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

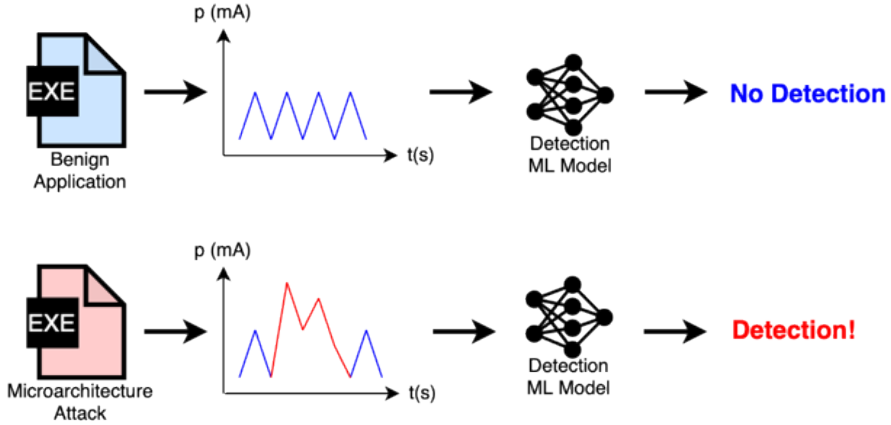
Group: 16

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

IOWA STATE UNIVERSITY

Introduction

Introduction



Note: Power is measured with Intel RAPL as microjoules, not milliamps

Power Signature: An application's overall power consumption – used to identify behavior.

Detection Machine Learning Model: A machine learning model that analyzes power signatures to classify/"detect" attacks. Made available to us via our advisor, Dr. Gulmezoglu.

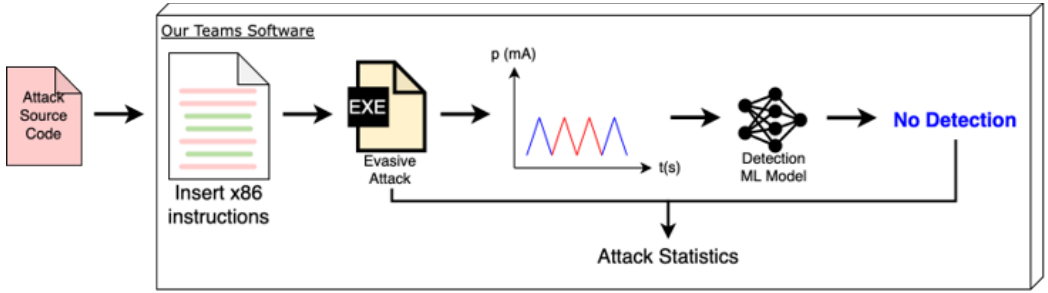
Adversarial Example: Specially crafted code that is designed to cause the ML model to misclassify the code as benign.

Dipta, D. R., & Gulmezoglu, B. (2022). MAD-EN: Microarchitectural Attack Detection through System-wide Energy Consumption. arXiv preprint arXiv:2206.00101.

Objective

Develop software to generate adversarial attacks that mimic benign power signatures in order to deceive the given Detection Machine Learning Model

- The adversarial attacks will contain extraneous x86 Assembly code
- The reasoning for this is to generate enough "noise" to confuse the ML model in hopes that the ML model classifies the attack as benign
- We want to keep the attack within 2x the normal power usage and not surpass a 5x slower data leakage rate.



