Robustness of Microarchitecture Attacks/Malware Detection Tools against Adversarial Artificial Intelligence Attacks

DESIGN DOCUMENT

Team: sdmay23-16
Client/Adviser: Berk Gulmezoglu

Team Members/Roles:
Liam Anderson / Internal Logic Member
Kevin Lin / Machine Learning Lead
Shi Yong Goh / Internal Logic Member
Felipe Bautista / UI Lead
Connor McLoud / OS Lead
Eduardo Robles / Internal Logic Member

Team Email: sdmay23-16@iastate.edu
Team Website: http://sdmay23-16.sd.ece.iastate.edu/

Revised: December 2nd, 2022/1.0

Executive Summary

Development Standards & Practices Used

Software Development Life Cycle

IEEE 12207: This standard provides processes that can be employed for defining, controlling, and improving software life cycle processes within an organization or a project.

IEEE 1074: This standard provides a process for creating a software project life cycle process (SPLCP), and it is primarily directed at the process architect for a given software project.

Software Testing

IEEE 29119: This is a series of five standards for software testing. They define vocabulary, processes, documentation, techniques, and a process assessment model for testing that can be used within any software development lifecycle.

Summary of Requirements

Functional Requirements

- Generate adversarial examples for all provided attack codes
- Adversarial examples successfully perform the attack while under the following constraints:
  - Score below 20% detection certainty
  - Never exceeding 2x normal power consumption
  - Attack speed doesn't exceed 5x slower data leak rate than its non-evasive counterpart

UI Requirements

- The software must feature a user-friendly graphical user interface with the following features and functions:
  - Define an attack type
  - Upload datasets and a machine learning model
  - Upload attack source code
  - Select between different detection models
  - Launch evasive power-mimicking attacks

Applicable Courses from Iowa State University Curriculum

Courses at Iowa State whose content applies to this project:

- **CPR E 381:** Computer Organization and Assembly Level Programming
- **CPR E 308:** Intro to Operating Systems
- **ENGL 314:** Technical Communications
- **COM S 309:** Software Development Practices
- **CPR E 185:** Intro to Computer Engineering and Problem Solving
- **CPR E 230:** Cybersecurity Fundamentals
- **COMS 474**: Intro to Machine Learning

New Skills/Knowledge acquired that was not taught in courses

New skills or knowledge needed for this project:

- x86 Assembly
- Machine Learning
- Scripting Languages (Bash, Python, etc.)
- MATLAB

Table of Contents

List of Figures

List of Tables

1. Team

1.1 Team Members

Liam Anderson

Kevin Lin

Shi Yong Goh

Felipe Bautista

Connor McLoud

Eduardo Robles

1.2 Required Skill Sets for Your Project

This project requires a wide variety of skills to complete. Knowledge of the Ubuntu operating system and bash is necessary to work with the provided systems for collecting and analyzing data. Assembly level and C programming skills are needed to work with provided attack codes and to create adversarial attacks. The project also requires us to work with machine learning models, so knowledge in that area and python is necessary. Lastly, this project is centered around microarchitecture vulnerabilities and compromising information systems, so a background in cybersecurity is helpful.

1.3 Skill Sets covered by the Team

1. **Ubuntu OS experience:** Kevin, Connor, Felipe, and Eduardo
2. **Cybersecurity background:** Liam
3. **Machine Learning knowledge:** Kevin
4. **Assembly level programing:** All members
5. **C programming language:** All members
6. **Python:** All members
7. **Bash:** All members

1.4 Project Management Style Adopted by the team

Our team will use a hybrid development process to manage the project. For many tasks we must follow the waterfall process and other areas of the project will follow the agile process.

1.5 Initial Project Management Roles

- **Client Interaction:** Shi Yong
- **Team Organization:** Kevin
- **Testing:** Felipe Bautista Salamanca, Connor McLoud
- **Individual Component Design:** Eduardo Robles, Liam Anderson

2. Introduction

2.1 Problem Statement

In a technology centric world, cybersecurity is crucial to ensuring consumer privacy of information. Some microarchitecture-based malware attacks cannot be detected using current existing software – causing a breach of security and loss of privacy. Major chip manufacturers and cloud computing providers need a way to identify and differentiate these attacks from benign signals in order to ensure consumer privacy. These attacks can happen at any given time, making it difficult to identify them consistently and accurately. Thus, our team is creating a software tool that can assess the robustness of an AI based detector against microarchitecture attacks. This will allow companies to strengthen and improve their own software to better detect and quarantine said attacks.

2.2 Requirements & Constraints

2.2.1 Functional Requirements

The software our team will develop will assess the robustness of security systems that attempt to detect microarchitecture attacks. The robustness will be measured by its ability to detect microarchitecture attacks specially designed to evade detection. The software will generate these evasive adversary attacks by inserting artificial noise into the attack instructions to mimic benign power signatures and exploit the security system's underlying machine-learning model.

Five microarchitecture attack codes will be provided, and all five attacks must be able to execute without detection and without significantly slowing down the attack. The security system cannot report any higher than 20% detection certainty for the attack to be undetected. The power usage should not exceed 2x normal activity, and the attack should not surpass a 5x slower data leak rate than its non-evasive counterpart.

2.2.2 UI Requirements

Our team's program will feature a simple and user-friendly graphical user interface (GUI). The interface must allow users to do the following tasks:

1. Define the attack type
2. Include the data sets they used.
3. Upload the application's source code used to train the model
4. Upload attack source code
5. Select between different detection models
6. Run evasive power-mimicking attacks

After running an attack, the GUI must display statistics about the attack, including the data leak rate in bits per second and the security systems malware detection certainty for the selected model as a percentage.

2.2.3 Resource Requirements

The project will require specialized hardware to pull the necessary CPU power consumption data and achieve the performance needed to run the AI-based microarchitecture attack detector. Our team will be provided and required to use the following experimental setup:

- Intel Comet Lake Microarchitecture
    - CPU Model: Intel(R) Core (TM) i7-10610U CPU @ 1.80GHz
    - OS: Ubuntu 20.04 LTS
    - Linux Kernel: 5.11.0-46-generic
- Server Information
    - Nvidia GeForce RTX 3090 GPU
    - CPU Model: Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz

2.3 Engineering Standards

Software Development Life Cycle

IEEE 12207: This standard provides processes that can be employed for defining, controlling, and improving software life cycle processes within an organization or a project.

IEEE 1074: This standard provides a process for creating a software project life cycle process (SPLCP) and it is primarily directed at the process architect for a given software project.

Software Testing

IEEE 29119: This is a series of five standards for software testing. They define vocabulary, processes, documentation, techniques, and a process assessment model for testing that can be used within any software development lifecycle.

2.4 Intended Users and Uses

2.4.1 Researchers

Researchers focusing on microarchitecture attacks will use our tool to test and further research power-anomaly detection systems.

Key Characteristics

Researchers are the leaders in developing and further exploring microarchitecture vulnerabilities and security solutions. They have a vast knowledge of computer engineering and system security. They are more concerned with the pursuit of knowledge and development of theory than they are with implementing solutions for economic reasons.

Needs

Researchers need a better way to test new microarchitecture attacks against their deep learning power-anomaly detection systems. They must validate that their security systems hold up against evasive microarchitecture attacks.

How they will benefit

With the tool our team is developing, researchers can quickly generate new, highly evasive microarchitecture attacks. Saving them a lot of time implementing it themselves and providing them with the tools they need to test their systems.

2.4.2 Implementors

Implementors, such as Intel, AMD, Nvidia, and PaaS providers, will use our tool to test their systems against microarchitecture attacks.

Key Characteristics

Implementors are companies that manufacture chipsets or provide access to computer resources. Vulnerabilities in their products can damage their reputation and be very costly, so they tend to invest heavily in cybersecurity.

Needs

Implementors need a tool to test their products and discover potential vulnerabilities.

How they will benefit

Implementors will be able to conduct penetration testing on their products with evasive microarchitecture attacks and, as a result, discover existing vulnerabilities.

2.4.3 End Users

End users indirectly benefit by trusting their data with systems tested by our tool.

Key Characteristics

End users are typical everyday computer users. They don’t have much knowledge of cybersecurity or the inner workings of the system they are using, and they trust the product they use is secure.

Needs

End users need their data to be secure. They need their personal computer or cloud environment not to be vulnerable to microarchitecture attacks.

How they will benefit

End users can be assured that their private information is secured and inaccessible to non-authorized individuals. They can also have better protection on their personal devices and as well as their cloud environments.

