Introduction

Literature Review

Some selected literatures indicated below are closely related to the problem that was addressed in our work

1. ROP detection via HPC and Machine Learning Techniques:

Most existing ROP detection were done via HPC combined with HPC signature examine or ML techniques. Low level embedded device ROP detection works are less common than ones done on full-scale x86 or ARM processors. In recent publications, Omotosho's paper used HPC with ML model to detect possible ROP/JOP attacks on a Xtensa platform [5] and Koranek's team uses RISC-V's trace data to detect ROP/JOP attacks [6]. Their works indeed provide good framework on various embedded device, but the scope is quite limited as either works concern only with platform-specific traits such as Xtensa's F1-F27 HPCs or QEMU execution tracer for RISC-V. The inability to do ROP detection across platform shows its weakness for other ROP-vulnerable platforms like TIVA-C, which is based on CORTEX-M. Weidler et al demonstrated the possibility to create a Turing-Complete ROP gadget on a resource-constrained TIVA-C (TM4C123GH6PM) device [4] yet it is impossible to use HPC on Xtensa to do detections on such device. In addition, Dhavelle's CR-Spectre in 2022 proposed a defense-Aware ROP attack that can bypass many HPC detection mechanisms by contaminating device's HPCs to degrade detection performance [7]. Although Dhavelle's work targets a OS-based machine, the concepts can be applied to embedded devices and thus posts a threat to the existing HPC detection methods. Finally, a novel approach of using code-mutation for ROP prevention was introduced in 2022 [11] where researchers permute function memory regions in MCU's flash to increase

randomness for firmwares. The prevention technique works at large scale, but will fail for a particular MCU in preventing ROP/JOP attacks. Hence, a timely quick detection technique for multiple platforms at run-time is needed to address the ROP/JOP attack issues for embedded devices.

2. Side Channel Power Analysis:

Researches on side channel power analysis concerns mostly with attacking techniques than defense ones. Kocher et al introduced differential power analysis in early 2000 and shows the possibility to use power pattern's differences for AES secret inferences [8]. Later works by Ramezanpour introduces SCAUL, a combination of Side Channel Power Analysis and unsupervised learning to use auto-encoder features to improve key extraction efficiency on AES targets [9]. The ML and Side Channel Power Analysis on attacks serve as inspiration for many later works that applies such techniques on the defense side. However, the additional ML or just Differential Power Analysis still suffers from large overheads in vector computations. The attack model concerns less about time-critical nature of certain embedded programs as most data are analyzed long time after its collection.

3. Power Analysis for attack detection:

Many works investigate the possibility of using side-channel power analysis as a defense mechanism emerge later after 2010. Works were initially done on using side channel power analysis techniques to detect Hardware Trojans—a permeant hardware modification to the IC for malicious purposes—based on statistical calculation on power dissipation during a program's

static and dynamic switching phase [1]. The idea of Trojan Attack detection provides great insights to code modification detection in static scenarios, but ROP and JOP attacks are more versatile in nature. In same year, several works addresses the issue of dynamic power analysis for code injection or ROP/JOP attack detections. Liu et al proposed the code execution tracking model as a state machine and they uses power traces to construct a sequence graph of execution with likelihood estimation for such execution sequence [2]. Their work, however, lacks practicability during a program's run time with various possible inputs. The computation overheads to calculate and estimate attacks is bigger and requires the analysis model to operate on an extremely fast and efficient machine. In Bai et al's work, the group created a platform for side-channel monitoring implemented a SVM model for attack-trace detection [3]. But their work fails to investigate the complexity of using ML to do such detection during run-time. In addition, they fail to show the possible tradeoffs for this side-channel detection technique. In our paper, we addresses both issues by not only showing the side-channel power analysis' ability to detect ROP/JOP attacks without employing ML techniques but also outlines the tradeoffs for such techniques during a program's run time.

Model Framework

Overview of the Side Channel Power Analysis

Commonly used as an attacking mechanism, Side Channel Power Analysis aims to gain program's execution insights via its CPU's power consumption over time [10, 12]. At the most basic level, CPU and memories are made up of transistors and capacitors, respectively. A register or a memory block holding a bit 1 requires more power driven than that holding a bit 0, where no power flows through the transistor [10]. In an AES attacking scenario, different S-Box slot lookup yields minimal yet discernible power consumption differences at different memory locations, which provides the basis for differential power analysis.