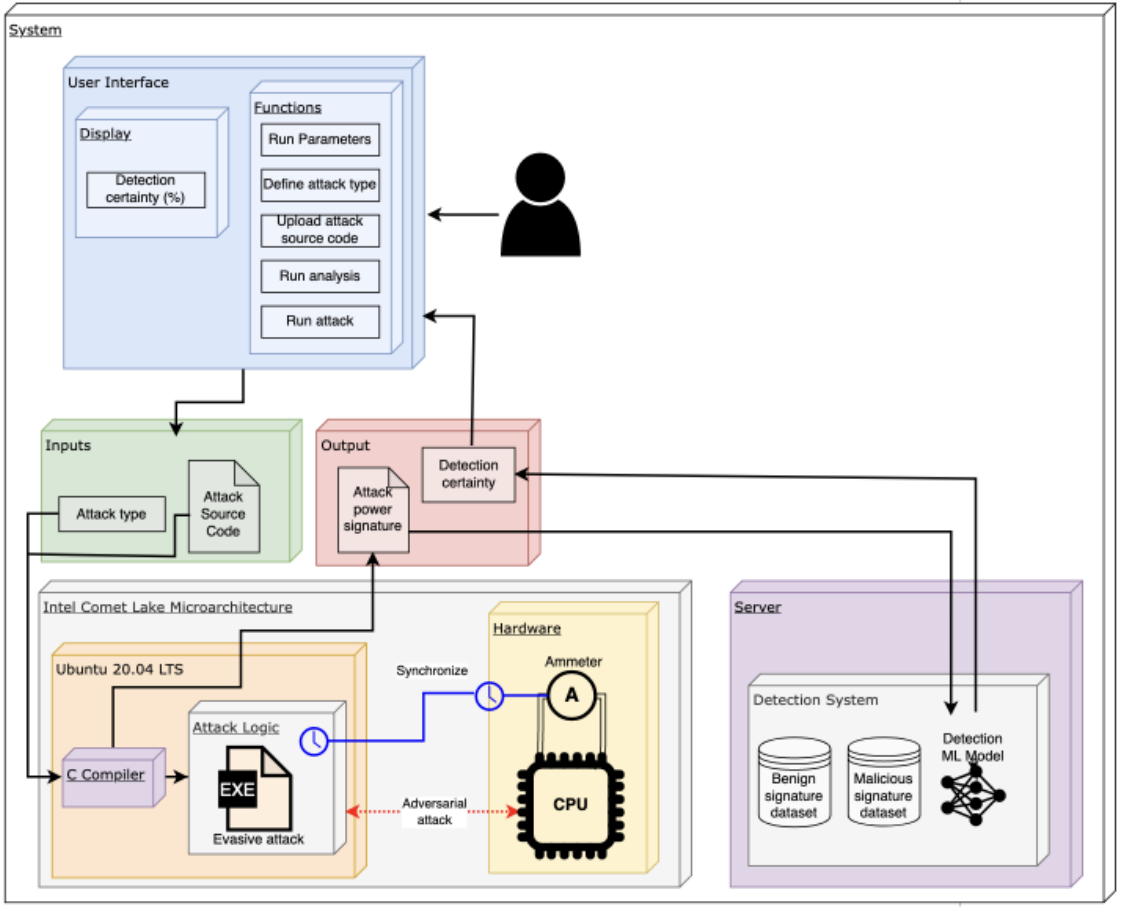
Dipta, D. R., & Gulmezoglu, B. (2022). MAD-EN: Microarchitectural Attack Detection through System-wide Energy Consumption. arXiv preprint arXiv:2206.00101.

System Architecture

Components

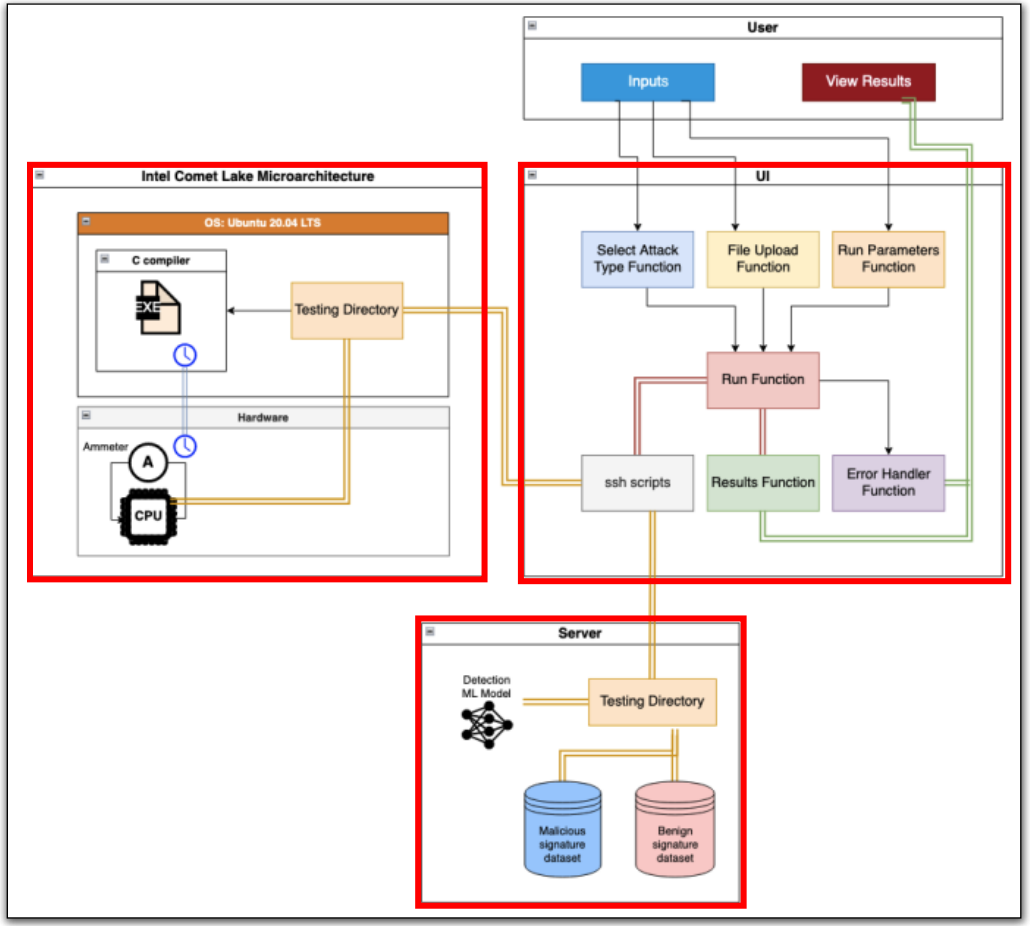
- User Interface
  - Python
  - PyQt6 Framework
- Intel Comet Lake Microarchitecture
  - CPU Model: Intel(R) Core (TM) i7-10610U CPU @ 1.80GHz
  - OS: Ubuntu 20.04 LTS
  - Linux Kernel: 5.11.0-46-generic
- Server (Detection ML Model)
  - Nvidia GeForce RTX 3090 GPU
  - CPU Model: Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz

System Architecture



Dipta, D. R., & Gulmezoglu, B. (2022). MAD-EN: Microarchitectural Attack Detection through System-wide Energy Consumption. arXiv preprint arXiv:2206.00101.

System Communication Diagram



Work Accomplishments

Tasks

- Become familiar with attack codes and test system
- Develop a UI to run the attack and collect power measurements
- Analyze power signatures, instruction's power consumption, and evasive attacks
- Implement basic instruction insertion and attack logic
- Finalize the project

UI

Goal: Automate the old system that enabled users to test adversarial examples on the detection ML Model

Task 2A: Set up python environment and Communication Scripts

- Server SSH connection script
- Intel Comet Lake SSH connection script

Task 2B: Design a graphical user interface

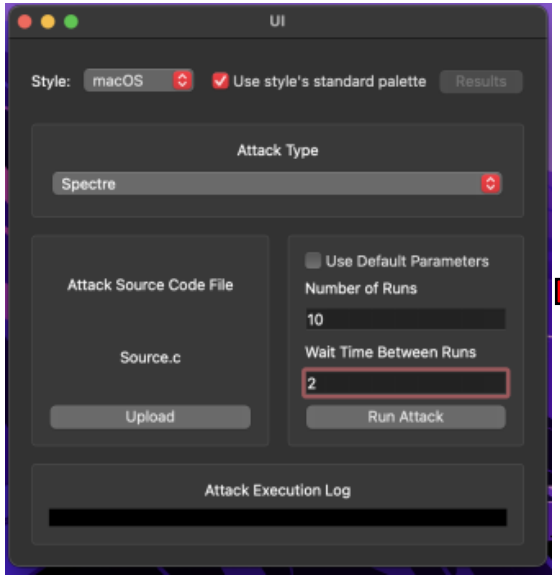
Task 2C: Implemented required features

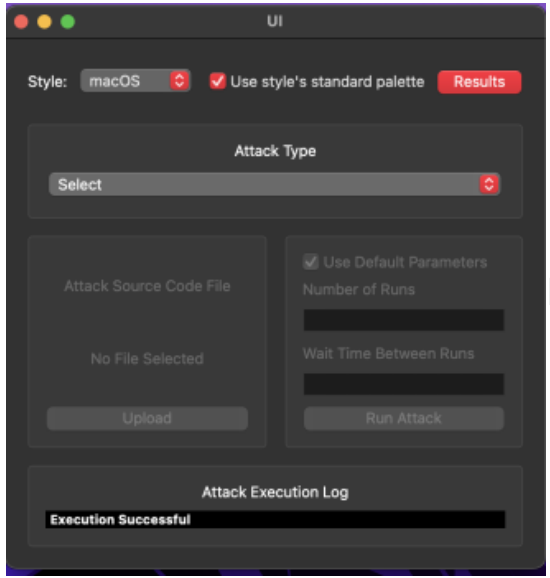
- Attack selection dropdown
- File upload
- Run button
- Log Console
- Results Window
- Error Handling

