Computers & Security 148 (2025) 104135

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

Improving IIoT security: Unveiling threats through advanced side-channel analysis

Dalin He [a], Huanyu Wang [b,*], Tuo Deng [a], Jishi Liu [a], Junnian Wang [a]

[a] School of Physics and Electronic Science, Hunan University of Science and Technology, China
[b] School of Computer Science and Engineering, Hunan University of Science and Technology, China

ARTICLE INFO

Keywords:
Side-channel analysis
Control flow monitoring
Function level intrusion
Deep learning
Correlation power analysis

ABSTRACT

The widespread deployment of IIoT edge devices makes them attractive victims for malicious activities. Consequently, how to implement trustworthy operations becomes a realistic topic in embedded systems. While most current physical systems for detecting malicious activities primarily focus on identifying known intrusion codes at the block level, they ignore that even an unnoticeable injected function can result in system-wide loss of security. In this paper, we propose a framework called CNDSW built on deep-learning side-channel analysis for function-level industrial control flow integrity monitoring. By collaboratively utilizing correlation analysis and deep-learning techniques, the dual window sliding monitoring mechanism in the proposed CNDSW framework demonstrates a real-time code intrusion tracking capacity on embedded controllers with a 99% detection accuracy on average. Instead of focusing on known block-level intrusions, we experimentally show that our model is feasible to detect function-level code intrusions without knowing the potential threat type. Besides, we further explore how different configurations of the CNDSW framework can help the monitoring process with different emphases and to which extent the model can concurrently detect multiple code intrusion activities. All our experiments are conducted on 32-bit ARM Cortex-M4 and 8-bit RISC MCUs across five different control flow programs, providing a comprehensive evaluation of the framework's capabilities.

1. Introduction

The Industrial Internet of Things (IIoT) is revolutionizing our society by introducing connectivity, automation, and data-driven decision making across various industries (Boyes et al., 2018). The IIoT edge devices, constituting control systems, play a crucial role in key infrastructures such as energy, production and healthcare. Consequently, the increasing deployment of embedded IIoT devices has made them attractive targets for malicious activities (Sengupta et al., 2020) such as code intrusion (Hemsley and Fisher, 2018) in control flows (Abadi et al., 2009) and compromise of cryptographic implementations (Mekala et al., 2023), which can definitely lead to a complete loss of security.

For example, the Stuxnet (Falliere et al., 2010) virus, discovered in 2010, was the first malware to attack real-world infrastructures, directly causing a delay in Iran's nuclear program. In 2015, the Ukrainian power sector fell victim to the BlackEnergy (F-Secure Labs, 2016) worm virus, where intruders infiltrated the monitoring and management systems, leading to widespread power outages. In 2017, the Triton (Di Pinto et al., 2018) malware specifically targeted industrial safety systems at a petrochemical plant in Saudi Arabia, manipulating the safety protocols and risking a potential catastrophic failure. These incidents underscore the urgent need for robust cybersecurity measures in the era of IIoT.

To monitor whether the operations of IIoT devices are working properly, there are two commonly applied existing approaches for detecting malicious software in embedded devices: network traffic analysis (Qin et al., 2024; Jayalaxmi et al., 2023; Rendón-Segador et al., 2023) and host-based detection methods (Liu et al., 2016; Han et al., 2017; Yang et al., 2024). However, the network traffic analysis cannot examine detailed activities within embedded devices. Besides, this approach would become ineffective when devices operate offline. Host-based Intrusion Detection Systems (IDS), on the other hand, are primarily effective on PC/x86/x86 controller systems that have sufficient resources to run the IDS. They are not suitable for direct deployment on the actual Programmable Logic Controllers (PLCs) or similar embedded units due to their limited resources and interfaces. As a result, Side-Channel Analysis (SCA), a non-intrusive monitoring approach, has become one of the most realistic and effective way for detecting malicious code intrusions in industrial edge devices. For clarity, "industrial edge devices" in this paper includes various computing hardware used for localized data processing and control in

* Corresponding author.
E-mail address: huanyu@hnust.edu.cn (H. Wang).

https://doi.org/10.1016/j.cose.2024.104135
Received 16 May 2024; Received in revised form 20 September 2024; Accepted 22 September 2024
Available online 25 September 2024

industrial settings, such as industrial gateways, embedded controllers and industrial PCs. This definition is linked to the discussion of CPU architectures later in the paper, as different industrial edge devices use varying CPU architectures, which influences the effectiveness of side-channel detection methods.

In general, SCA techniques are used by adversaries for malicious activities, such as extracting secret keys from cryptographic implementations (Wang et al., 2021). However, every advanced technology has both its dark and bright sides. Recently, numerous studies have shown that SCA holds great potential in code intrusion detection. By exploiting the side-channel leakage generated by target devices during the execution of control flows, it is feasible to check whether the instructions are perfectly executed. In 2016, Liu et al. (2016) confirmed that Control Flow Integrity (CFI) (Ligatti et al., 2005) technology can be applied to embedded systems for code execution tracking by using power consumption as the side channel. Afterwards, Han et al. (2017) utilized a pre-trained neural network to classify the electromagnetic (EM) emissions to identify malicious code intrusion. Building on the approach introduced by Han et al. (2017), Khan et al. (2019) analyzed EM emissions generated from the target device during the execution of the program to remotely detect code intrusion. Moreover, in 2022, Han et al. (2022) introduced a method allowing malware to evade physical monitors, making the defense more challenging. However, existing works typically focus on block-level code intrusions, in which they ignore the function-level intrusions can also cause significant threats to control flows. Besides, their designs are limited to known malware, which seems unrealistic in real-world scenarios.

Motivated by the aforementioned research efforts, we propose a real-time function-level code intrusion detection framework called **CNDSW** by collaboratively utilizing **C**orrelation power analysis (CPA), **N**eural networks and **D**ual-**S**liding **W**indow monitoring. This framework is designed for intrusion detection, with a focus on power analysis as our analytical method. In Fig. 1, we illustrate how power consumption traces can be generated from a 8-bit RISC MCU. In the context of SCA, the power variations of the chip during its operations can be utilized for analyzing running activities. Potential practical applications of the CNSDW framework include deployment on security-critical IIoT edge devices for warning against intrusions attacks, such as cases on nuclear infrastructure controllers and wastewater sector units. Despite its ability to detect various code intrusions on PLCs, CNSDW can be also deployed in stability-critical applications to ensure their proper functioning. This includes the deployment of a CNSDW board connected to the power supply of the computational core of smart cars to monitor driving status, enabling real-time capture of vehicle behavior in autonomous mode. In summary, our contributions can be listed as follows.

(1) Side-channel leakage analysis. We conduct extensive empirical analysis of side-channel leakage generated from instructions invoking *C* library functions on 32-bit Arm Cortex-M4.[1] and 8-bit RISC MCUs.[2] We show that it is feasible to detect *C* library functions from the power traces with a 99% detection accuracy.

(2) Introducing CNDSW framework. We propose a novel online code intrusion detection framework, CNDSW, offering meticulous analysis at the level of individual function. This approach ensures precise code integrity validation and robust detection of malicious software activities.

(3) Comprehensive evaluation. We demonstrate extensive experiments across two target devices with five different control flows to evaluate the extent to which the CNDSW framework can detect code intrusions. The implemented control flows are ***AES-128*** (Rijmen and Daemen, 2001), ***PID*** (Johnson and Moradi, 2005), ***PSD*** (Youngworth et al., 2005), ***Sensor20*** (Huang et al., 1979), and ***SensorSorting*** (Astrachan, 2003). We further explore how different configurations of the proposed CNDSW framework can help the monitoring process with different emphases and to which extent the model can concurrently detect multiple code intrusion activities.

