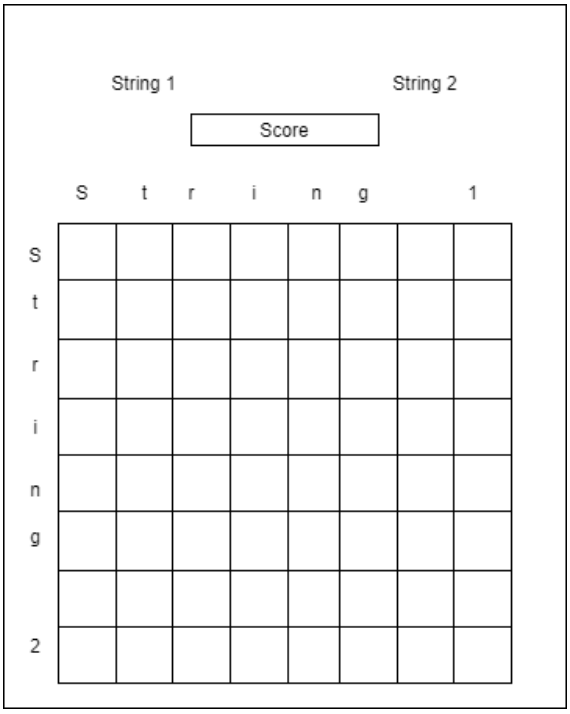

Figure 3.5: Feature Display Schematic

3.4 Technology Considerations

One design alternative available is to use an ASIC system instead of an FPGA. We think the project would be worse off with an ASIC as the processing cycles would be interrupted by the primary compute unit. The ASIC unit would also be more difficult to scale.

Another tradeoff our team is considering alongside our clients is whether to use PMU or static labels to determine if control flow integrity has been breached. Static labels would simply be labels we assign to different blocks of code which would be simpler then PMU but potentially comes at a performance cost. PMU utilizes the libperf library which is a performance analyzing tool which can be used to keep track of the number of times a block of code has been run. PMU will offer a less resource intensive solution when compared to static labels but the reliability of using PMU is uncertain. With more testing of both of these possible solutions, we hope to determine the best for our project.

3.5 Design Analysis

As of earlier this semester, our team completed testing on both our race logic in C and Java. Based on this testing, we found our design to not need any major revisions or iterations. Therefore, our proposed design from section 3.3 seems to be sufficient to meet the needs we tested for.

3.6 Development Process

Our team is using a waterfall model development process. We decided the nature of our project was not suited for an agile development process with small, incremental iterations. Our project will move through each stage of the development process once and in order.

3.7 Design Plan

The system is composed of three modules:

1. Reading labels from binary
2. Read expected labels
3. Classifier

The first module reads control labels from the executed binary. The labels could be the PMU data from the processor, or another label to be determined. The second module reads the expected control flow model for the executable. The data from both modules is passed to the classifier, which determines whether there is a breach in control flow integrity.

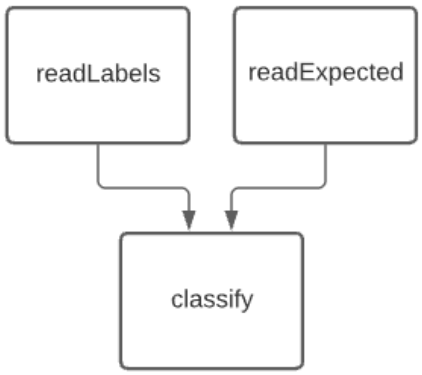

Figure 3.6: Program execution flowchart

This system fulfills the use cases specified for our solution. Our user needs to be notified if a control flow breach is detected. The 'classify' module will determine if there is a control flow breach, which receives data from the 'readLabels' and 'readExpected' modules.

4 Testing

4.1 Unit Testing

Software we plan to test in an isolated environment includes our dynamically programmed race logic algorithm, and any CFG programs we design. Hardware we plan to test in an isolated environment includes an FPGA board running our solution.

The race logic tests will include edge cases such as 2 empty strings, 1 empty string on each side, uneven strings, 1 character strings, and strings that use letters and numbers. The tests are built in JUnit and a Custom C test framework for corresponding Java and C code

4.2 Interface Testing

The communication between the reading modules and the classification module will be tested by recording the data output by the reading modules and analyzing the behavior of the classification module.

4.3 Acceptance Testing

To verify the functionality of the system, develop input sets for best-case and worst-case scenarios. Apply the input sets to the system. For example, create a scenario when the model and the input data match exactly. Upon completion of the project, we will demonstrate the system to the client during one of our meetings.

4.4 Results

The race logic test gives a verification on which edge cases have passed. All 13 tests on both Java and C have passed.

Our results pertaining to LLVM were below our expectations. The BlockLabelsPass was expected to follow the control flow of the program and print the sequence of basic blocks that should be encountered. However, the pass simply iterates over a list of basic blocks contained in the program, not necessarily following the control flow path. Similarly, the InjectFunctionPass was not providing the desired output.

5 Implementation

Preliminary implementation for our project includes finalizing our software solution before translating it to an FPGA board.

For the spring semester, we started by testing our solution more thoroughly in Java and C. We then worked on implementing the solution on a physical FPGA board. While doing this, some of our team members worked to complete the LLVM parts for our project. Lastly, we worked towards setting up the communication link between the FPGA board and a computer.

6 Closing Material

6.1 Conclusion

As of the end of the first semester, our team has worked with our professor and graduate assistants to understand control flow integrity and the basics of program analysis. Based on our meetings with the advisers, we developed an algorithm in pseudocode that we could discuss and revise. Then, we implemented the algorithm in a higher-level language. We are currently in the process of testing the algorithm.

Our primary goal is to implement the solution in a physical FPGA board. Secondary to this, we would like the algorithm to run with time complexity $O(n\log(n))$. Our best plan of action is to implement the algorithm in a high-level language first, then an FPGA emulator, then a physical FPGA board. This way, we are able to transition from one medium to the next, testing and detecting problems along the way.

6.2 References

A. Madhavan, T. Sherwood and D. Strukov, "Race Logic: Abusing Hardware Race Conditions to Perform Useful Computation," in *IEEE Micro*, vol. 35, no. 3, pp. 48-57, May-June 2015, doi: 10.1109/MM.2015.43.

6.3 Appendices

https://www.intel.com/content/dam/altera-www/global/en_US/portal/dsn/42/doc-us-dsnbk-42-1404062209-de2-115-user-manual.pdf

The above documentation is the manual to the DE2-115 FPGA board our team plans to use in our solution.

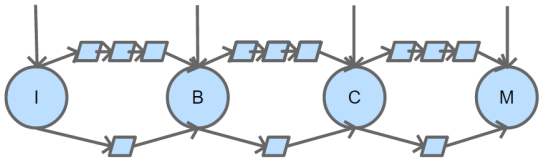

Figure 6.1: Conceptual Sketch