3 Project Plan

3.1 Project Management/Tracking Procedures

Our team plans to adopt a hybrid development process to manage our project. Since some tasks depend on one another, it is necessary to follow the waterfall process to meet the requirements of these tasks. Our project has a unique feature in that each task must be done for each one of the attacks we are modifying. This feature allows us to work in parallel by assigning an attack to each member. Doing so allows us to work simultaneously on the same tasks for each attack. By doing this, we follow an agile process that would allow our team to tackle various tasks at the same time, which will save us time.

The team will use GitHub for version control and Notion to keep track of assigned tasks and due dates. We have a Kanban board on Notion and will follow the Gantt chart to ensure we're on track.

3.2 Task Decomposition

3.2.1 First Semester

Task 1: Become familiar with attack codes and test system

Task 1A: Understand the necessary Ubuntu scripts and test systems configuration

Task 1B: Collect power measurements/model power signatures

Task 1C: Modify attack codes to extract data leak rate & collect detection accuracy

Task 1D: Understand each attack's atomic instructions & possible areas to insert code

Task 2: Implement UI with basic functionality

Task 2A: Set up python environment and UI foundation

Task 2B: Create necessary functions

Task 2C: Implement command line interface

Task 2D: Design a graphical user interface

Task 2E: Test UI

3.2.2 Second Semester

Task 3: Analyze power signatures, instructions' power consumption, and evasive attacks

Task 3A: Analyze differences in malicious and benign power signatures

Task 3B: Attempt to mimic benign power signatures by inserting instructions

Task 3C: Profile and record x86 instruction's effects on power consumption

Task 4: Implement basic instruction insertion and attack logic

Task 4A: Develop instruction insertion structure with instruction's power signature dataset and code insertion functions

Task 4B: Develop attack structure with ammeter synchronization, code execution, and attack analysis functions.

Task 4C: Test functionality

Task 5: Leverage NLP and CNN AI techniques to create adversarial examples (2 months)

Task 5A: Develop ML models

Task 5B: Implement model into instruction insertion logic

Task 6: Finalize project

Task 6A: Add additional functions to the GUI

Task 6B: Final testing

Task 6C: Fix any minor issues

Task 6D: Wrap-up Documentation

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Milestone 1: Understand the attacks

The team should be able to collect the power measurements of the model. Understand where instructions can be added/removed from the attack code to reduce detection. Be able to calculate the leakage rate and detection accuracy.

Milestone 2: Completed UI

The UI should allow the user to upload an attack code, a detection model, and a data set. The user should be able to select the type of attack as well as run and end the attack. Once the attack is executed, the UI should display the detection rate, the average leak rate (bytes/sec), and the total bytes leaked.

Milestone 3: Complete 3 different attack codes

The team should have a completed Spectre, Row Hammer, and Port Smash attack. The ML model confidence rate should be below 20% after including instructions in each attack.

Milestone 4: Deliver the project to client

The GUI is updated with any changes that were made throughout development. Any final modifications of the attack codes should be made.

3.4 Project Timeline/Schedule

3.4.1 Fall 2022 Schedule

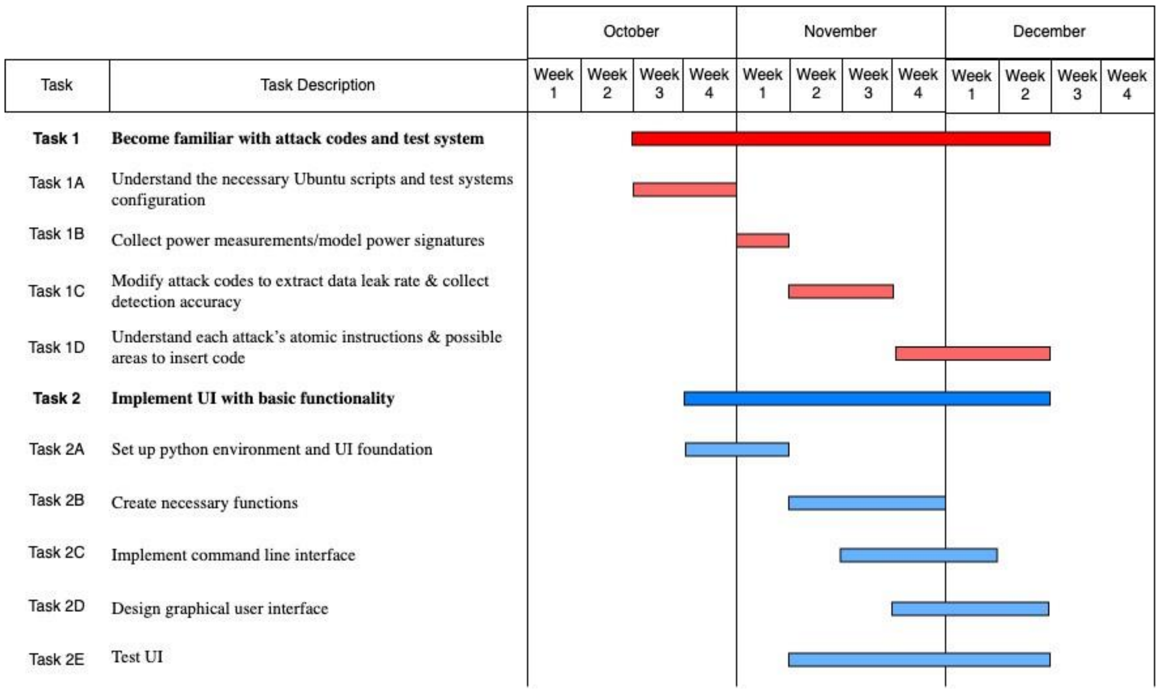
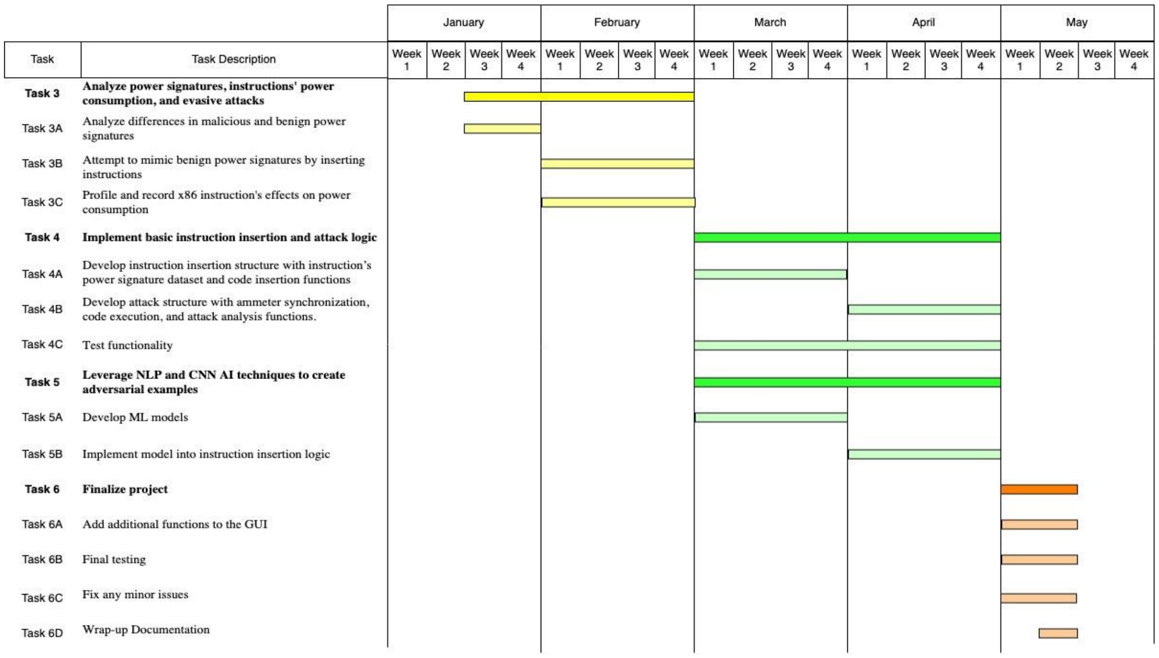


Figure 1: Fall Gantt Chart

We assigned Tasks 1 and 2 to be finished by the end of the semester. Having a strong understanding of the attack codes is crucial to developing/modifying them. If we have the GUI finished by Spring semester, testing will be much more efficient in later tasks.

Both tasks can be worked on in parallel – knowledge of the attack isn't necessary to complete the GUI. Task 1 has few overlaps since all of them build on each other. Task 2 allows for more overlap since the subtasks can be worked on simultaneously.