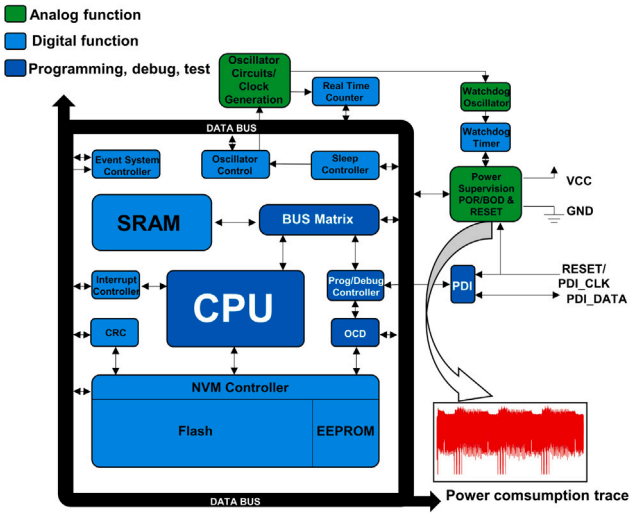


Fig. 1. An architectural diagram of an 8-bit RISC MCU chip, illustrating the layout of functional modules such as CPU, SRAM, Flash, and watchdog timer. It has been observed that significant power consumption information leaks during CPU-to-memory interactions. The power trace plot in the bottom right corner of the diagram illustrates the power variations of the chip during its operations, which can be utilized for analyzing running activities.

The remainder of this paper is structured as follows. Section 2 provides an extensive background on the subject matter. Section 3 outlines the specific detection objectives addressed in this study. Section 4 discusses the threat models. Section 5 introduces the CNDSW detection framework. Section 6 presents our experimental setup. Section 7 presents the experimental results and the corresponding analysis. Finally, Section 8 summarizes our findings and reflects on their implications, as well as recent developments in related research areas.

2. Background

This section provides an overview of IIoTs to illustrate how the compromise of different parts can affect the system-wide security. Besides, we further review the concept of deep-learning based side-channel analysis and the relevant background knowledge on physical side-channel monitors.

2.1. Industrial internet of things

The IIoT refers to the network of interconnected industrial devices and systems that leverage sensors, actuators, and other technologies to collect, exchange, and analyze data. Unlike traditional IoT, which focuses on consumer applications, IIoT is for industrial settings such as manufacturing, energy, transportation, and healthcare, in which security issues can lead to more serious and costly damages. IIoT enables seamless integration of physical machinery with digital technologies, allowing for real-time monitoring, automation, and optimization of industrial processes. By connecting devices and systems across operational layers, IIoT enhances efficiency, productivity, and decision-making capabilities in industrial environments. Fig. 2 illustrates how IIoT works across different components. At the edge layer, interconnected actuators, control systems, and sensors form the backbone, alongside gateway units responsible for aggregating and forwarding directives to facilitate intercommunication among connected devices.

[1] https://developer.arm.com/ip-products/processors/cortex-m/cortex-m4
[2] https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/8-bit-mcus/avr-mcus

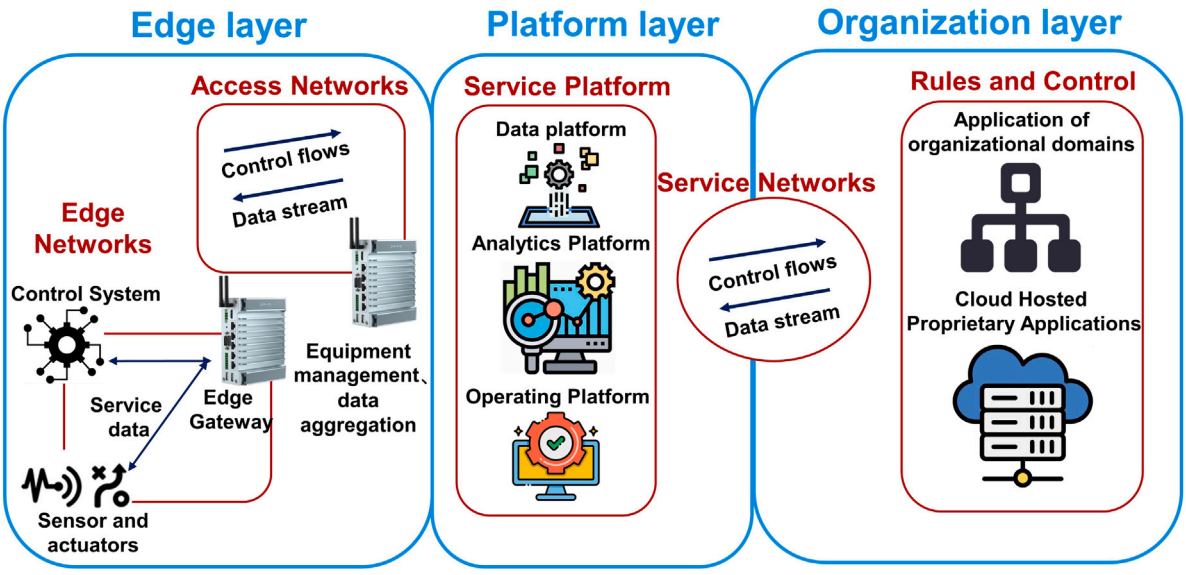


Fig. 2. The illustration of the IIoT framework.

Moving to the platform layer, data, analytics and operating platforms play a critical role in data integration, transformation, and analysis. These platforms leverage web or mobile network connections to establish robust pipelines for seamless data and control exchange across different operational levels. The organizational layer supports domain-specific applications and cloud-hosted proprietary systems, providing user interfaces via intranet infrastructure and web-driven protocols for rule deployment and control management. Wired and wireless interconnections logically link these layers, enabling efficient data collection, analysis, and exchange across IIoT environments. Despite these advancements, initial implementations of industrial IoT systems often prioritized functionality over security, leaving them vulnerable to cyber threats. Even attacks targeting upper-layer platforms can compromise lower-layer physical systems, resulting in anomalous behaviors and operational disruptions. For example, it might seem unrelated at first, but a Windows platform worm can impact IIoT systems as well. Many IIoT systems are managed or monitored through Windows-based control systems. If a worm infects these Windows systems, it could disrupt the control and monitoring processes, potentially leading to operational failures or security breaches in IIoT environments (Mekala et al., 2023).

2.2. Deep-learning side-channel analysis

In 1996, Paul C. Kocher first introduced SCA (Kocher, 1996), demonstrating that the non-constant run time information generated from cryptographic implementations can be used to derive the secret key of cryptographic algorithms. Afterwards, he pioneered power analysis in Kocher et al. (1999), in which shows that the power consumption generated from CMOS components' opening and closing caused fluctuations and memory based interactions in digital chip can be used as the side channel. In 2010, Yuval Yarom et al. introduced the flush + reload attack (Yarom and Falkner, 2014), a cache-based side-channel analysis. Subsequent developments include the discovery of the KRACK vulnerability in Wi-Fi networks using WPA2 in 2017 (Vanhoef and Piessens, 2017), enabling data theft, and the revelation of the Load Value Injection (LVI) attack in 2020 (Van Bulck et al., 2020), exposing flaws in modern CPU designs. Deep neural networks have been widely employed in addressing SCA problems (Staib and Moradi, 2023; Cao et al., 2022), offering superior performance compared to traditional techniques like Differential Power Analysis (DPA) or CPA. In general, a well-trained deep-learning model can make the attack several orders of magnitude more efficient than traditional signal processing approaches (Wang et al., 2021). Excluding certain exceptions (Timon, 2019), deep-learning techniques are predominantly employed in profiled settings, which typically involve two stages: profiling and analysis. During the profiling stage, operators are often presumed to have full control over one or more profiling devices, identical to the target device. This allows the operator to gather numerous side-channel traces and associated information. Next, a chosen deep-learning model is trained to establish a leakage profile linking the activities running on the chip to the traces. During the analysis stage, it is common in the community to assume that operators could obtain side-channel traces during the execution of the target chip and use the pre-trained model to deduce the procedures (Wang, 2024). This process underscores the importance of how a side-channel monitor is integrated into a chip or device, which can be done either during the design and manufacturing stage or added externally at a later stage. Integrating a side-channel monitor directly into a chip during the design and manufacturing stage offers a better security level but increases complexity and cost, aligning with the high-control environment of the profiling stage. Conversely, adding a monitor at a later stage provides the flexibility necessary for adapting to evolving threats and technologies, though it may face integration challenges and potentially offer less effectiveness in security, similar to the conditions encountered during the analysis stage.

