

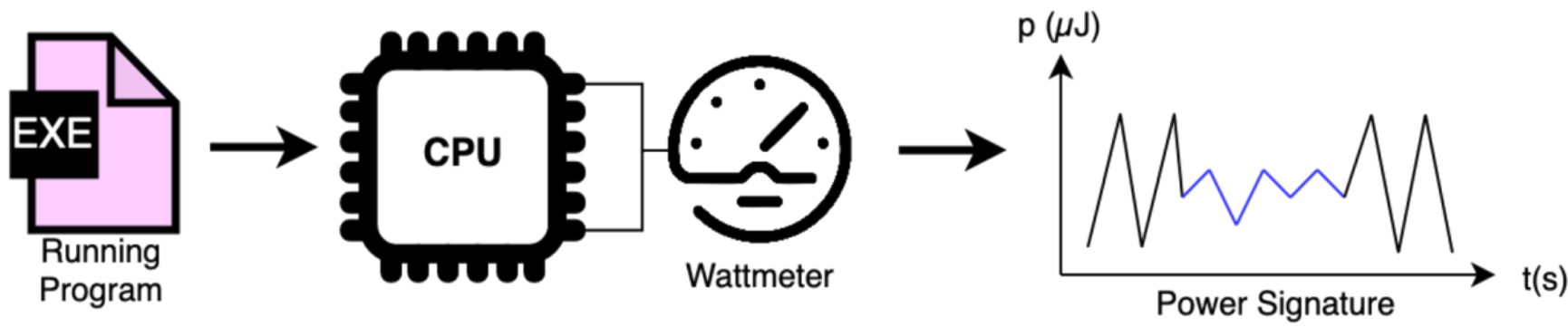
Robustness of Microarchitecture Attacks/Malware Detection Tools against Adversarial Artificial Intelligence Attacks

Group: 16

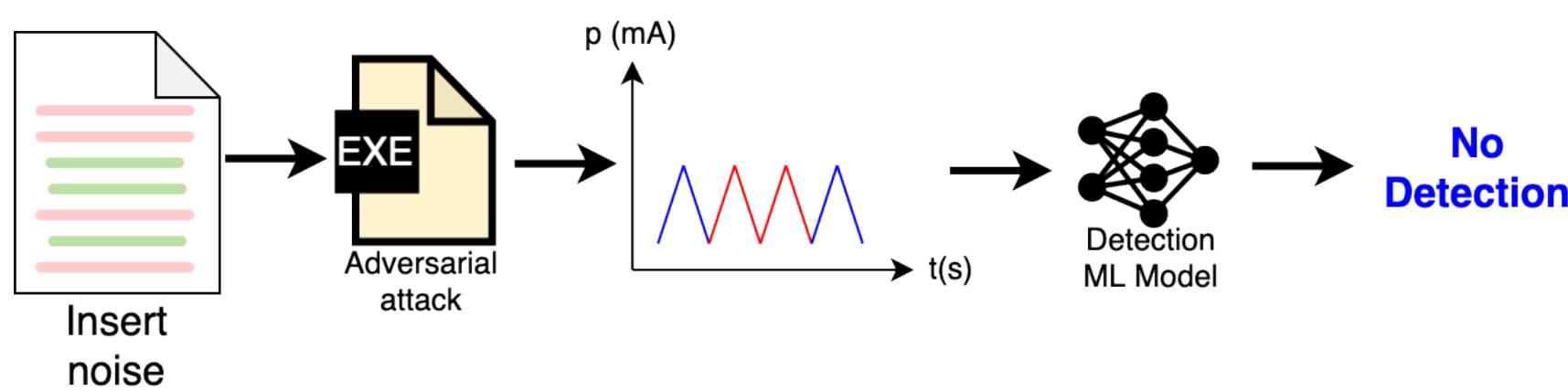
Shi Yong Goh, Connor McCloud, Felipe Bautista Salamanca, Kevin Lin, Liam Anderson, Eduardo Robles

Introduction

- Malware exploiting underlying hardware implementation, or **microarchitecture attacks**, are tough to detect with traditional anti-malware solutions.
- One detection method involves using an AI to monitor CPU power consumption for malicious patterns of energy consumption called **power signatures** [1].