3.4.2 Spring 2023 Schedule



		January				February				March				April				May			
Task	Task Description	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4
Task 3	**Analyze power signatures, instructions' power consumption, and evasive attacks**			🟨	🟨	🟨	🟨	🟨	🟨												
Task 3A	Analyze differences in malicious and benign power signatures			🟨	🟨																
Task 3B	Attempt to mimic benign power signatures by inserting instructions					🟨	🟨	🟨	🟨												
Task 3C	Profile and record x86 instruction's effects on power consumption					🟨	🟨	🟨	🟨												
Task 4	**Implement basic instruction insertion and attack logic**									🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩				
Task 4A	Develop instruction insertion structure with instruction's power signature dataset and code insertion functions									🟩	🟩	🟩	🟩								
Task 4B	Develop attack structure with ammeter synchronization, code execution, and attack analysis functions.													🟩	🟩	🟩	🟩				
Task 4C	Test functionality									🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩				
Task 5	**Leverage NLP and CNN AI techniques to create adversarial examples**									🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩				
Task 5A	Develop ML models									🟩	🟩	🟩	🟩								
Task 5B	Implement model into instruction insertion logic													🟩	🟩	🟩	🟩				
Task 6	**Finalize project**																	🟧	🟧		
Task 6A	Add additional functions to the GUI																	🟧	🟧		
Task 6B	Final testing																	🟧	🟧		
Task 6C	Fix any minor issues																	🟧	🟧		
Task 6D	Wrap-up Documentation																		🟧		

Figure 2: Spring Gantt Chart

We have already begun Task 3: analyzing power signatures, power consumption, and evasive attacks. It is necessary to analyze the benign and malicious power signatures.

Tasks 3B and 3C can be done simultaneously after testing and analyzing the additional instructions that affect the power signatures. Task 4 and 5 is where the attack codes start development. The team will be split to allow both to be done at the same time. Task 6 is where the team finishes up the project. Here the team will allocate resources to complete the project at the same time.

3.5 Risks and Risk Management/Mitigation

Low Risk: Team Members are unable to finish work by the assigned deadline.

- Probability of occurring: high (70%)
- Mitigation: Clear communication with team members so that duties can be transferred and delegated, if necessary, to finish work by the deadline.

Moderate Risk: An attack code has a detection rate higher than 20%

- Probability of occurring: moderate (50%)
- Mitigation: allocate more time into analyzing the benign application's power consumption.

Moderate Risk: Leakage rate may be difficult to calculate

- Probability of occurring: moderate (40%)
- Mitigation: Spend more time researching the attack code.

High Risk: Two people run an attack at the same time (will cause incorrect data for both attacks)

- Probability of occurring: low (15%)
- Mitigation: Communicate when one starts and ends an attack.

High Risk: Row hammer attack may crash the system resulting in needing a new ram.

- Probability of occurring: low (5%)
- Mitigation: Have a good understanding of the attack and carefully implement it.

High Risk: Inability to analyze and understand power samples from ammeter

- Probability of occurring: low (5%)
- Mitigation: Ask for assistance if unable to understand the power consumption graphs

3.6 Personnel Effort Requirements

Tasks	**Total number of person-hours**	**Justification**
Task 1: Become familiar with attack codes and test system	150 hours	Task 1A: Understand the necessary Ubuntu scripts and test systems configuration (All members, 6 hours each) Task 1B: Collect power measurements/model power signatures (All members, 3 hours each) Task 1C: Modify attack codes to extract data leak rate & collect detection accuracy (All members, 8 hours each) Task 1D: Understand each attack's atomic instructions & possible areas to insert code (All members, 8 hours each)
Task 2: Implement UI with basic functionality	120 hours	Task 2A: Set up python environment and UI foundation (three members, 2 hours each) Task 2B: Create necessary functions (three members, 18 hours each) Task 2C: Implement command line interface (three members, 9 hours each) Task 2D: Design a graphical user interface (three members, 8 hours each) Task 2E: Test UI (three members, 3 hours each)

Task 3: Analyze power signatures, instructions' power consumption, and evasive attacks	240 hours	Task 3A: Analyze differences in malicious and benign power signatures (All members, 8 hours each) Task 3B: Attempt to mimic benign power signatures by inserting instructions (All members, 16 hours each) Task 3C: Profile and record x86 instruction's effects on power consumption (All members, 16 hours each)
Task 4: Implement basic instruction insertion and attack logic	168 hours	Task 4A: Develop instruction insertion structure with instruction's power signature dataset and code insertion functions (three members, 22 hours each) Task 4B: Develop attack structure with ammeter synchronization, code execution, and attack analysis functions (three members, 25 hours each) Task 4C: Test functionality (three members, 9 hours each)
Task 5: Leverage NLP and CNN AI techniques to create adversarial examples	168 hours	Task 5A: Develop ML models (three members, 30 hours) Task 5B: Implement model into instruction insertion logic (three members, 26 hours each)
Task 6: Finalize project	60 hours	Task 6A: Add additional functions to the GUI Task 6B: Final testing (all members, 4 hours each) Task 6C: Fix any minor issues (all members, 3 hours each) Task 6D: Wrap-up Documentation (all members, 3 hours each)
Total	906 hours	Finish

Table 1: Task Time Requirements

3.7 Other Resource Requirements

The project will require specialized hardware to pull the necessary CPU power consumption data and achieve the performance needed to run the AI-based microarchitecture attack detector. Our team will be provided and required to use the following experimental setup:

- Intel Comet Lake Microarchitecture
    - CPU Model: Intel(R) Core (TM) i7-10610U CPU @ 1.80GHz
    - OS: Ubuntu 20.04 LTS
    - Linux Kernel: 5.11.0-46-generic
- Server Information
    - Nvidia GeForce RTX 3090 GPU
    - CPU Model: Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz

4 Design

4.1 Design Context

4.1.1 Broader Context

The tool our team is developing is designed for those in academia researching machine learning solutions to malware attacks. It will help test, evaluate, and strengthen malware detection software to create more robust and secure systems and ultimately addresses the ever-growing need for safe and reliable information technology.

In addition, this can be used to supplement existing software that chip manufacturers like AMD and Intel can utilize.

Safety and Welfare

If we are not careful, our software can easily be used maliciously. Someone could use it to help them get past a system's anti-malware protection system and compromise the machine, affecting the confidentiality, integrity, and availability of the victim's data. We must keep this in mind when making design choices so the software can only be used for research purposes. Some of these design choices include not outputting the evasive attack code and only leaking designated data

4.1.2 Prior Work/Solutions

Before we started designing our project, we read a research paper, "*Using Power-Anomalies to Counter Evasive Micro-Architectural Attacks in Embedded Systems*", published by the Department of Electrical and Computer Engineering, The University of Texas at Austin.

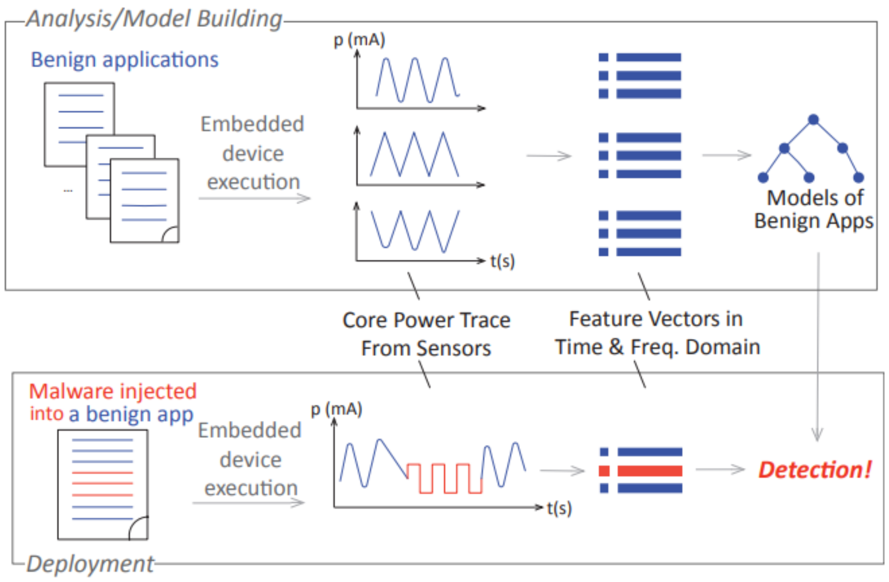

Analysis/Model Building
Benign applications
Embedded device execution
p (mA)
t(s)
Models of Benign Apps
Core Power Trace From Sensors
Feature Vectors in Time & Freq. Domain
Malware injected into a benign app
Embedded device execution
p (mA)
t(s)
Detection!
Deployment

Figure 3: Overview of Power-anomaly Detector System[1]

This research proposed a power-anomaly detector system (Figure 1), which is very similar to the detection tool that we will create. This system provides an effective way to detect a range of microarchitectural attacks: Spectre attacks, covert-channels and row hammer attacks. The research demonstrates that power anomalies can reliably cut through the noise of diverse benign programs to detect microarchitecture attacks in complex embedded systems. To model evasive attackers, they introduce evasive, power-mimicking micro-architectural attacks that replicate benign applications' power behavior while executing malicious tasks.