2.3. Physical side-channel monitors

SCAs can also be used for the bright side (Feng et al., 2023). Physical side-channel monitors (Han et al., 2019; Ugurlu et al., 2021; Maillard et al., 2023; Das et al., 2020; Zhao et al., 2024) are embedded tools used to detect and analyze side-channel leakage during the operation of IIoT edge devices for control flow integrity monitoring (Awal and Rahman, 2023). Physical side-channel monitors offer a significant advantage over software-based monitors, especially in air-gapped environments, where network-based software monitors may not be feasible or secure due to the lack of connectivity. Unlike software monitors that rely on system resources and can potentially be compromised by the very software they are meant to monitor, physical side-channel monitors operate independently, making them inherently more secure. These monitors analyze hardware-level signals, such as power consumption or electromagnetic emissions, which are difficult for malicious software to alter or obscure. Moreover, the use of advanced signal processing

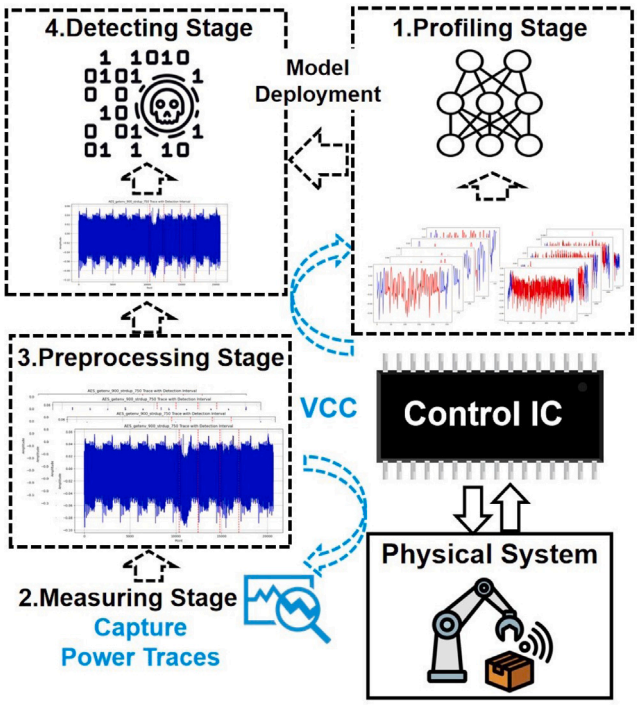


Fig. 3. Illustration of how a physical side-channel monitor works. The monitor first measures power consumption from a control IC in a physical system during the execution of a specific series of instructions. By further analyzing the captured traces, it is feasible for monitor to check whether the system is working properly. The overall deployment and detection process includes four stages: profiling, measuring, preprocessing and detecting. The system aims to provide real-time warnings against potential malicious code intrusion threats.

techniques, including machine learning and noise filtering, enhances the accuracy and efficiency of physical side-channel monitoring, providing a robust defense against code intrusions in critical systems. By capturing power consumption traces and analyzing them during the execution of the code, it is feasible to monitor whether the device is running instructions properly. In 2016, Liu et al. (2016) explores how power traces can be used to detect malicious software intrusions in IoT systems. Subsequently, Han et al. proposed ZEUS (Han et al., 2017), a framework which utilizes EM emissions generated from chips to monitor the malicious code intrusions at the block level in industrial control systems. Alireza et al. then introduced EDDIE (Nazari et al., 2017), utilizing physical side-channel detection for monitoring program execution deviations.

Fig. 3 is an illustration of a general physical deep-learning based side-channel monitor designed to detect malicious activities by analyzing power consumption traces captured from integrated circuits within a physical system. Typically, A physical side-channel monitor is a tool or system used to detect and analyze unexpected side-channel measurement during the execution of the chip to reveal information about internal processes and providing timely alerts to potential security threats. This monitoring system functions through four distinct stages.

1. Profiling stage. Initially, deep-learning models are trained to identify correlations between patterns in power consumption traces and corresponding activities on the chip. Subsequently, these trained models are implemented in the side-channel monitoring system for ongoing analysis.
2. Measuring stage. During the execution of the target program, the side-channel monitor continuously captures power traces generated from the chip.
3. Preprocessing stage. The monitor subsequently applies essential preprocessing techniques, including filtering and scaling, to refine the collected data.
4. Detecting stage. The deployed model then conducts real-time classification on newly processed power consumption trace segments to determine the presence of any potential security threats.

Through this sequence of processes, the system effectively monitors the security status of the physical system, preempting potential attacks.

3. Detection target

In modern industry, *C* programming language is renowned for its low-level features, high performance, and direct hardware access capabilities, making it an ideal choice for embedded systems programming. When developing complex embedded systems, *C* language library functions assist developers by simplifying processes, handling operations like string manipulation, file handling, and network communication. However, these conveniences also provide opportunities for adversaries. Programming without the utilization of library functions is nearly impractical, and the same holds true for malicious code. However, existing works in detecting malware using physical side-channel monitors are limited to identify known malicious code execution at the block level. A code block is a segment used to organize and control the flow of execution and variable scope within a program. A code block can contain multiple statements and is typically employed in conditional statements, loop structures, or within functions to aid developers in organizing and managing program logic. In contrast, a single function operation is a segment of code encapsulating specific functionality with its own scope and return value, designed for reuse and modularity throughout the program. Given the infinite nature of malicious code blocks and the finite nature of library functions, we focus on detecting unexpected *C* language library function operations in planned control flows. We narrow down our detection by using the novel CNDSW side-channel monitor to the level of *C* library functions, enabling the identification of potentially malicious software. Notice that physical side-channel monitors differ from other malicious code detection approaches primarily because they rely on indirect evidence of malicious activity rather than analyzing the code itself. This allows them to detect even well-obscured threats without needing direct access to the targeted software or hardware's operational code, making them particularly effective against sophisticated attacks that might not alter the code in detectable ways.

Building on this conceptual framework, we first conduct a detailed power leakage assessment on each *C* language library function individually on embedded devices based on ARM Cortex-M4 and RISC architectures. For every library function, we repetitively execute the same operation and monitor the corresponding power consumption waveform. For the ARM device, we successfully identify 29 library functions with significant power leakage during the runtime, while on the RISC device, we identify 21 library functions with noticeable power leakage. Afterwards, we collect these power leakage data and form the original dataset, with 500 power traces collected for each category of library functions. As shown in Table 1 and Table 2, we present all the leaked library functions we have detected. In the following experiments, we use the ChipWhisperer-Lite CW1173 for measuring power consumption traces during the execution of two devices with different core architectures. ChipWhisperer-Lite is an open-source toolchain designed for hardware security research, particularly focusing on side-channel power analysis. It consists of both hardware and software components, with the hardware being a small, USB-powered device equipped with a microcontroller for target operation and an FPGA for power measuring. The experimental setup details are further presented in the following Section 6.1. Before the training process and the deployment of the proposed CNDSW model, we conduct a preliminary

Table 1
Library functions exhibiting power leakage on the ARM board.

Header file	Functions with obvious leakage
`stdlib.h`	system(), bsearch(), atoi(), atol(), div(), getenv(), itoa(), ldiv(), realloc(), qsort(), rand(), srand(), strtol(), swab()
`string.h`	strcpy(), strcat(), strdup(), strlwr(), strncat(), strncpy(), strtok(), strupr()
`mem.h`	memccpy(), memmove()
`ctype.h`	tolower(), toupper()
`float.h`	clear87()
`setjmp.h`	setjmp(), longjmp()
`assert.h`	No Obvious Power Leakage Library Functions Detected So Far.
`io.h`	No Obvious Power Leakage Library Functions Detected So Far.
`math.h`	No Obvious Power Leakage Library Functions Detected So Far.
`stdio.h`	No Obvious Power Leakage Library Functions Detected So Far.
`signal.h`	No Obvious Power Leakage Library Functions Detected So Far.
`time.h`	No Obvious Power Leakage Library Functions Detected So Far.

Table 2
Library functions exhibiting power leakage on the RISC board.