Side Channel Power Analysis is an non-intrusive technique that is easy to setup for both defense and attacking purposes [13]. In order to measure power of the CPU, a parallel circuit can be added between MCU's Vdd pin and the power supply. To avoid shorting the circuit, the new circuit requires an additional resistor between two measuring points [10]. The change of voltage can be obtained through Ohm's law, where V is the voltage, I is the current, and R is the constant resistance we applied to the circuit. A circuit schema can be drawn as follow, in figure ??.

$$V = IR$$

At instruction-level, different ALU or memory operations yields different power consumption, as shown by figure 1. From the graph alone, it is obvious that the ALU operation consumes lower amount of power than stack (memory) operations. In addition, ALU operations vary from instruction to instruction, as shown by the graph of ADD and AND operation. For a larger program, which is shown in figure 2, the AES encryption process can be easily observed

via the CPU's power side channel. The Round-Key, S-BOX operation are easily discernible based on the graph alone as the former one consists mostly ALU operations whereas the latter one consists mostly memory operations.

<Graph 1, 2>

Side Channel Power Analysis For Return/Jump Anomalies

Based on the power pattern differences over a program's execution, we can monitor the execution behavior after a program executes the "return" or "jump" instruction from either a function call or if/else statement. This means that a trigger is needed from the embedded device to tell the sample collector when to start sampling. Luckily, most—if not all—embedded devices have General Purpose Input/Output (GPIO) ports that communicates with external devices. Driving the GPIO signal to 3.3 V

The experiment Setup

For the experiment, we uses the ChipWhispererLITE (CWLite CW1173 two parts) with its default XMEGA target. The CWLite has an Xilinx SPARTAN 6 chip as its main processing unit and runs on its 5.7.0 firmware version and can be interfaced with Jupyter notebook. The target board has an Atmel AVR instruction set architecture and connects to the CWLite Via its measure port—JP10—over a shunt resistor. The trigger was connected via CWLite's GPIO D pin, on both the target and the CWLite device. The testing programs are cross compiled using the Brew AVR GCC compiler. They were subsequently loaded onto the CWLite device and loaded to XMEGA target via serial port. The configuration of the setup can be found online at Github.

Programs and Inputs:

Programs for benchmarking follows the Omotosho's paper, with various known algorithms like Depth First Search, Binary Search, Minimum Spanning Trees, etc. The programs were written in C. Malicious modification is made to the programs by the addition of an extra function call that waits a global variable trigger to go up. The ROP chains were injected at either the beginning of the function call to simulate a ROP attack or after the if condition to simulate JOP attacks. The function call looks as the follow.

```
volatile short trigger=0;

void ROP_CALL(){
    if((trigger==25)){
        . . . <ROP Chains>
}
    trigger_low(); // to 'reset' the trigger for next use
    trigger++;
    return;
}
```

For each program, the ROP chain models the exact part of the main program in no more than 10 ALU instructions, written in inline AVR assembly. Before each Jump/Call/Return instruction, the hardware trigger (GPIO D) goes high to notify CWLite to sample 900 to 1200 power data points, which were later used in either the learning or the testing phase. The inputs to these benchmarks were randomly generated at each iteration via the *rand()* and other key functions from the AVR standard library. For example, in the binary search program, the sorted array were first generated by the *rand()* function and were then sorted in place using *q_sort()*. Because these are part of the program, but rather data preparation process, they are out of the scope of the potential ROP/JOP attacks and will not be measured or tested against potential ROP/JOP attacks.

Data and Pattern Comparison

The comparison of power data pattern was based on the assumption that data was sampled at the same starting point, by the GPIO trigger. In addition, because of CWLite's ability to define the length of the sample point, each data pattern should have the same length. A simple linear comparison was used to check the similarities of power patterns where each corresponding data point was compared with a "tolerance of difference" of 0.1 mV. In detail, the algorithm can be explained as follow:

```
def linear_comp(l1,l2, threthold=300, min_seq=2):
    length=min(len(l1),len(l2))
    seq=0
    counter=0
    for i in range(length):
        if(abs(l1[i]-l2[i])<threthold):
            counter+=1
        else:
            if counter>=min_seq:
```

```
            seq+=counter
            counter=0
        else:
            counter=0
    if counter>=min_seq:
        seq+=counter

    return seq
```

To determine whether or not the program was under ROP/JOP attack, the power pattern matching should be above 70%. For example, if a data sample has a length of 1200, at least 840 consecutive data point should be matching. Such comparison will usually yield accurate result as the ROP chain usually disrupts the pattern and causes all the rest data points to mismatch, which will significantly reduces the number.

Learning and Testing

Each benchmark testing consists two phases: learning and testing.

In the learning part of the experiment, target was operated under the “safe” condition for data sample gathering. Once the trigger goes up, the sampled data from the CWLite will be delivered to the host machine via its USB serial port. The sample pattern will either be stored, if no data exist, or be checked against existing samples. Should the pattern be “new”—a linear comparison that yields a score lower than 50% to any known ones—the new pattern will be added to the pattern collection. The testing phase will repeated collect samples for 120 iterations as a program

may involve different branches, return/jump instructions with conditions, etc. The set of result will be used as standards for future testing detections.