This power-anomaly detection can be used to defeat a particularly harmful and sophisticated class of attacks. The pro of the detector system is it is easy to deploy. Hence, it does not require expensive upgrades and can be strengthened to combat the class of attacks. But this system is limited to certain types of attacks.

4.1.3 Technical Complexity

Since our project can be seen as a research project, it includes various scientific, mathematical, or engineering principles. A big emphasis of our project is machine learning, which is a very complex topic for all of us since many of us don't have any prior background, which will require us to do research to be more familiar with the topic. Another key feature of the project is the various types of micro-architectural attacks we will examine and modify.

The goal of the project is to make changes to these given attacks for them to disguise themselves from the detection models. Our team needs to understand how these attacks exploit different weaknesses before modifying them, so we don't change the attack logic by accident when making changes to the source code. When it comes to modifying the attacks, it is essential for us to have

[1] Wei, Shijia & Aysu, Aydin & Orshansky, Michael & Gerstlauer, Andreas & Tiwari, Mohit. (2019). Using Power-Anomalies to Counter Evasive Micro-Architectural Attacks in Embedded Systems. 111-120. 10.1109/HST.2019.8740838.

good knowledge of C and assembly code. Since all the attacks will be written in these languages, any modification will also be required using the existing source code.

Assembly language can be tricky since it varies within the architecture of the system you are working on. Since the environment we are working on requires knowledge of x86 assembly code, our team needs to learn and understand this assembly language.

4.2 Design Exploration

4.2.1 Design Decisions

1. Implementing a GUI in python
    a. Choosing a specific language is important to get everyone on the same page.
    b. Python is the language the team has the most experience in. This means less time is being spent on learning a new programming language.
2. Attack codes will all be in C
    a. The team was supplied template C code on certain attacks. Analyzing and building off the templates would speed up the process of development. Translating the existing code into another language would be unnecessary and take too much time.
3. All assembly instructions will be in x86
    a. We will be trying to attack an Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz to test our attacks. This CPU is known to use assembly x86.

4.2.2 Ideation

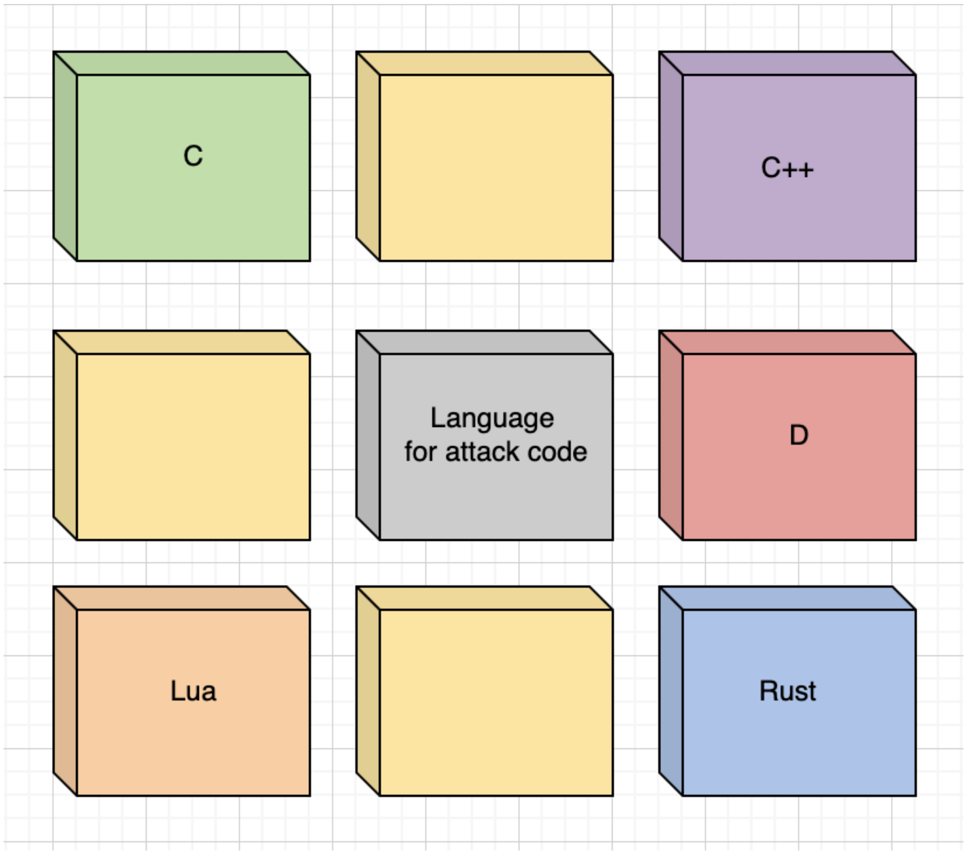


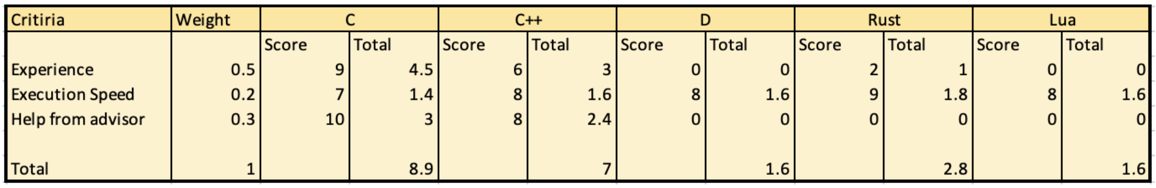
Figure 4: Lotus Blossom

We decided to use the Lotus Blossom ideation technique to narrow down which language was best suited for our attack codes.

4.2.3 Decision-Making and Trade-Off

There were 3 main factors when we decided the language of the attack codes: experience, execution speed, and help from advisor. Experience had a higher weight because the level of experience we have in a language determines the speed of development. Execution speed also plays a role in ensuring the results of the attack return quickly. This project is new territory for most of the team so being able to get help from the advisor would ensure the team has a good understanding of the project.

Below is a weighted decision matrix of the 5 potential languages for the attack code. D, Rust, and Lua scored very low because no one in the group had any experience with the languages. C was the best choice because the entire group has worked with it in previous classes. The advisor gave a recommendation to use C and provided a lot of help getting started on the project.



Critiria	Weight	C		C++		D		Rust		Lua	
		Score	Total	Score	Total	Score	Total	Score	Total	Score	Total
Experience	0.5	9	4.5	6	3	0	0	2	1	0	0
Execution Speed	0.2	7	1.4	8	1.6	8	1.6	9	1.8	8	1.6
Help from advisor	0.3	10	3	8	2.4	0	0	0	0	0	0
Total	1		8.9		7		1.6		2.8		1.6

Table 2: Weighted Decision Matrix

4.3 Proposed Design

4.3.1 Overview

Our software will offer a graphical user interface (GUI) or command line interface (CLI) for the user to interact with the application. Navigating the interface, the user can upload attack codes and select a detection model and attack type, which is fed into the instruction insertion algorithm. The algorithm then generates an evasive attack that exploits the detection system's underlying machine learning (ML) model. Running the attack with the program will generate statistics about the attack.