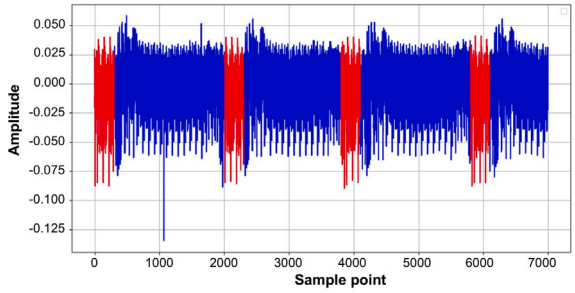
Header file	Functions with obvious leakage
`stdlib.h`	bsearch(), atoi(), getenv(), itoa(), realloc(), srand(), strtol()
`string.h`	strcpy(), strcat(), strdup(), strlwr(), strncat() , strncpy(), strtok(), strupr()
`mem.h`	memccpy()
`ctype.h`	tolower(), toupper()
`float.h`	clear87()
`setjmp.h`	setjmp(), longjump()
`assert.h`	No Obvious Power Leakage Library Functions Detected So Far.
`io.h`	No Obvious Power Leakage Library Functions Detected So Far.
`math.h`	No Obvious Power Leakage Library Functions Detected So Far.
`stdio.h`	No Obvious Power Leakage Library Functions Detected So Far.
`signal.h`	No Obvious Power Leakage Library Functions Detected So Far.
`time.h`	No Obvious Power Leakage Library Functions Detected So Far.

Algorithm 1 A malicious code with `system` library function

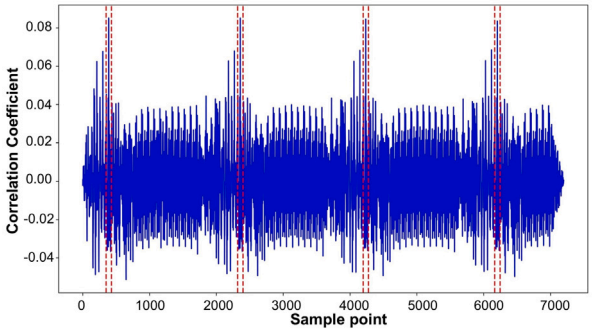
```
malicious_cmd ← "rm important_file.txt"
while true do
    ret ← system(malicious_cmd)
end while
```

Algorithm 2 A malicious code with three library functions

```
function maliciousOperations(input):
    modifiedInput = strdup(input)
    strcat(modifiedInput, "_malicious")
    malicious_cmd = "rm "
    command = concatenate(malicious_cmd, modifiedInput)
    system(command)
    free(modifiedInput)
function main():
    while true:
    input = "file_to_be_modified.txt"
    maliciousOperations(input)
```



(a) The plot of a power trace captured from an ARM Cortex-M4 implementation of Algorithm 1 during the execution period. The red segments represent malicious operations invoking the `system` library function.



(b) The correlation analysis results between the captured traces (Algorithm 1) and the trace template in the previously database. The dashed red lines show the beginning and end of correlation 'peaks'.

Fig. 4. An example trace representing the execution of Algorithm 1 and the corresponding correlation analysis results. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

experiment to capture power consumption traces during the execution of each library function from two target devices. By monitoring the power consumption of the STM32F3 and STM32F4 chips, both based on the ARM Cortex-M4 architecture, we confirm that the traces were highly similar across devices with the same core architecture, as expected. Besides, we find that for each type of the function, collecting more than 20 traces is sufficient to establish a baseline. This is further discussed in the following small-sample model tests in Section 7.1.

We elaborate on the relationship between trace segments representing *C* library function and traces captured from implementations of control flows through the following two specific examples.

In the first example, we design Algorithm 1 which invokes the *C* library function *system* to acquire privileges for deleting a crucial file (pseudo code in Algorithm 1). We capture power traces from target devices when they are repeating the same algorithm. Afterwards, we conduct the correlation analysis between the captured traces and the trace segment of the previously captured database involving library functions. Remarkably, even when the specific operation performed by the *system* function calls differ, we consistently find the approximate position of the malicious *system* function in the captured trace according to the correlation analysis result. The top picture in Fig. 4 shows an example trace captured from an ARM Cortex-M4 implementation of Algorithm 1 with segments representing malicious activities in red. The trace represents four consecutive executions of Algorithm 1, with sample points 0–2000 in the figure corresponding to a single execution of the algorithm. The bottom plot in Fig. 4 illustrates the correlation analysis results between the captured traces and the trace segment of the previously captured database involving library functions. From the bottom plot in Fig. 4, we can clearly identify the position of malicious activities from the correlation 'peaks' and we use the dashed red lines for further visualization.

In the second example, we first define a malicious execution algorithm that encompasses multiple library function operations: *system*,

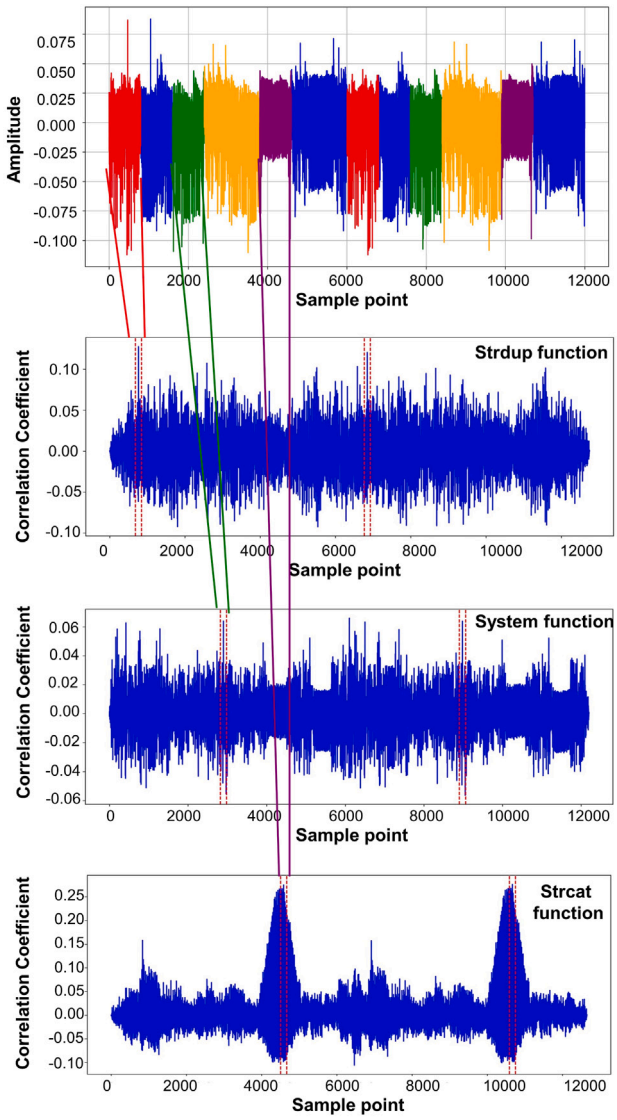


Fig. 5. An example trace representing the execution of Algorithm 2 and the corresponding correlation analysis results. The first figure shows a power trace captured from an ARM Cortex-M4 during the execution of Algorithm 2. The red segments represent calls to the *strdup* library function, the green segments represent calls to the *system* function, and the purple segments represent calls to the *strcat* library function. The last three figures present the correlation analysis results between the captured traces (Algorithm 2) and the trace templates in the previously established database. The dashed red lines show the beginning and end of correlation 'peaks'. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

strdup, and *strcat* (see the pseudo code in Algorithm 2). The top picture in Fig. 5 shows the captured trace of an ARM Cortex-M4 implementation of Algorithm 2. The red segments in the power trace represent calls to the *strdup* library function, the green segments represent calls to the *system* function, and the purple segments represent calls to the *strcat* library function. By repeating this algorithm and monitoring the generated trace, we similarly observe three distinct library function calls from the correlation analysis results as shown in Fig. 5. The bottom plot in Fig. 5 illustrates the correlation analysis results between the captured traces and the trace segment of the previously captured database involving library functions. From the bottom plot in Fig. 5, we can clearly identify the position of different malicious activities from the correlation 'peaks' and we use the dashed red lines for further visualization.