In the testing part of the experiment, 1024 samples were taken at run time. This is because the short type takes in a 8-bit value that goes up to 255 only. Throughout each 256 iteration, one ROP attack is injected at a random trigger value. In addition, inputs (arrays, graphs, strings) of the benchmarks are constantly modified by the *gen_rand()* function, and will not be subjected to reset throughout the whole testing phase. The testing pattern will be compared iteratively with all the known valid patterns until there is a match, which we conclude a "valid" execution, or no match at all, in which case we conclude a ROP attack.

In order to evaluate the method's robustness against potential false positives, we also tested our method against 1024 non-ROP attacks with random inputs at each iteration. In these iterative testings, the ROP call function will consists only the trigger++ and reset_trigger() action, as shown below. In theory, none of the testing will cause a mismatching pattern. But to allow for power glitches, the validity threshold was set to 90%.

```
volatile short trigger=0;

void ROP_CALL(){
    trigger_low(); // to 'reset' the trigger for next use
    trigger++;
    return;
}
```

Result and Discussion (data missing)

Research Questions:

Our benchmark experiments aim to answer the following research questions:

RQ1: Using side-channel power analysis techniques, what precision and recall accuracy at run-time we can achieve for detecting ROP/JOP attacks? Because ROP/JOP attacks happen rarely throughout a embedded program's extended lifecycle, it's important to detect such attack both precisely (time) and accurately (accuracy). In addition, since a potential ROP gadget may be small and highly similar to the valid program, a near-perfect detection under such condition is critical for the program's safety.

RQ2: What are the tradeoffs for using side-channel power analysis? How does these tradeoffs relates to the programs' complexity. These tradeoffs serve as good hardware implication, should the techniques be applied to a bare-metal hardware.

Benchmark Performance:

Comparing with the known HPC methods, our side-channel method has a significant accuracy advantage for security-critical embedded devices. This is especially true for programs that are more simple/recursive in nature—Binary Search, LCS, etc.

Using our method, we calculated the precision and recall rate on all the programs. The precision was calculated using $Precision = TP / (TP + FP)$ while the recall was calculated by $Recall = T P / (T P + F N)$. As the following shows, most programs yields a precision near 100% except the more sophisticated ones (prim and …). Even though these more sophisticated programs suffer from reduction in precision, the power matching score between ROP attack and false positive ones differ by XXX, which is a % difference. The following graphs shows the pattern similarities between valid patterns and ROP attack patterns, the X-axis ranks the complexity of the program in ascending order.

Tradeoffs:

The tradeoff for such accuracy comes in terms of processing time delay and memory usage for valid patterns, which goes up proportionally with the complexity for different benchmarks. In the table below we listed the complexity (in terms of O notation and # Return/ Jump branches). On the right column we also listed the approximately memory overheads (normalized) and Floating Point calculation overheads for pattern comparison.

To further demonstrate such tradeoffs from a software perspective, we modified and analyzed the minimum time delay (in terms of ALU addition operation) that needs to be supplied to a program in order for the host machine to have sufficient time to process a sampled pattern. From the table XX, the time delay needed for processing 1300 data samples increases as program complexity increases.

Hardware Implication and Implications:

The idea for this experiment is to demonstrate the possibility to add specific hardware components to analyze the side channel power pattern for an embedded device for potential ROP/JOP attacks. In this paper, we outlined the practicality for using an combination of external hardware—the Chipwhisperer Lite—and full scale computer software to accurately detect malicious execution of […]

Work Cited

1. A Side Channel Based Power Analysis Technique for Hardware Trojan Detection using Statistical Learning Approach
2. On Code Execution Tracking via Power Side-Channel
3. RASCv2: Enabling Remote Access to Side-Channels for Mission Critical and IoT Systems
4. Built-In Return-Oriented Programs in Embedded Systems and Deep Learning for Hardware Trojan Detection
5. Detecting Return-Oriented Programming on Firmware-Only Embedded Devices Using Hardware Performance Counters
6. Identification of Return-Oriented Programming Attacks Using RISC-V Instruction Trace Data
7. CR-Spectre: Defense-Aware ROP Injected Code-Reuse Based Dynamic Spectre
8. Differential Power Analysis
9. SCAUL: Power Side-Channel Analysis with Unsupervised Learning
10. Power Analysis Based Side Channel Attack
11. Code Mutation as a mean against ROP Attacks for Embedded Systems
12. SIDE-CHANNEL ATTACKS
13. S. S. Clark, et al. Wattsupdoc: Power side channels to nonintrusively discover untargeted malware on embedded medical devices. In 2013 USENIX Workshop on Health Information Technologies, HealthTech ’13, 2013.