4.3.2 Detailed Design and Visual(s)

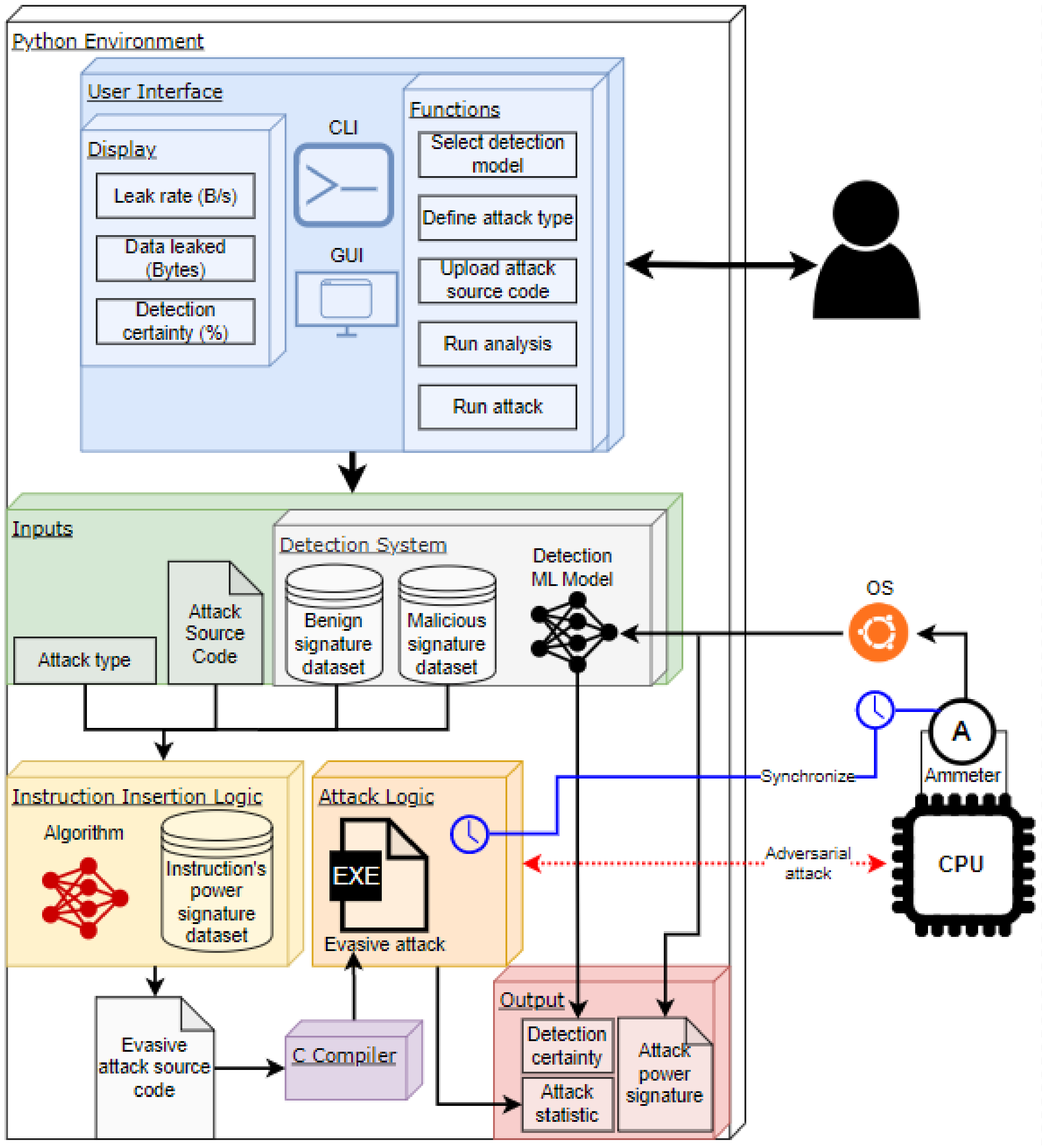


Figure 5: Detailed Design Overview

4.3.2.1 Overview

Our system primarily lives in a python environment. Currently, there exists a machine learning model that can differentiate between malignant and benign code that is available. Our job is to find ways to lower the model's detection certainty to below 20% - if possible.

Users are provided with a GUI and CLI to access the functionality of the software that we will be writing. Users can upload attack source code by navigating the interface and selecting the attack type and detection model. This is fed into the instruction insertion logic, which generates the attack code, but is modified to attempt to avoid the selected machine learning model's detection. Running that attack with the program would create statistics about the attack, including data leaked, certainty, and leakage rate.

4.3.2.2 User Interface

We provide clients with a GUI as well as a CLI, both with the same functionality. Clients can interact with our application through it. They can input models and attack types/codes to receive, and output, which is displayed after the machine learning model analyzes it. This is output to the console/GUI with detection certainty, data leaked, and leakage rate.

4.3.2.3 Inputs

The inputs that are available for the client to put in consist of the attack type, attack code, and detection model. The attack source codes will either be C files or x86 assembly.

4.3.2.4 Instruction Insertion Logic

All the inputs are fed into another algorithm that is compared with the power reading dataset. This generates attack source code that is meant to subserve the selected mode. It then compiles the code and sends this into the attack logic block.

4.3.2.5 Attack Logic

After running the code, it determines whether the code was malware and outputs its certainty percentage and various statistics. This is done by reading the power usage of the computer as the attack is running and looking for any abnormalities that have already been quantified in the machine learning model.

4.3.2.6 Outputs

All this info will be outputted on the CLI or GUI for the client to view.

4.3.3 Functionality

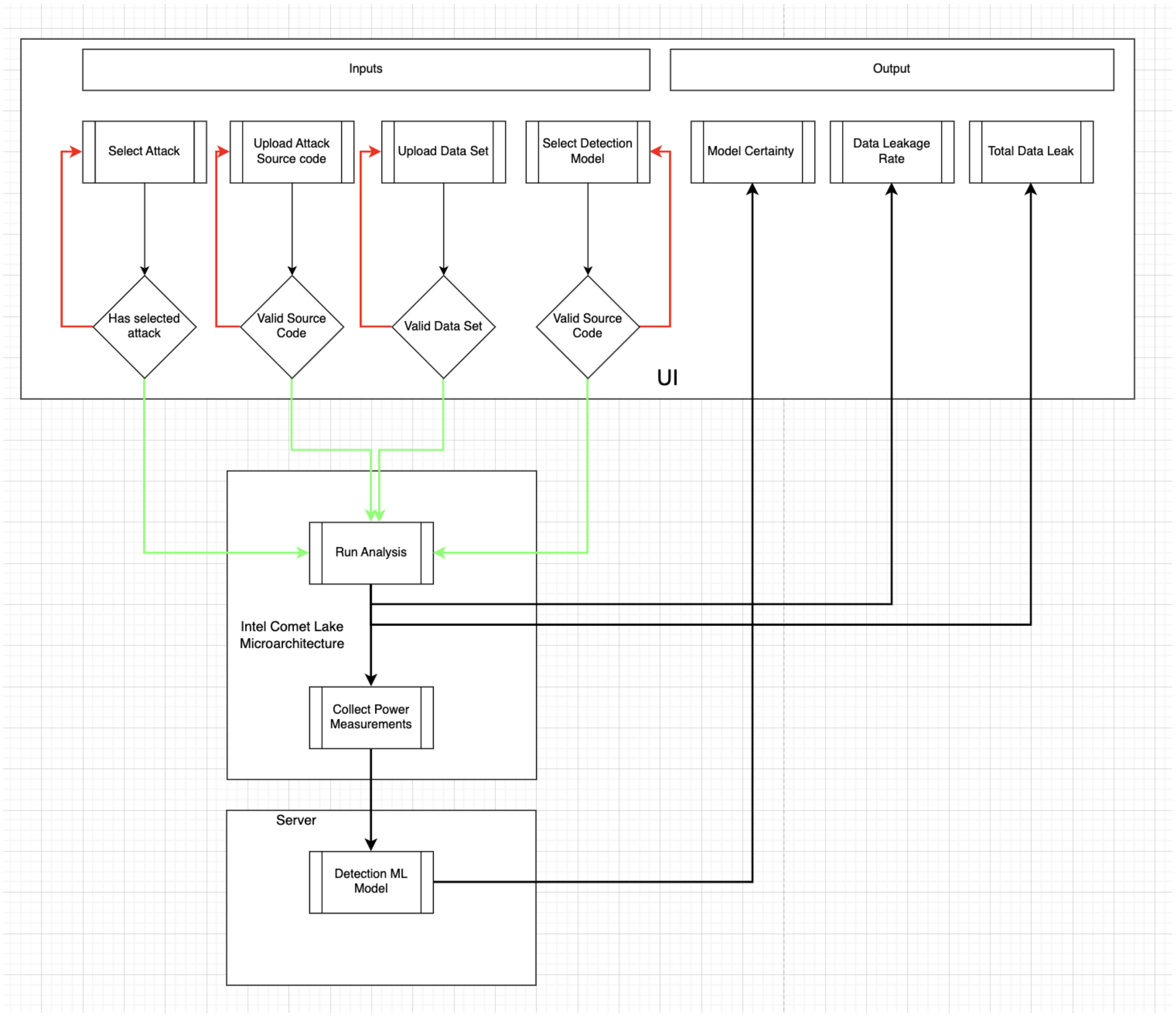


Figure 6: Functionality Graph

Our design consists of using the UI as a way for the user to interact with the system as well as to receive information. As shown in the image above the user can select and add unique inputs to the system that fits their needs. These inputs are selecting which attack the trained model is looking for, the source code of the attack the user is testing, the data set used, and lastly, selecting which trained model the user wants to test the attack on. Once the user has inputted all these paraments, then can go ahead and run the analysis.

The microarchitecture will run the code and output the rate at which the attack leaked the data as well as the total amount of data that was leaked. This information will be sent to the UI where the user can get those results. After the code was run, the system will then go ahead a collect the power measurements generated by the uploaded attack on the microarchitecture. These measurements will be sent to the server along with the data set of power measurements generated by the selected attack type. The detection model will run an analysis and output a certainty level on the match between the data set given by the user and the uploaded attack power measurements collected. If the power measurement matches the power measurements of the selected attack it would result in a higher certainty. This result would also be sent to the UI where the user can see the result of the analysis.