4. Threat model

The importance of embedded devices in industrial control systems is self-evident, but the security challenges are equally significant (Khalil et al., 2023; Bhamare et al., 2020). One major challenge is the inadequate security protection of embedded devices during remote programming. Remote programming refers to the process of updating or configuring embedded devices, such as PLCs, over a network from a remote location. Remote programming between a host computer and edge embedded devices refers to uploading program code or firmware updates via a network or other remote connection methods. The primary advantage of this approach is that developers can debug, update, and maintain embedded devices without physical contact, thereby improving efficiency and flexibility. However, remote programming also faces numerous potential security threats, including unauthorized access, man-in-the-middle attacks, firmware tampering, and data leakage. For example, attackers might exploit vulnerabilities to gain remote access to the embedded device, upload malicious code, or intercept and alter data during communication between the host computer and the embedded device. Without appropriate verification mechanisms, attackers can also replace legitimate firmware, causing the device to execute malicious operations. This capability allows for real-time management and updates but also introduces significant security vulnerabilities, thereby threatening the overall security of the industrial control system. Some devices do have hardware write protect jumpers to prevent unauthorized writing or modification of the embedded device's memory (such as flash memory). By configuring the jumper, developers can enable or disable write access to the memory, thereby enhancing device security level. However, these features are often disabled in practice. This lack of hardware protection leaves many embedded devices exposed to potential threats, necessitating more advanced security measures in many security critical fields. In this context, the proposed CNDSW framework aims to provide the capability to timely detect and warn against unexpected updates.

We assume there is a trusted path from CNDSW to system operators to alert them to when malicious executions happen. The non-intrusive monitoring of CNDSW allows for secure monitoring in situations where the software stack below the control logic of embedded devices (such as firmware or operating system) is threatened. The CNDSW framework is designed to use out-of-band communication via dedicated link to ensure secure interaction with system operators. This method avoids the security risks associated with using the same network paths as the control logic execution. The network links bound to embedded devices are assumed to be trusted, allowing CNDSW to obtain legitimate copies of control logic and compare them with the runtime execution of embedded devices for control flow integrity checks. CNDSW does not assume access to source code and can be used with binary files. CNDSW is refined to the function level to detect control channel attacks (e.g., Stuxnet (Falliere et al., 2010)). In these attacks, adversaries upload arbitrary and potentially malicious control logic to embedded devices for execution. Specifically, types of control logic attacks that CNDSW can detect include:

1. Modified control logic, such as injection, deletion, and replacement of code segments in legitimate control logic programs, using library functions during injection.
2. Control flow hijacking of legitimate control logic execution through network attacks (e.g., code reuse attacks such as return-oriented programming[3]).

[3] https://www.checkpoint.com/harmony-2/

5. CNDSW detection framework

In this section, we introduce the working principles of the CNDSW detection framework in industrial control flow monitoring. The framework comprises two main stages: the training stage and the detection stage.

During the training stage, we employ five different deep-learning algorithms for the classification task of raw power consumption traces associated with library function leaks. Simultaneously, we compare the differences in classification task accuracy after applying fast Fourier transform to the traces. We refer to the traces captured and stored directly by the ChipWhisperer-Lite board from the target device as raw traces. During the capture process, the measured signal undergoes minimal processing, being handled only by the low-noise amplifier and ADC.

During the detection stage, we combine correlation analysis with the pre-trained deep-learning models to analyze the captured power traces for detecting the potential threats.

5.1. Preprocessing

We perform data preprocessing on the collected raw power traces $\{s_1, s_2, \dots\}$ which include inserting short traces and truncation operations. Due to varying lengths of the original power traces, the number of sample points in different library function traces ranges from 100 to 1100. Subsequently, we gathered power curves $l_1, l_2, \dots$ during normal program execution flow.

A random position is selected within the long sequence, and the short trace S_i is inserted at that position with the short trace as the center. Assuming the length of the long sequence is n_l, the insertion position i ranges from $1 \le i \le n_l$. Specifically,

$$l_i' = l_i[1 : i-1] \cup S_i \cup l_i[i : n_l], \tag{1}$$

where l_i' represents the long sequence after insertion, $l_i[1 : i-1]$ denotes the first $i-1$ points of the long sequence l_i, and $l_i[i : n_l]$ denotes the points from the ith to the n_lth point of the long sequence l_i.

Subsequently, we truncate the preprocessed traces to ensure uniform dimensions. Let the length of the truncated trace be $n_{\text{truncated}}$. If the original number of points is less than or equal to 256, we truncate it to 256 points: $n_{\text{truncated}} = 256$. If the original number of points is between 256 and 1024, we truncate it to 1024 points: $n_{\text{truncated}} = 1024$. The specific operation is

$$l_i'' = \begin{cases} l_i'[1 : 256], & \text{if } |S_i| < 256 \\ l_i'[1 : 1024], & \text{if } 256 < |S_i| \le 1024 \end{cases} \tag{2}$$

where l_i'' is the truncated trace sequence.

Simultaneously, we perform fast Fourier transform (FFT) on the truncated traces, establishing a new FFT dataset. The process of fast Fourier transform is as follows:

$$X(f) = \int_{-\infty}^{\infty} x(t) \cdot e^{-i2\pi ft}\, dt, \tag{3}$$

where $X(f)$ is the continuous signal in the frequency domain, $x(t)$ is the continuous signal in the time domain, f is the frequency, and i is the imaginary unit. Finally, the preprocessed trace sequence l_i'' obtained after truncation undergoes fast Fourier transform (FFT) to convert it into frequency domain representation.

5.2. Training stage

During the training stage, we utilize five different deep-learning algorithms to learn how to classify the short sequence traces in the S_{256} and S_{1024} collections into corresponding library functions. Here, S_{256} and S_{1024} denote collections of preprocessed short traces with 256 and 1024 points, respectively. We also find that for each type of function, collecting more than 20 traces is sufficient to establish a baseline. We use the one-hot encoding method to label every trace segment with the corresponding function name. In the following experiments, we test the following five types of model to explore the optimum.

1. Multilayer Perceptron

The Multilayer Perceptron (MLP) model is employed to address the short trace classification problem. The model consists of three layers: input layer, two hidden layers, and output layer. The input layer dimension is 256, connected to two hidden layers containing 64 and 32 neurons, respectively. ReLU activation functions are used to introduce non-linearity. The output layer comprises 18 neurons, utilizing softmax activation for classification. During experiments, we employ the Adam optimizer for model training, cross-entropy loss function, and monitor model accuracy.

2. Convolutional Neural Network

The 1D Convolutional Neural Network (1D CNN) model is also employed for short trace classification. Initially, the model reshapes input data into $256 \times 1256 \times 1$ and then performs feature extraction using 1D convolutional layers. Convolutional layers utilize $3 \times 13 \times 1$ kernels with ReLU activation to introduce non-linearity. Subsequently, max-pooling layers are used for downsampling to reduce feature dimensions. Further, another set of 1D convolutional layers is used for extracting higher-level features, followed by max-pooling layers for dimensionality reduction. Finally, a flatten operation is applied to map features to a one-dimensional vector for processing by fully connected layers. The output layer maps features to the final output space and uses softmax activation for classification. During model training, the Adam optimizer is employed for parameter updates, cross-entropy loss function, and model accuracy monitoring.

3. Long Short-Term Memory

The Long Short-Term Memory (LSTM) neural network model is employed for short trace classification as well. It utilizes LSTM layers to model input sequences, where each input sample is with specific time steps and feature numbers. The LSTM layer consists of 64 memory units to capture long-term dependencies and temporal dynamics in input sequences. Subsequently, a fully connected layer maps LSTM layer outputs to a higher-dimensional representation space for better capturing non-linear relationships in the data. Finally, another fully connected layer maps high-dimensional representations to the final output space and uses softmax activation for classification. During model training, the Adam optimizer is used for parameter updates, cross-entropy loss function, and model accuracy monitoring.

for $t = 1$ to T :

$$\begin{aligned} i_t &= \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \\ f_t &= \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \\ o_t &= \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \\ g_t &= \tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g) \\ c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\ h_t &= o_t \odot \tanh(c_t) \end{aligned} \tag{4}$$

end for

Here, σ represents the sigmoid function, $\odot$ denotes element-wise multiplication, and W and b are the LSTM layer weights and biases.