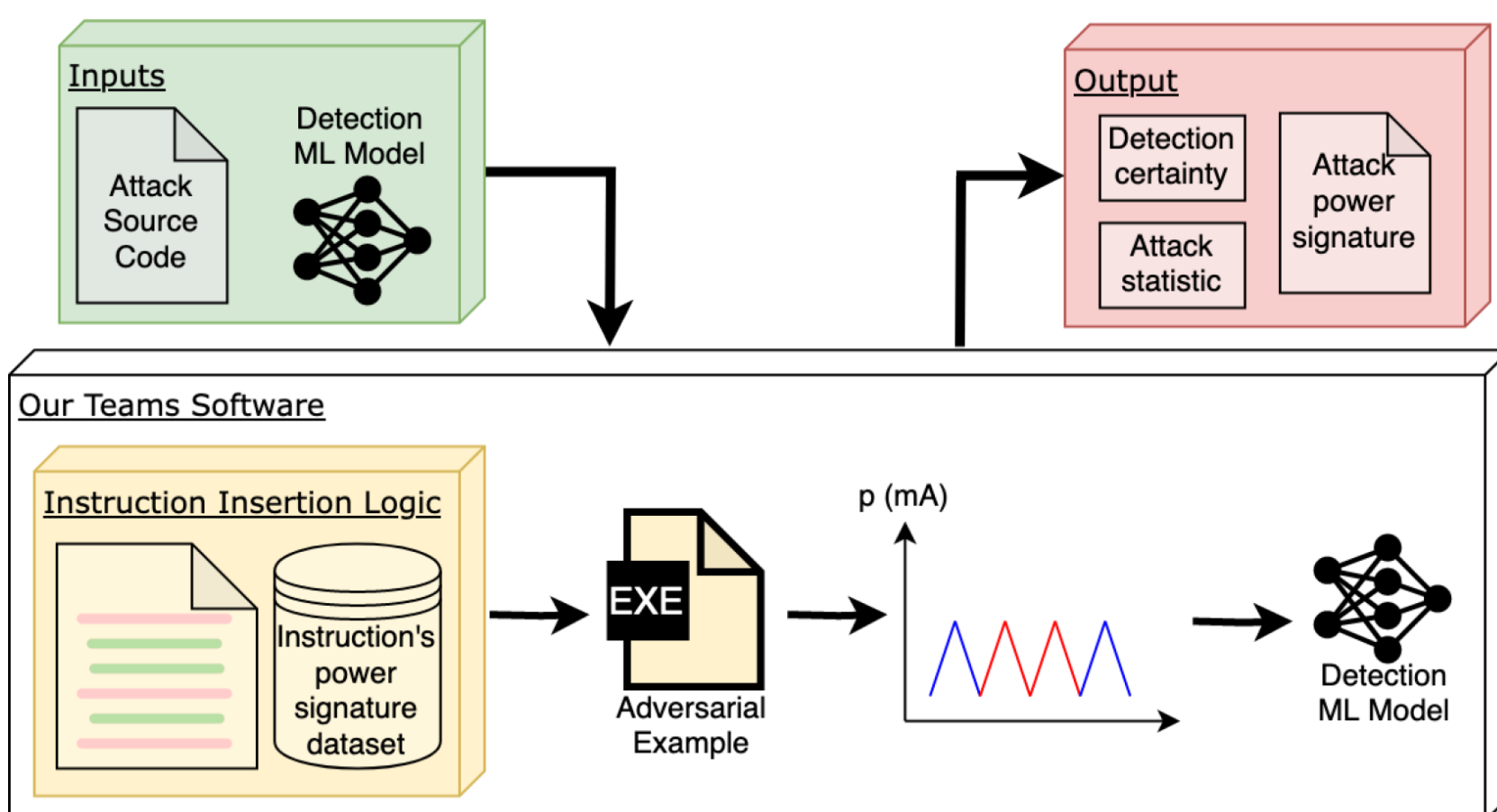
- Detection systems need to be resistant to evasive malware that uses **adversarial examples** to alter power consumption patterns and evade detection.



- We want to design software that will test these system's robustness with adversarial examples.