Task 2D: Test UI

- Tested with current fully implemented features

UI


UI
Style:
macOS
Use style's standard palette
Results
Attack Type
Spectre
Attack Source Code File
Source.c
Upload
Use Default Parameters
Number of Runs
10
Wait Time Between Runs
2
Run Attack
Attack Execution Log


UI
Style:
macOS
Use style's standard palette
Results
Attack Type
Select
Attack Source Code File
No File Selected
Upload
Use Default Parameters
Number of Runs
Wait Time Between Runs
Run Attack
Attack Execution Log
Execution Successful


Results
Spectre_Attack_0425-210259 Results
Test loss: 0.21600131690502167
Test accuracy: 0.8666666746139526
1/1 [==============================] - ETA: 0s
1/1 [==============================] - 0s 137ms/step
Confidence Matrix:
[[9.99974847e-01 2.51559468e-05]
[9.99894023e-01 1.05927276e-04]
[9.99957919e-01 4.20267643e-05]
[9.99992847e-01 7.19827176e-06]
[3.08041126e-01 6.91958904e-01]
[7.16581821e-01 2.83418208e-01]
[9.99995470e-01 4.58004342e-06]
[9.99964952e-01 3.50192458e-05]
[9.99976635e-01 2.34183717e-05]
[9.99996185e-01 3.78434333e-06]
[9.53736752e-02 9.04626310e-01]
[4.12107915e-01 5.87892115e-01]

X86 Instructions Experiment

- Goal:
  - Create a time-efficient adversarial example that can fool the ML model.
  - Find the instructions that consume more power during execution and the instructions has less execution time
- Used **Intel RAPL(Running Average Power Limit)** to collect the estimate of the energy(microjoules)
- Developed automated systems for fast testing and data collection.
  - Found out all the executable instructions from Intel Skylake instruction table
  - Profiled the instructions
- Put some instructions as noise into the attack code and successfully evade the detection

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

Figure: Table of individual instructions' power consumpt

X86 Instructions Experiment

- Faddps & flp instructions (floating point instructions)
  - Only instruction
  - Use "rept." to repeat the instruction in the attack
    - Accuracy 0%
      - At least 200000 number of loops
      - Around 1.03 times of the original attack code's execution time
    - Took longer time to compile
    - The power consumption is around 250uJ increase (as shown on the figure)

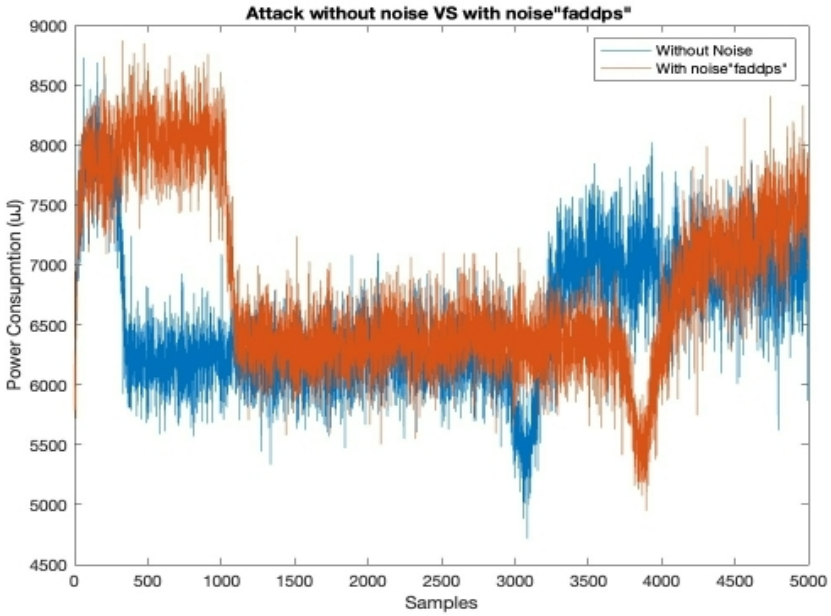


Figure: Power Consumption Comparison (faddps&flp)

Sleep function as noise

- nanosleep() & usleep()
  - Two times slower than the original attack code
  - Used longer time to execute
  - Might alter the power signature of the system