4.3.4 Areas of Concern and Development

Based on our design, the users are required to upload the attack source code, select a detection model, and define the attack type to run the analysis. The result must meet the project requirements, especially the detection certainty and leakage rates.

One of the concerns is that the tool might not be able to achieve the desired detection rate (20%). We will need to use different methods to optimize the ML model and allocate more time to analyzing the benign application's power consumption.

Besides that, the leakage rate may be difficult to calculate. We will need assistance from the graduate TA to help us understand better how to calculate the leakage rate and spend more time researching the attack code.

We are also concerned about whether users might upload a wrong attack source file to cause the tool to terminate or freeze. We will add some functions/error handlers to make the tool display an error message on the GUI.

4.4 Technology Considerations

Python:

- Strengths:
  - Simplicity makes it easy to use for AI/ML applications.
  - Platform independent and versatile.
  - Good library/module ecosystem.
- Weaknesses:
  - Slow execution due to Python being an interpreted language.
  - High memory consumption.
- Alternatives:
  - Scala, C/C++

Attack Code in C:

- Strengths:
  - Extremely fast execution and compilation
  - Low level language, making it easy to program machine level hardware.
  - Well used language in the team.
- Weaknesses:
  - No run time checking – all errors handled after writing the program.
  - No exception handling.
  - Manual memory management,
- Alternatives:

    - Rust

Intel x86 Assembly:

- Strengths:
    - One of the most popular ISA (instruction set architecture) today
    - Targets the devices that the attack source codes are meant for
- Weaknesses:
    - Steep learning curve
- Alternatives:
    - RISC V, ARM

4.5 Design Analysis

As the end of the semester approaches, we were able to successfully finish the milestones that we set for fall semester. We have begun implementing the GUI, pending minor adjustments, and our team is familiarized with the Ubuntu environment and attack source codes for the Spectre attack.

Current plans are still to continue analyzing power measurements and see if there is a way to make the Spectre attack more obvious and clearer.

Our proposed design is being implemented in chunks and will eventually be put together.

5 Testing

5.1 Unit Testing

Our team's software internal logic is comprised of two units, instruction insertion unit (IIU) and attack unit (AU).

Instruction Insertion Unit (IIU)

The IIU takes in an attack type, source code, and detection system and outputs an evasive attack. It's responsible for leveraging an instruction insertion algorithm and instruction power signature dataset to insert x86 instructions into the provided attack source code to create an undetectable microarchitecture attack.

Early in developing the IIU, we will first test its basic functionality of inserting instructions. We will see if, provided any arbitrary C source code, it can adequately insert the correct x86 instructions in the right location without breaking the program. This will be done using a python script which will call the insertion function with various inputs, receive the source output, compare it with the expected value, and compile and test-run the program for errors. After any changes to the unit, we will test it by running the script.

Later in the project, once the instruction power signatures have been collected and the algorithm has been developed, we will test the unit's ability to identify the instructions and locations needed to create an evasive power signature. This manual effort will require us to provide the unit with our attack codes, make the changes suggested by the unit, run the evasive attacks, and analyze the power signatures with MATLAB scripts and the detection system. We can confirm that it is working correctly by the evasive attack not getting detected and if its attack speed has not significantly slowed down.

Attack Unit (AU)

The AU is responsible for properly executing the attack and collecting statistics about the attack. The testing for this unit will be simple. Again, it will use a Python script to call the unit with various attack inputs and see if it operates as expected. We want to see that the unit runs an attack a given number of times with the given amount of space between them, and it collects the amount of data leaked and how long each attack takes to execute.

5.2 Interface Testing

User Interface

Our team's program will feature a simple and user-friendly graphical user interface (GUI). The interface will allow users to do the following tasks:

- Define the attack type
- Include the data sets they used.
- Upload the application's source code used to train the model
- Upload attack source code
- Select between different detection models
- Run evasive power-mimicking attacks

After running an attack, the GUI must display statistics about the attack, including the data leak rate in bits per second and the security systems malware detection certainty for the selected model as a percentage.

UI Testing

- Data Type Errors & Error Logging
    - We plan to test each one of the input fields and make sure each one of the methods handles data type errors correctly as well as to make sure it is logging any errors it encounters. Each method will be tested by using unit testing and testing different scenarios where different input will be passed on each method and check if the outcome is as expected.
- UI Component Testing
    - All fields, labels, buttons, and other items on the screen will be tested to make sure they are functioning as desired.
- Screen Response Testing
    - We will be checking screen controls such as colors, fonts, sizes, icons, and others to see how they respond to user's inputs. Make sure each place is easy on the eye of the user and also to provide an easy experience to the user while using the UI.
- Navigation Elements Testing
    - We will be testing navigation tools such as scroll bars and navigation bars to ensure smooth transitions while navigating the page, and refresh rate of the page.
- Output Testing

    - We will test the output given back to users to meet the requirements we set on our UI requirements.

5.3 Integration Testing

Instruction Insertion Unit / Attack Unit Integration

Integration between the IIU and AU is crucial as they make up the two parts of the internal logic. The output of the IIU directly feeds into AU, and hidden errors with the IIU might not be known until it reaches the AU. Testing this integration is very similar to testing the AU and IIU. A python script will provide the IIU with various attack codes and check for correctness. It will see if:

- The attack runs the appropriate number of times without any errors
- The correct statistics are outputted
- The attack runs undetected
- The attack speed is not significantly slowed down

UI / Internal Logic Integration

The UI / internal logic integration joins the two main parts of the application together. Testing will consist of navigating the UI, calling UI functions, and testing their behavior and outputs with expected values. Like most of the tests, this will be done with a Python script, but manual effort will be involved with navigating the UI to ensure everything is connected correctly. We expect to see the following:

- Importing source files works as expected
- Selecting a detection model successfully loads the internal machine-learning model and data sets
- Attacks successfully run with the selected type
- All statistics are shown correctly
- Each button pressed successfully calls all the necessary parts of the application.

5.4 System Testing

System testing will be very similar to the UI / internal logic integration testing, with the main difference being that we will also check if we meet all the functional requirements. These tests will use previously developed testing python scripts and manual efforts of going through each attack code and seeing if we are meeting the requirements. The system test will check to see if:

- All attack codes are executed below the detection certainty of 20%
- Power usage never exceeds 2x normal activity
- No attack exceeds a 5x slower data leak rate compared to its unaltered version

5.5 Regression Testing

Almost all tests use Python scripts to check for correctness and can be efficiently run when changes are made. Before any group member pushes their changes, they will run a master testing script that performs all developed tests to ensure nothing is broken.

5.6 Acceptance Testing

User Acceptance Testing (UAT):

Our clients will test the detection tool to determine whether it is working for the users correctly and expected.

Expected:

- Client able to download the tool and run the tool on their devices.
- All the buttons on the tool should work on client's site.
- All files can be uploaded.
- When finishing the task, the results should display on the tool.
- All the functionalities fulfil the clients' needs.
- Client able to run the tool multiple times without any unexpected errors.
- The power usage should not exceed 2x normal activity, and the attack should not surpass a 5x slower data leak rate on the user's side

Test procedure:

- Clients will download the tool on their devices.
- Client will upload different sources files to run the tool.
- Clients will use different attack types to run.
- Client will evaluate the correctness of the tool.

5.7 Results

Figure 7 maps areas of our design to each test and Table 3 shows what area each test evaluates. Testing will be crucial in ensuring that our design works and our attack modifications help to evade the machine learning model. By testing the various attacks, we know that our testing works modularly. When we eventually combine these together into the algorithm, we know that any issues that occur are due to the algorithm, not the separate attacks. This concludes that our design is useful.