Overview

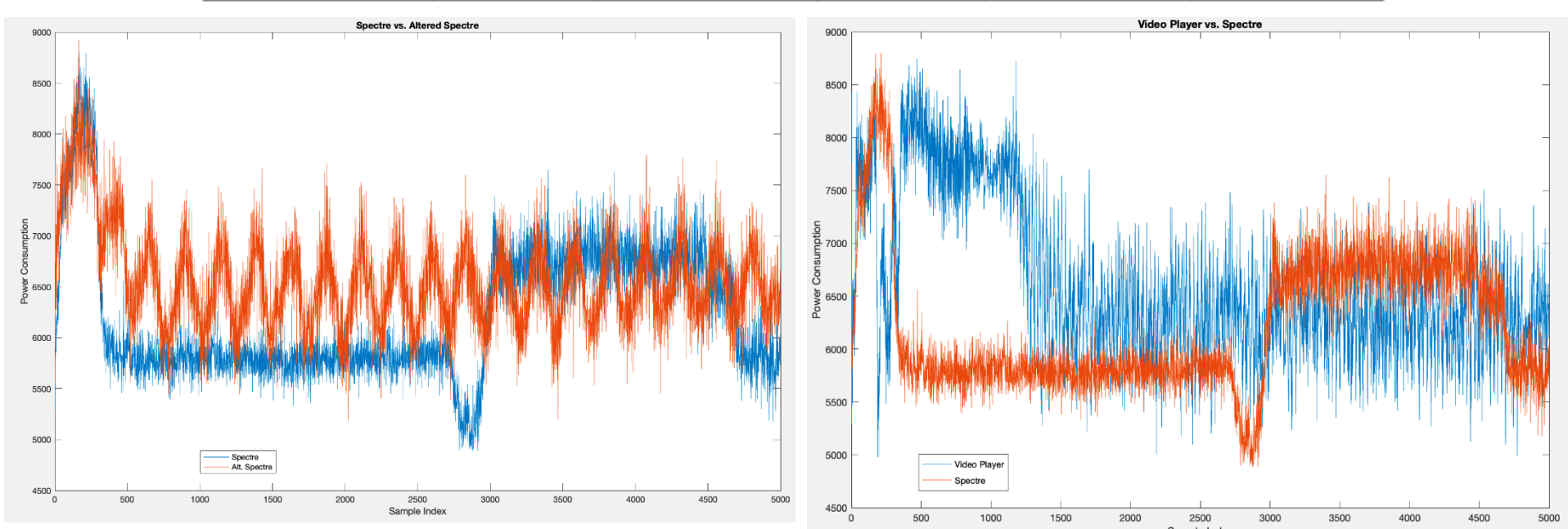
- Our software will help automate the current manual process of assessing the robustness of malware detection systems.
- Features a simple and user-friendly graphical user interface
- It can be used by researchers or developers to efficiently generate adversarial examples and test the robustness of their detection system.



Results and Impact

- Achieved an original goal to create a time-efficient adversarial example that can fool the ML model.
- Progressed an instruction power signature dataset with data on over 140 x86 instructions.
- Developed automation systems for fast testing and data collecting.
- Developed a GUI and Python environment that enables user-friendly model testing with attack codes.
- Work serves as a foundation for future teams.

Instruction	Time (ms)	Instructions (M)	Avg. Power	# Power Values	Inst. / ms (K)
add i r16	458.31	450	7400.04	3917	981.86
add i r32	223.73	450	8493.38	1644	2011.33
add i r64	267.95	450	8828.76	1923	1679.41
dec r64	136.29	450	7656.22	1162	3301.88
cmp i r64	272.54	450	8616.44	1967	1651.16
imul i r64 r64	274.20	450	8760.30	2008	1641.11
mulx r32 r32 m	241.22	960	8382.29	2415	3979.69
mov r64 m	231.46	153	6878.65	2308	661.01
mov r64 r64	259.51	2700	7791.87	2576	10404.09
prefetchw	807.28	200	7479.29	8294	247.74
rdtsc	898.44	136	8112.16	9336	151.37
sleep	1001.32	0	6081.71	9826	N/A
spectre	807.91	0	5939.56	8319	N/A



Methodology

Step 1: Learn attack codes for insight into limits and requirements

- Each microarchitecture attack has its own structure and set of instructions that can't be altered.
- A deep understanding of the attack is crucial so **noise** (x86 instructions) can be inserted in optimal positions.