4. Separable Convolutional Neural Network (Sifre and Mallat, 2014)

The Separable Convolutional Neural Network (SeCNN) model is employed for short trace classification. SeCNN is a variant of convolutional neural networks that reduces parameter count and computational load by applying depthwise and pointwise convolutions separately in spatial and channel dimensions. This structure maintains model performance while achieving higher efficiency and faster training speed. In this study, the SeCNN model first reshapes input data into $256 \times 1256 \times 1$ and then performs feature extraction using separable convolutional layers. After each separable convolutional layer, downsampling is performed using max-pooling layers to reduce feature dimensions. Subsequently, another set of separable convolutional layers is employed

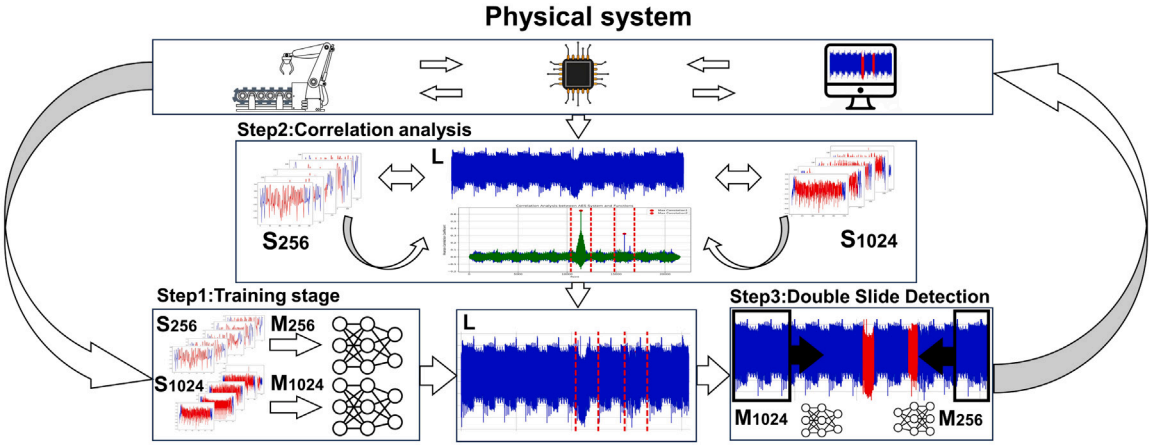


Fig. 6. The proposed CNDSW detection framework.

for extracting higher-level features, followed by max-pooling layers for dimensionality reduction. Through this structure, SeCNN effectively reduces parameter count while maintaining model performance.

5. Four-Channel Regularized CNN We design a four-channel regularized convolutional neural network (FLCNN) suitable for training with small samples. Preprocessing and reshaping operations are applied to the input data to meet the requirements of the neural network model. Initially, we introduce a channel dimension and replicate it, resulting in input data with four channels. The model architecture comprises a convolutional layer with 64 filters, followed by ReLU activation and L2 regularization (Bühlmann and Van De Geer, 2011) to prevent overfitting. This is succeeded by a max-pooling layer for downsampling. Subsequently, another convolutional layer with 128 filters, ReLU activation, and L2 regularization is utilized. Following this, another max-pooling layer is added, succeeded by a convolutional layer with 256 filters, ReLU activation, and L2 regularization. Finally, a max-pooling layer is incorporated for dimensionality reduction. After the convolutional layers, a flatten layer and a dropout layer are introduced to enhance model generalization. Lastly, a fully connected layer is employed to map features to the final output space, utilizing softmax activation for classification. Throughout model training, the Adam optimizer is utilized for parameter updates, cross-entropy loss function, and monitoring model accuracy.

$$X_{\text{reshaped}} = \text{reshape}(X, (T, N, 4)) \tag{5}$$

$$z_i^{(t)} = \text{ReLU}(W_i * x_t + b_i) + \lambda \cdot \text{Regularization}(W_i) \tag{6}$$

5.3. Detection stage

In the detection stage, our focus shifts to the real-time trace captured from the running MCUs, encompassing one or more occurrences of library function operations. The overall detection process can be further divided to three steps as shown in Fig. 6: (1) capture traces from the running target device, (2) locate library functions in traces by using the correlation analysis, (3) detect unexpected malicious function with the pre-trained models.

It is crucial to establish a normal execution sequence of the control flow functions before detection, which serves as a baseline for identifying deviations. While block-level intrusion detection analyzes control flow transitions between basic blocks (Kuang et al., 2020), it is limited in scope as it does not capture inter-function relationships or anomalies that span across multiple functions. Our function-level monitoring addresses this gap by providing a holistic view of system behavior, allowing us to detect more complex, multi-function deviations that could indicate sophisticated attacks. By comparing real-time traces against the pre-established normal sequence, our system can identify deviations not only within functions but across their interactions, thus enhancing detection capabilities for both intra- and inter-function anomalies.

The first step is to record the power trace during the runtime of the target device. We denote the newly captured long trace as L.

Afterwards, we sequentially use trace segments in S_{256} and S_{1024} as template to conduct correlation analysis with the newly captured long trace for approximately locating different library functions, as shown in formula (7).

$$\text{Corr}(L, S) = \frac{\sum_{i=1}^{n}(L[i] - \bar{L}) \cdot (S[i] - \bar{S})}{\sqrt{\sum_{i=1}^{n}(L[i] - \bar{L})^2 \cdot \sum_{i=1}^{n}(S[i] - \bar{S})^2}}, \tag{7}$$

where S denotes a specific trace segment in S_{256} or S_{1024}. By checking the location and order of library functions, it is feasible to roughly know the order of different functions. However, as shown in Fig. 5, correlation analysis results may not perfectly reflect the locations. There might be some non-function trace intervals in L are highly correlated with the prepared template of the specific function. Thus, it is necessary to use deep-learning model to further check these intervals with high correlation coefficient (detection intervals).

Next, we use the pre-trained M_{256} and M_{1024} to slide on detection intervals of L for classifying segments with intervals. For a segment with offset i in L and sliding step size k, the detected function $\tilde{f}_i$ is reported as the one with highest probability classified by M_{256}:

$$f_{i+k} = \arg\max(M_{256}(L[i+k : i+k+256])) \tag{8}$$

So the detected function $\tilde{g}_i$ for M_{1024} is denoted as follows:

$$g_{i+k} = \arg\max(M_{1024}(L[i+k : i+k+1024])) \tag{9}$$

By adjusting the sliding step size k, it is possible to set the detection time T for a specific detection interval as the following formula.

$$\begin{aligned} T_{256} &= \left\lfloor \frac{Y - 256}{K} \right\rfloor \cdot X_{256}, \\ T_{1024} &= \left\lfloor \frac{Y - 1024}{K} \right\rfloor \cdot X_{1024}, \\ T &= T_{256} + T_{1024} + \text{constant}, \end{aligned} \tag{10}$$

where X denotes the detection time for calling deep-learning model once and Y is the length of the detection interval. Obviously, we can find that a lager sliding window means a shorter detection time. Once the detected function f_{i+k} or g_{i+k} matches the real embedded malicious function within the detection interval, the malicious library function is successfully detected.