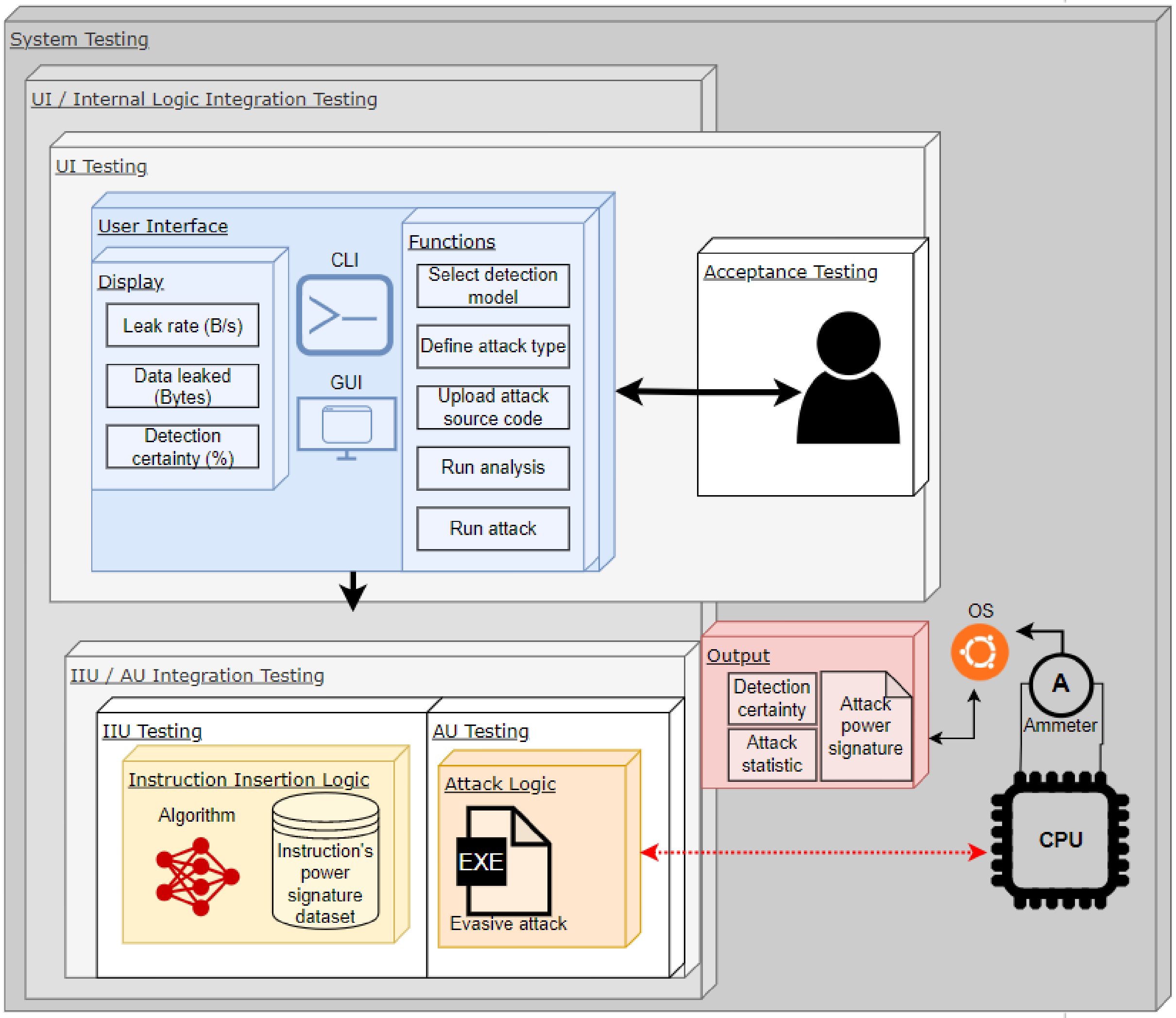
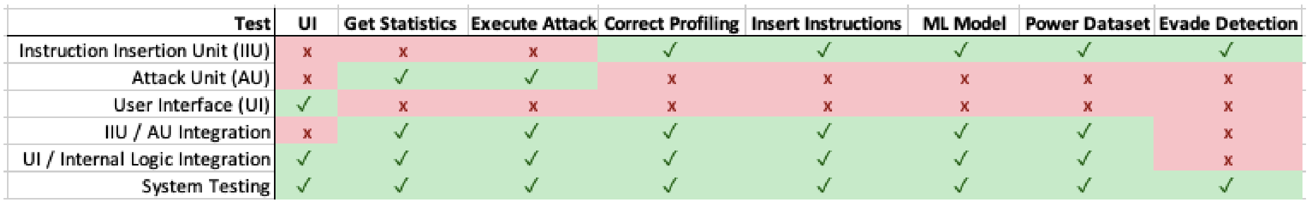


Figure 7: Testing Diagram



Test	UI	Get Statistics	Execute Attack	Correct Profiling	Insert Instructions	ML Model	Power Dataset	Evade Detection
Instruction Insertion Unit (IIU)	x	x	x	✓	✓	✓	✓	✓
Attack Unit (AU)	x	✓	✓	x	x	x	x	x
User Interface (UI)	✓	x	x	x	x	x	x	x
IIU / AU Integration	x	✓	✓	✓	✓	✓	✓	x
UI / Internal Logic Integration	✓	✓	✓	✓	✓	✓	✓	x
System Testing	✓	✓	✓	✓	✓	✓	✓	✓

Table 3: Testing Coverage

6 Implementation

Our team has taken the initiative and we have begun implementing the UI as well as getting familiar with the attack codes and the testing environment this semester. We decided to split the team into two groups to focus on the two main tasks we wanted to accomplish before next semester.

Group 1: Connor and Felipe

- **Task:** Implement UI with basic functionality
    - Set up python environment and UI foundation
    - Create necessary functions
    - Implement command line interface
    - Design a graphical user interface
    - Create communication between User Interface and Intel Comet Lake Microarchitecture device via SSH
    - Create communication between User Interface and detection Model server via SSH
    - Test UI

Group 2: Liam, Kevin, Eduardo, and Shi Yong

- **Task:** Become familiar with attack codes and test system
    - Understand the necessary Ubuntu scripts and test systems configuration
    - Collect power measurements/model power signatures
    - Modify attack codes to extract data leak rate & collect detection accuracy
    - Understand each attack's atomic instructions & possible areas to insert code

Since we are working with a basic UI, we estimated it could be implemented this semester. The UI would also be very beneficial for us to have done early on the development since it would automate a lot of the manual work, like collecting power signatures and well as running attacks on the environment. As for getting familiar with the attacks and system, it is very important for our team to learn everything about the provided resources by our client such as the attacks code and system environment. By learning everything about our project it would prevent future roadblocks in the implementation next semester.

Implementation Plan for Next Semester

Our first task for next semester is to analyze our power signatures, instructions' power consumption, and evasive attacks. After testing and learning more about our various attack's atomic structures, we were able to identify areas in the source code where we could insert meaningful x86 instructions. We will then be able to profile and record their effects on power consumption to then decide which x86 instructions create the most noise when analyzing the power signature graphs.

Additionally, we will need to implement our UI in a user-friendly way that allows ease of access when establishing a remote connection to our ML model server. This is achieved on our end using SSH and terminal. The GUI should securely maintain this connection while allowing the user to upload attack code, execute the attack, and collect the resulting data signatures. Once the user has finished interacting with the model and has collected the power consumption data, they may safely end the connection.

7 Professional Responsibility

7.1 Areas of Responsibility

Area of Responsibility	Definition	NSPE	IEEE
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence.	Perform services only in areas of their competence; avoid deceptive acts.	To maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations;
Financial Responsibility	Deliver products and services of realizable value and at reasonable costs.	Act for each employer or client as faithful agents or trustees.	None
Communication Honesty	Report works truthfully, without deception, and understandable to stakeholders.	Issue public statements only in an objective and truthful manner; avoid deceptive acts.	To avoid real or perceived conflicts of interest whenever possible, and to disclose them to affected parties when they do exist;
Health, Safety, and well-being	Minimize risks to safety, health, and well-being of stakeholders.	Hold paramount the safety, health, and welfare of the public.	to hold paramount the safety, health, and welfare of the public
Property Ownership	Respect property, ideas, and information of clients and others.	Act for each employer or client as faithful agents or trustees.	To seek, accept, and offer honest criticism of technical work, to acknowledge and correct errors, to be honest and realistic in stating claims or

			estimates based on available data, and to credit properly the contributions of others;
Sustainability	Protect the environment and natural resources locally and globally.	None	To strive to comply with ethical design and sustainable development practices, to protect the privacy of others, and to disclose promptly factors that might endanger the public or the environment;
Social Responsibility	Produce products and services that benefit society and communities.	Conduct themselves honorably, responsibly, ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession.	To improve the understanding by individuals and society of the capabilities and societal implications of conventional and emerging technologies, including intelligent systems;

Table 4: Area of Responsibility

Work Competence - You should only take work that you can manage based on your background and skillset. This would help improve and maintain our technical competence. IEEE and NSPE here see this area with the same scope as it tells you to focus on areas you are good at within your project.

Financial responsibility - IEEE didn't address any financial responsibility but I think we should make decisions that would benefit the customer financially but also not impact the quality of the product we are making.

Communication honesty - It is important to always communicate with our team to avoid any conflicts from developing any further and try to find a solution for them as soon as possible. IEEE and NSPE focus on different aspects of communication honesty as one is more focused on making sure teams are communicating constantly while the other focuses more talking to with the public such as customer or investors.