Step 2: Create scripts and configure systems for efficient data collection/analysis

- Configure servers for diverse operations.
- A program that can efficiently interact with these machines to collect and analyze data is necessary for both the backend side of the project for further experiments and research.

Step 3: Implement user interface

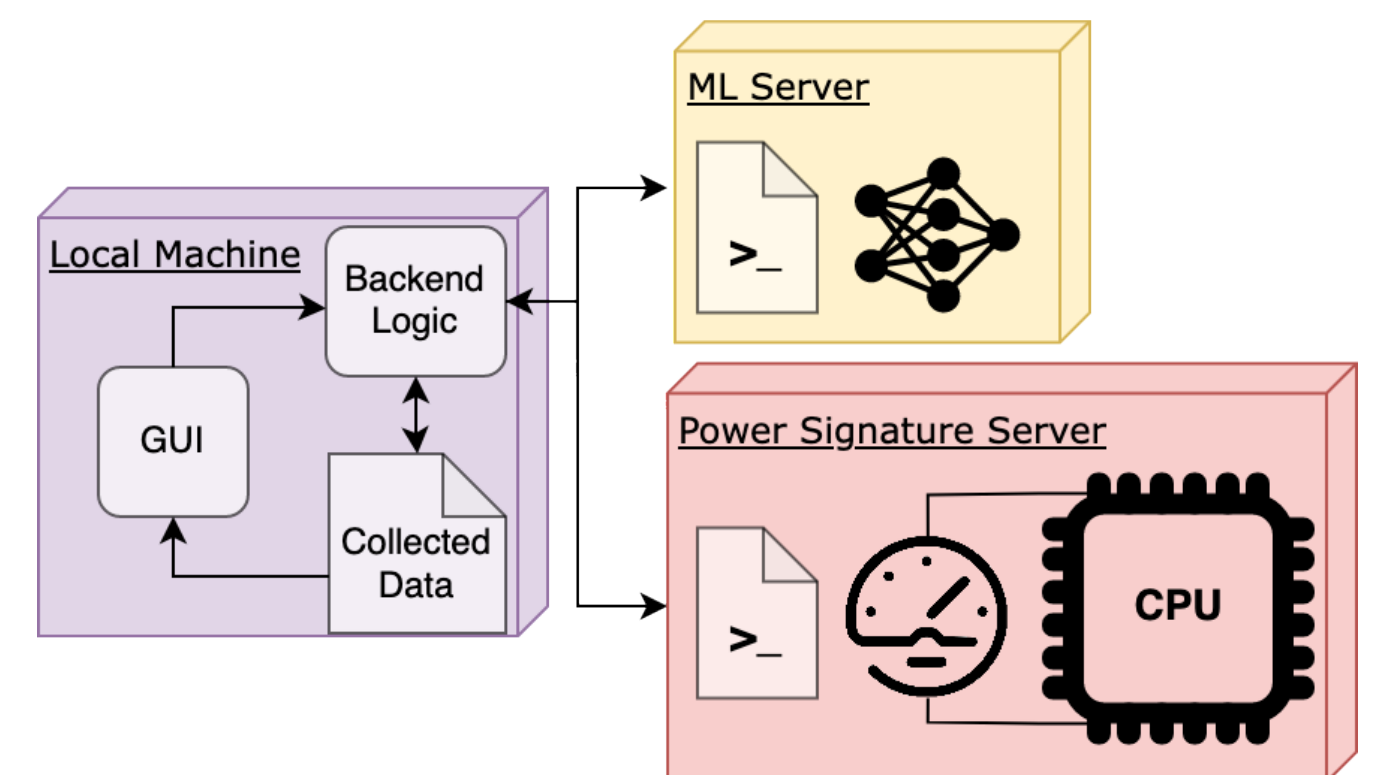
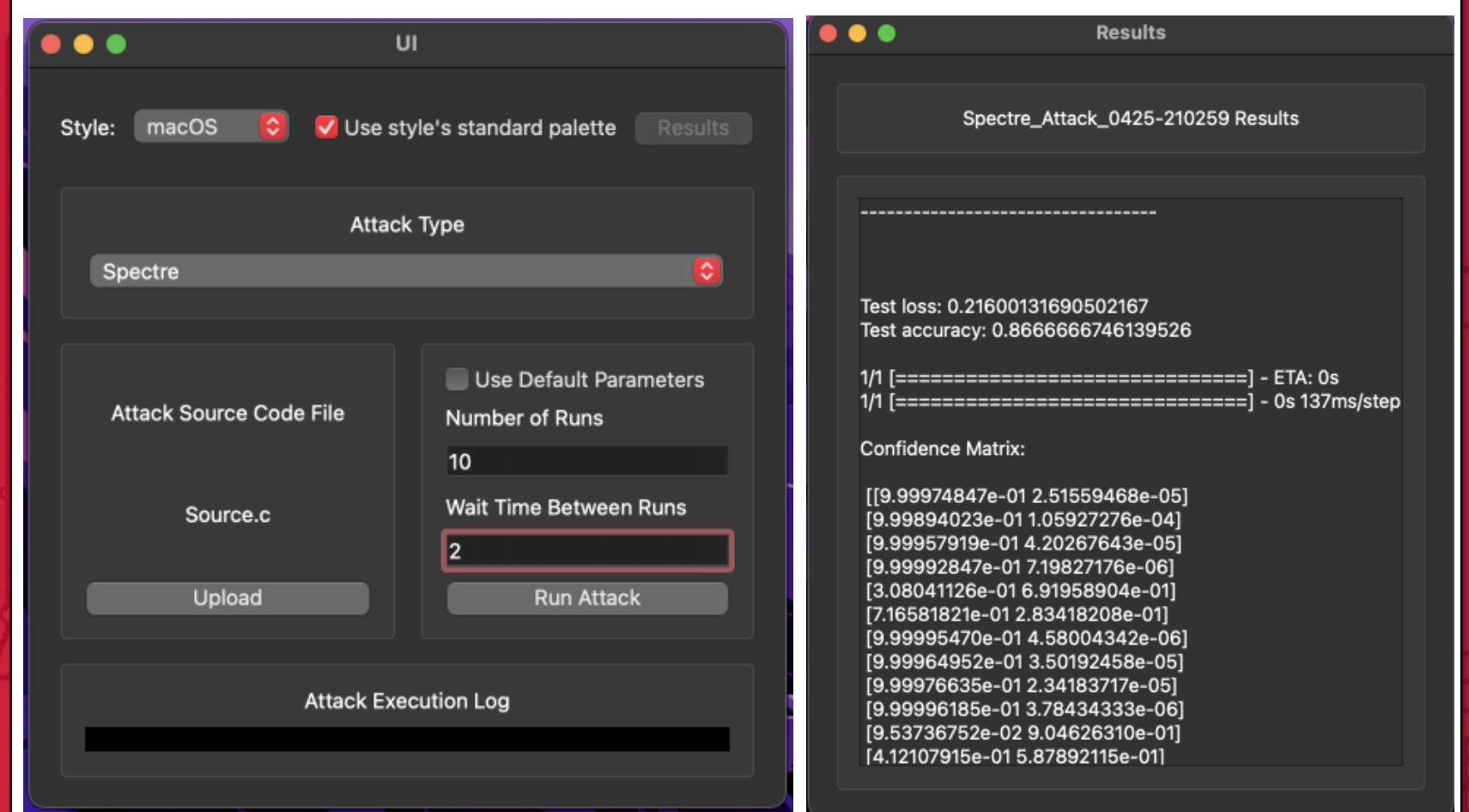
- A graphical user interface built on top of the underlying systems provides a user-friendly way to launch tests and view the results in a convenient format.

Step 4: Profile instruction's power consumption and changes on power signatures

- We must find out how each x86 instruction affects the power consumption so we can freely manipulate the power signature with precise **noise**.
- It requires research and experimentation to understand how instructions correlate to the power signature.

Implementation

- The GUI is implemented using python and PyQt6 GUI Framework. As well as internal logic on the local machine
- The backend logic uses SSH to establish communication channels with the machine learning (ML) and power signature server to collect and analyze data
- The servers are configured with Bash scripts that provide the following functionality
 - Test an attack's ability to fool the ML model
 - Collect statistics about an attack
 - Profile an instruction's power consumption



Conclusion

- UI successfully automated the old system that enabled users to test adversarial examples on the detection ML model.
- Manually created adversarial examples that exposed security vulnerabilities in the ML model.
- Advanced the effort towards comprehensively understanding the effects of x86 instructions on power signatures.
- Established groundwork for future team progress.