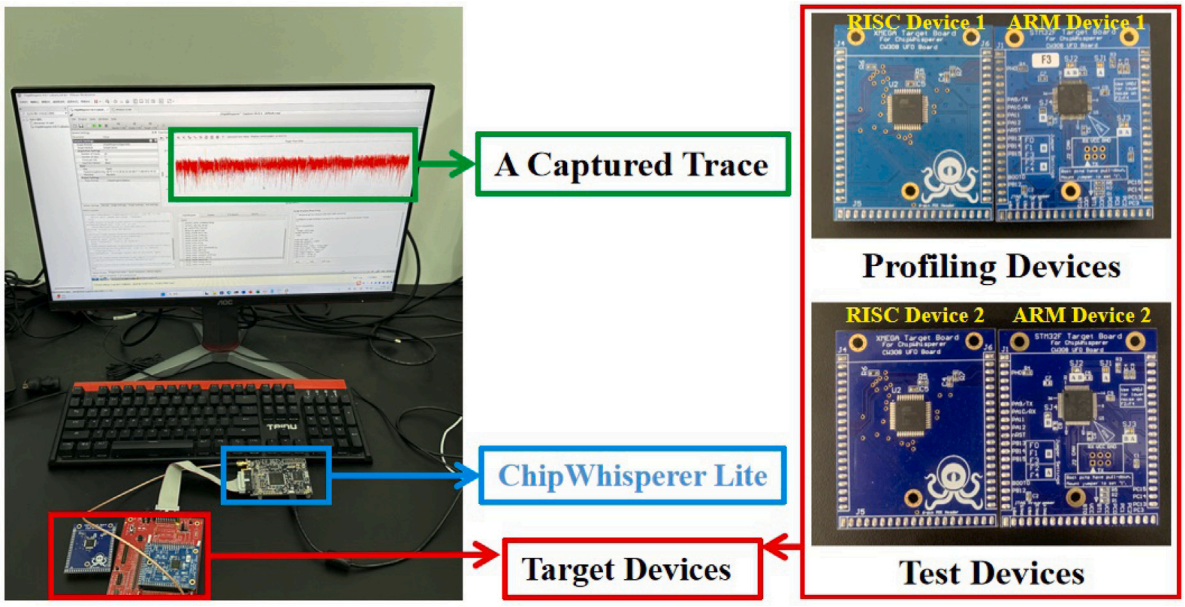


Fig. 7. Experimental setup.

6. Experimental setup

In this section, we first provide detailed descriptions of our experimental setup, including how we capture the power consumption traces from different target boards at the data collection phase and the type of the target embedded devices used. Afterwards, we describe the control flow implemented on target devices.

6.1. Hardware setup

We use the ChipWhisperer Lite board to capture traces, which is a toolkit designed for hardware security evaluation (O'flynn and Chen, 2014). The ChipWhisperer-Lite board is designed to be used in conjunction with the ChipWhisperer software suite, which provides a user-friendly interface for capturing power traces. Meanwhile, we use two different target devices to test the transparency of the proposed CNDSW framework: the CW308T-STM32F3 board, which features an ARM Cortex-M4 microcontroller, and the ATXmega128D4 board, which is built around a RISC architecture microcontroller. For simplicity, we refer to these as the ARM device and the RISC device in the sequel, respectively. The ARM device is operated at its maximum clock frequency of 73.7 MHz, while the RISC device is clocked at its peak frequency of 16 MHz. These settings ensure that both devices are functioning under optimal and realistic conditions for our experiments.

The process of collecting power traces involved a meticulous hardware setup. We inserted a small resistor in series with the V_{cc} pin of each microcontroller. This resistor served as a shunt, allowing us to monitor the current drawn by the device indirectly. By measuring the voltage drop across the resistor in real-time using an oscilloscope connected to the ChipWhisperer-Lite, we could accurately capture the power consumption profile of the devices. This real-time measurement is crucial for constructing detailed power traces that reflect the instantaneous power usage during the execution of various operations.

The captured power traces are further used for subsequent side-channel analysis, providing insights into the power consumption patterns and potential vulnerabilities of the microcontrollers. This setup and methodology ensured high fidelity in the power measurements, forming a solid foundation for our security analysis and helping us to identify possible side-channel attack vectors in these embedded systems. The hardware setup is as shown in Fig. 7. The left part of the figure shows how the ChipWhisperer-Lite board is connected to the target board. The right part of the figure are the four chips used in the following experiments. The top plot shows an ATXmega128D4 boards and a CW308T-STM32F3 board used for profiling, and the bottom plot shows two testing devices. In the sequel, we refer to these target boards as RISC device and ARM device, respectively, as shown in the right part of Fig. 7.

6.2. Evaluation programs

Our assessment focuses on five control flows or cryptographic algorithms: ***AES-128*** (Rijmen and Daemen, 2001), ***PID*** (Johnson and Moradi, 2005), ***PSD*** (Youngworth et al., 2005), ***Sensor20*** (Huang et al., 1979), and ***SensorSorting*** (Astrachan, 2003). These algorithms and control flows are selected for their relevance in real-world IIoT applications, spanning cryptographic operations, control system behaviors, and sensor data processing.

To compile the programs for our experiments, we utilized the GNU compiler with various optimization flags. While compiling and testing individual library functions, we observed that the trace segment of library function leaks showed no significant differences across different optimization levels. However, when these library functions were executed within a control flow program, the overall waveform exhibited variations due to optimization levels. In the following experiment, we fix the optimization level to the default mode for each chip.

Each of these programs serves a specific purpose in practical applications:

- ***AES-128***: This widely-used encryption standard is fundamental for securing communication in IIoT. By analyzing its power traces, we can gain insights into the cryptographic operations and potential side-channel vulnerabilities.
- ***PID***: Proportional–Integral–Derivative controllers are integral to industrial control systems. Evaluating PID algorithms helps us understand control system behaviors and detect anomalies that could indicate security breaches.
- ***PSD***: Power Spectral Density analysis is crucial for signal processing in various sensor applications. Studying PSD implementations allows us to assess vulnerabilities in the signal processing routines of IIoT edge devices.

Table 3
Composition of the training set.

Device	# Sample points for each trace	# Functions	# Traces per function	# Traces in total
ARM device	256	17	500 traces	8,500 traces
	1024	12	500 traces	6,000 traces
RISC device	256	13	500 traces	6,500 traces
	1024	10	500 traces	5,000 traces

- ***Sensor20***: This program represents fast data collection from sensors, which is essential for real-time monitoring and response in IIoT systems. By examining Sensor20, we can identify potential weaknesses in the sensor data processing pipeline.
- ***SensorSorting***: Sorting algorithms are often used in sensor data processing to organize and prioritize information. Analyzing SensorSorting helps us evaluate the robustness of data handling in IoT devices.

In our experiments, we inject malicious code into each of these five programs and implemented them on our two target devices: the ARM device and the RISC device. This setup allowed us to simulate real-world attack scenarios and assess the effectiveness of our proposed CNDSW framework.

After deploying the modified programs, we meticulously evaluate the CNDSW framework's capability to identify anomalies arising from the injected code across various contexts. This evaluation provided valuable insights into the framework's robustness in detecting security threats in embedded systems.

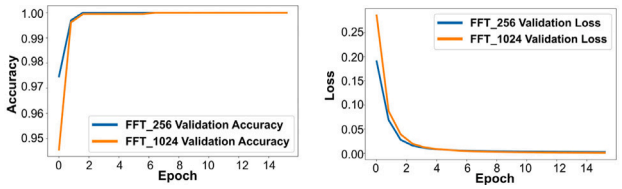
7. Experimental results

This section presents the results of our experiment, which is divided into two parts: detection on captured traces and detection on real-time signals. In the preliminary stage, we successfully collect traces for library function operations on ARM Cortex-M4 and RISC architecture chips. It is noteworthy that ARM chips exhibited leakage in 29 out of library functions, while RISC chips showed leakage in 21 functions.

7.1. Training stage

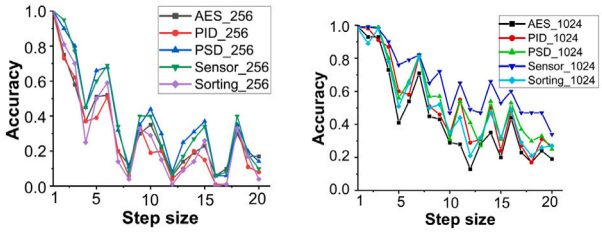
During the model training phase, we first insert library function trace segments into five evaluation programs. We extract 256 and 1024 data points containing library function operations, mark function names with one-hot encoded vectors, and establish corresponding training sets. Subsequently, we train two types of models for each evaluation algorithm on the constructed training sets. In Table 3, we list the composition of our training phase dataset. The number of functions, the number of traces for each function, and the total number of traces for both *ARM device1* and *RISC device1* are listed for 256 and 1024 data points. For each device, 500 traces are collected for different library functions at different numbers of data points.[4]

For the traces collected on the ARM board, as shown in Tables 4–5, all models demonstrate excellent performance with sufficient training data. The MLP model achieves 100% validation accuracy within ten epochs on the dataset with 256 data points and within 20 epochs on the dataset with 1024 data points, yielding outstanding test results with an average accuracy of 99.96%. The CNN model achieves an average validation accuracy of 99.82% within 20 epochs on both 256 and 1024 data point datasets, performing even better in testing with an average accuracy of 99.99%. The LSTM model achieves 100% validation accuracy within 20 epochs on both 256 and 1024 data point datasets, with a test accuracy of 99.98%. The SeCNN model, balancing lightweight and performance, converges within fifteen epochs with an



(a) Training and validation accuracy on FFT traces. (b) Training and validation loss on FFT traces.