Health, Safety, Well-Being - Make sure all the made decisions are beneficiary to the public health and safety of others. IEEE and NSPE talk about the same idea in this area.

Property Ownership - Make sure you are willing to accept criticism when needed. Also acknowledge any error made you and be willing to fix them.

Sustainability - Make sure you are using sustainable practices that would mitigate danger to the public or the environment.

Social Responsibility - Make sure you are helping society understand the capabilities of new emerging technologies.

7.2 Project Specific Professional Responsibility Areas

Area of Responsibility	Definition	Context to our project	Performance
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence.	Work competence is applicable because our group is trying to deliver a quality project.	High- Our team has proven good work on all the reports. All deadlines have been met.
Financial Responsibility	Deliver products and services of realizable value and at reasonable costs.	There is little to no cost for the development of this project making it not applicable.	Not applicable- We haven't talked about finance in the group.
Communication Honesty	Report work truthfully, without deception, and understandable to stakeholders.	Being truthful with the client is very helpful to get more information. Our team depends on it make progress on the project.	High- We have been honest with the client and in return we get good feedback on the project.
Health, Safety, and well-being	Minimize risks to safety, health, and well-being of stakeholders.	This would be applicable since it has applications on future microarchitectures.	Medium- We are still researching the project.

Property Ownership	Respect property, ideas, and information of clients and others.	This area applicable because our project could harm the hardware of the user's device. We are careful not to cause damages.	Low- The code that could damage a device hasn't been implemented.
Sustainability	Protect the environment and natural resources locally and globally.	Our project doesn't affect the environment so it would not be applicable.	Not applicable- The project doesn't involve the environment.
Social Responsibility	Produce products and services that benefit society and communities.	This area is applicable because it is going to improve the security of future computers.	Medium- The team has completed the design but is still working on the implementation.

Table 5: Professional Responsibility Areas

7.3 Most Applicable Professional Responsibility Area

Work competence is the team's most applicable responsibility area. It is very important that everyone on the team contributes their own quality work to the project. Failure results in setting the group back, potentially missing the deadline, or delivering a poorly made product. As stated above the team has done a good job in delivering quality work and meeting deadlines. The team has been able to communicate with the client to gain better understanding

8 Closing Material

8.1 Discussion

Throughout the semester the team has been working on analyzing the attack code to see its effect on the device's power consumption. Through testing we were able to improve the method of collecting power consumption data. This modification will lower the total power used to collect the data; in addition, it made it easier to read the data graphs. The GUI for the project is also in development.

8.2 Conclusion

This past semester, our team has nearly finished the GUI for the application, as well as understanding the attack source codes and environment that we'll be working in.

The GUI is being developed using Python and the PyQt6 library. To connect the GUI to the laptop and server, we will create various Bash scripts that will handle connection to each device via SSH. The scripts will handle any request of the user: running the attack and collecting power measurements in tandem.

In terms of understanding the attack source code, our team has begun collecting power measurements. Using the power measurements collected, our team is utilizing MATLAB to graph the data. By doing so, we can understand the behavior of the laptop when benign code is running vs malicious code. To do so, the team is modifying the original source code by inserting various x86 instructions and comparing it with the original attack. This will provide us with a baseline of figuring out what instructions most mask the attack.

The team has a couple of goals for this project: finish documentation and design of our product, begin implementation, and create a working GUI to supplement the product.

To best achieve our goals, we decided on having team weekly meetings in addition to our weekly meetings with our client. This was necessary to help us stay on track and on the right path at times during the development of our project. This would help our team from running into obstacles during our design and implementation process.

Initially, our team struggled to understand the full scope of our project. This prevented us from starting development and design for a couple of weeks. After meeting with our advisor multiple times, we have all gained a good understanding of the project. As we did this past semester, asking questions and having a clear line of communication is key to a successful project.

8.3 References

Research Papers

S. Wei, A. Aysu, M. Orshansky, A. Gerstlauer and M. Tiwari, "Using Power-Anomalies to Counter Evasive Micro-Architectural Attacks in Embedded Systems," 2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 2019, pp. 111-120, doi: 10.1109/HST.2019.8740838.

P. Kocher et al., "Spectre Attacks: Exploiting Speculative Execution," 2019 IEEE Symposium on Security and Privacy (SP), 2019, pp. 1-19, doi: 10.1109/SP.2019.00002.

8.4 Appendix

8.4.1 Team Contract

Team Members:

- Shi Yong Goh
- Connor McLoud
- Felipe Bautista Salamanca

- Kevin Lin
- Eduardo Robles
- Liam Anderson

Team Procedures

1. Day, time, and location (face-to-face or virtual) for regular team meetings:
    a. We will meet twice weekly. Sunday virtually through a Webex meeting at 2:00 PM and Wednesday in person at 4:15 PM.
2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):
    a. For communication, we utilize email, Discord, and Webex.
3. 3. Decision-making policy (e.g., consensus, majority vote):
    a. The majority vote system will be used to make decisions when necessary.
4. 4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):
    a. After every meeting one team member will be responsible for recording minutes spent in the meeting.

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:
    a. All members are expected to show up to meetings on time unless prior notice has been given.
2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:
    a. Team members are responsible for delivering their assignments by the specified deadline.
3. Expected level of communication with other team members:
    a. All team members are required to always communicate any project related updates with the rest of the team.
4. Expected level of commitment to team decisions and tasks:
    a. Team members should be responsible for their decided tasks and commit to them unless in case of unforeseen circumstances.

Leadership

1. Leadership roles:
    a. **Client Interaction:** Shi Yong
    b. **Team Organization:** Kevin
    c. **Testing:** Felipe Bautista Salamanca, Connor McLoud
    d. **Individual Component Design:** Eduardo Robles, Liam Anderson
2. Strategies for supporting and guiding the work of all team members:
    a. Participating in the discussions.
    b. Friendly check-in on status.
3. Strategies for recognizing the contributions of all team members:
    a. Discussion during weekly meetings(check-ins). Things done the prior week, things to do the upcoming week, etc.

Collaboration and Inclusion

1. Skills, expertise, and unique perspectives:
    a. **Shi Yong Goh:** Python, C programming, Java, C#, Android Studio, Linux/Unix, Applied Math to program, Debugging skill.
    b. **Connor McLoud:** Unix, Shell scripting, Software Testing, C, C#, C++, Java, JavaScript, Python, Databases, Machine Processing, Graphic Design Programming, Network Security, Operating Systems, Android, Algorithms
    c. **Felipe Bautista Salamanca:** C, C++, Python, Java, JavaScript, SQL, mySQL, Ladder Programing, Scripting, Data Science, Linux/Unix, Mobile Development, Software Testing
    d. **Kevin Lin:** Linux/Unix, Artificial Intelligence, Python, Java, Spring, SQL, C/C++, Scripting.
    e. **Eduardo Robles:** Java, C, Python, Unix
    f. **Liam Anderson:** Unix, Java, JavaScript, C, Python, VHDL, Network security, software security, penetration testing
2. Strategies for encouraging and supporting contributions and ideas from all team members:
    a. Discussing with our advisor, allowing everyone to speak up at group meetings and have input.
3. Procedures for identifying and resolving collaboration or inclusion issues:
    a. In the case of the team environment obstructing their ability to contribute, team members are encouraged to relay their concerns in Discord or in weekly meetings. Team members can reach out to the advisor or Dr. Fila with concerns if necessary.

Goal setting, Planning, and Execution

1. Team goals for this semester:
    a. Our team's goal for this semester is to finish documenting and designing a tool that can create and analyze artificial noise and begin implementing this tool.
2. Strategies for planning and assigning individual and teamwork:
    a. To help with planning and organizing project tasks we will use Notion to assign and keep track of weekly tasks and progression.
3. Strategies for keeping on task:
    a. To stay on talk we will use our weekly meetings as a tool to check on everyone's progression. Assigned tasks should be finished or otherwise communicated.

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?
    a. We will handle infractions by having a team discussion and attempt to solve the issue as a team first.
2. What will your team do if the infractions continue?
    a. If we are unable to solve the issue ourselves, we will escalate it to Dr. Fila/advisor for assistance.

****

a) I participated in formulating the standards, roles, and procedures as stated in this contract.

b) I understand that I am obligated to abide by these terms and conditions.

c) I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.

1) Connor McLoud	DATE: 9/18/2022
2) Liam Anderson	DATE: 9/18/2022
3) Felipe Bautista Salamanca	DATE: 9/18/2022
4) Eduardo Robles	DATE: 9/18/2022
5) Shi Yong Goh	DATE: 9/18/2022
6) Kevin Lin	DATE: 9/18/2022