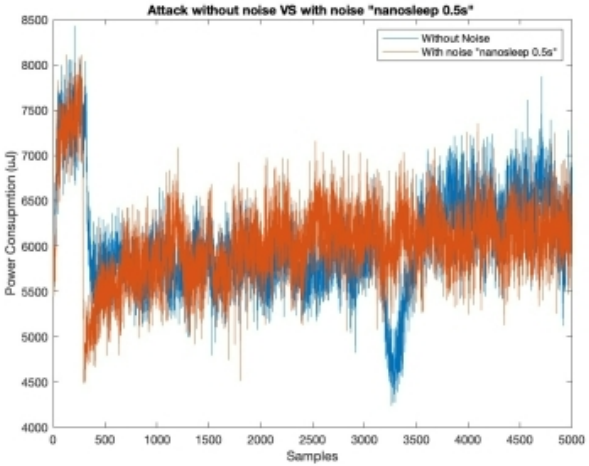


Figure: Power Consumption Comparison (nanosleep 0.5s)

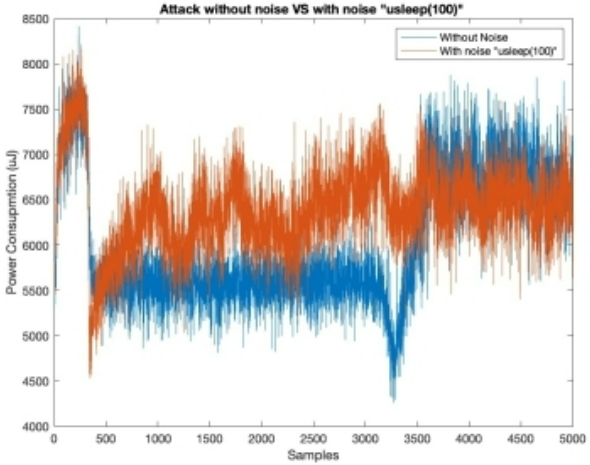


Figure: Power Consumption Comparison (usleep 100)

Key Contributions

- Shi Yong Goh
  - Experimented with the x86 instruction and analysis power signatures
- Connor McLoud
  - UI Error Handling and development
- Felipe Bautista Salamanca
  - UI design and development
- Kevin Lin
  - Instruction profiling with x86 instructions, analysis and comparison of power signatures
- Liam Anderson
  - Develop data collection and analyzing scripts. Progressed instruction profiling
- Eduardo Robles
  - Tested multiple x86 instructions with the attack code

Challenges and Solutions

Data Collection

- Collecting the power measurements using the built-in tool (Intel RAPL) required sudo permissions.
    - The team authorized personal SSH keys for each team member on the test machine and server

UI

- SSH connection scripts halts the UI until its done executing. Makes real-time error detection during SSH communication impossible.
    - Log execution output generated during script execution. Scan the log file and identify errors that may have occurred and reveal them to the user.

Challenges and Solutions

Finding Relevant x86 Instructions

- Many of the chosen x86 instructions would not compile our attacks into binaries
    - The team changed from using the Intel x86 syntax to uniformly using the AT&T syntax

ML Model Training Issues

- After initially gaining access to the server where the ML model was hosted, the team discovered that adding a plethora of instructions did not affect the model's detection rate at all.
    - Our team was collecting data during the compilation of binaries – we discovered that the model was trained without this in mind (which caused the discrepancy). Retraining the model with the compilation fixed this issue.

Future Work

Since this is a project that will continue through the upcoming semesters, there are a couple things that could be done to extend the project:

1. Addition of more adversarial attacks (including Rowhammer, Portsmash, etc)
2. Development of algorithm to automatically generate and insert x86 instructions into code
3. Better GUI display of the results of the attack (showing leakage rate, detection certainty)

Conclusion

- Going into the year, the objective of this project was a create a tool to create adversarial examples that can evade detection from a given ML model.
- Over the past year, the team has:
  - Developed a UI to make execution of attacks and testing of adversarial examples more intuitive.
  - Created a multitude of backend scripts that automate the process of running the attacks on the test machine and against the machine learning server.
  - Profiled various x86 instructions to better understand their individual power draw and effects on the Spectre attack.
  - Manually created adversarial examples that exposed security vulnerabilities in the ML model.
  - Established groundwork for future team progress.
- Future Senior Design groups may be able to extend this project by adding other side channel attacks and expanding on the work already done for the Spectre attack.