Fig. 8. Training and validation performance of the MLP model on traces after the spectrum transformation (FFT traces).



(a) Detection results of M_{256}. (b) Detection results of M_{1024}.

Fig. 9. Detection results of *MLP* models on ARM traces.

average validation accuracy of 99.99% and an average test accuracy of 99.99%. The FLCN model, trained and tested with only 4% of the original dataset, achieves satisfactory results in small sample learning, with an average validation accuracy of 98.78% and an average test accuracy of 98.76%.

For the traces collected from the RISC board, as observed in Tables 6 and 7, the MLP model, CNN model, SeCNN model, and LSTM model achieve an average validation accuracy of approximately 99% on the 256 data point trace dataset across five algorithms, while the average validation accuracy on the 1024 data point trace dataset reaches around 90%. The overall average test accuracy is not significantly different from the validation accuracy. The FLCN model, trained on small sample learning for traces on the ARM board, demonstrates good performance with a similar dataset size, with average validation and test accuracy close to 100%.

We attribute this discrepancy mainly to the differences in trace categories and data volume. The ARM board's 256 data point trace includes 17 categories, and the 1024 data point trace includes 12 categories, while the RISC board's 256 data point trace comprises only 13 categories, and the 1024 data point trace comprises 10 categories. Each trace category consists of 500 samples, divided into training, validation, and test sets in a 6:2:2 ratio. On the RISC board, the same models need to deal with a smaller amount of data, but we find that the designed small sample learning FLCN model can still achieve remarkable results with increasing data volume.

Simultaneously, we perform fast Fourier transform on the training set data, establishing a spectrum dataset. Figs. 8(a) and 8(b) show the performance of the spectrum dataset during the training of the MLP model.

[4] Traces, code and model structures are publicly available at https://github.com/DarColin.

Table 4
Performance comparison of different models on validation traces captured from the ARM board.

Program	Number of traces	Val acc (MLP)	Val acc (CNN)	Val acc (LSTM)	Val acc (SeCNN)	Val acc (FLCN)
AES_256	500*18	100%	100%	100%	100%	98.61%
AES_1024	500*13	100%	100%	100%	100%	98.61%
PID_256	500*18	100%	99.94%	100%	99.94%	98.61%
PID_1024	500*13	100%	100%	100%	100%	100%
PSD_256	500*18	99.89%	99.94%	100%	100%	100%
PSD_1024	500*13	100%	100%	100%	100%	100%
Sensor20_256	500*18	100%	100%	100%	100%	98.61%
Sensor20_1024	500*13	100%	100%	100%	100%	96.15%
Sensorsorting_256	500*18	99.94%	99.94%	100%	100%	98.61%
Sensorsorting_1024	500*13	99.92%	100%	100%	100%	98.61%

Table 5
Performance comparison of different models on testing traces captured from the ARM board.

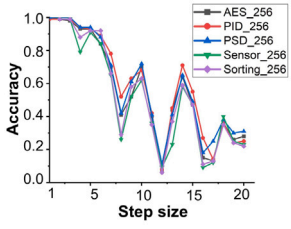
Program	Number of traces	Test acc (MLP)	Test acc (CNN)	Test acc (LSTM)	Test acc (SeCNN)	Test acc (FLCN)
AES_256	500*18	100%	99.94%	100%	100%	98.61%
AES_1024	500*13	100%	100%	100%	100%	100%
PID_256	500*18	99.94%	100%	99.94%	100%	100%
PID_1024	500*13	100%	100%	100%	100%	100%
PSD_256	500*18	99.89%	99.94%	100%	100%	100%
PSD_1024	500*13	100%	100%	100%	100%	100%
Sensor20_256	500*18	99.89%	100%	99.94%	99.94%	100%
Sensor20_1024	500*13	100%	100%	100%	100%	94.23%
Sensorsorting_256	500*18	99.89%	100%	99.89%	99.94%	98.61%
Sensorsorting_1024	500*13	100%	100%	100%	100%	96.15%

Table 6
Performance comparison of different models on validation traces captured from the RISC board.

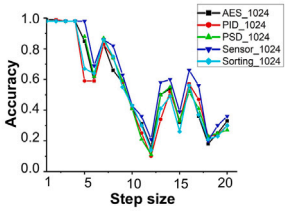
Program	Number of traces	Val acc (MLP)	Val acc (CNN)	Val acc (LSTM)	Val acc (SeCNN)	Val acc (FLCN)
AES_256	500*18	99.46%	99.85%	99.92%	99.62%	99.85%
AES_1024	500*13	90.40%	91.20%	89.50%	90.10%	99.85%
Sensorsorting_256	500*18	99.54%	99.92%	99.92%	99.69%	99.85%
Sensorsorting_1024	500*13	88.50%	90.40%	90.20%	88.10%	99.85%

Table 7
Performance comparison of different models on testing traces captured from the RISC board.

Program	Number of traces	Test acc (MLP)	Test acc (CNN)	Test acc (LSTM)	Test acc (SeCNN)	Test acc (FLCN)
AES_256	500*18	99.69%	99.85%	99.85%	99.85%	99.92%
AES_1024	500*13	90.10%	89.40%	90.50%	89.40%	99.90%
Sensorsorting_256	500*18	99.92%	100%	100%	99.92%	100%
Sensorsorting_1024	500*13	90.00%	89.70%	90.20%	89.50%	100%

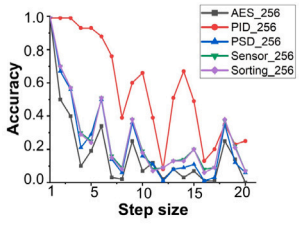


(a) Detection results of C_{256}.

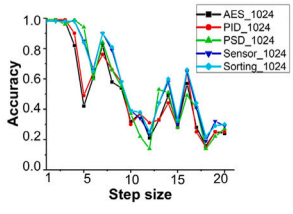


(b) Detection results of C_{1024}.

Fig. 10. Detection results of *CNN* models on ARM traces.



(a) Detection results of SeC_{256}.



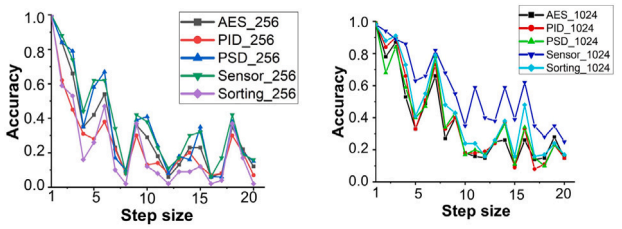
(b) Detection results of SeC_{1024}.

Fig. 11. Detection results of *SeCNN* models on ARM traces.

7.2. Detection on captured traces

We first utilize the CNDSW framework to detect the evidence collected from five evaluation procedures, which include inserted segments representing malicious activities. For each evaluation procedure, we detect the insertion of single and multiple library functions. During the dual sliding detection phase, we conduct detection experiments for each piece of evidence, varying the step size between every two detection windows from 1 to 20. We collect 50 pieces of evidence for each evaluation procedure. On the ARM board, we respectively conduct detection for entire long traces with 29 inserted leaking functions; on the RISC board, we respectively conduct detection for entire long traces with 21 inserted leaking functions.

We first randomly insert segments representing individual malicious activities into each trace of the evaluation program. For ARM devices, we collect traces from five different control flow programs and perform random insertion and detection of library function operations using five algorithms. Each algorithm is tested on a total of 1450 captured traces, each containing approximately 20,000 data points. Subsequently, we use the CNDSW framework to detect each trace. Figs. 9(a) and 9(b) show the detection accuracy of the MLP models M_{256} and M_{1024} on five evaluation programs implemented on ARM devices, with sliding



(a) Detection results of L_{256}. (b) Detection results of L_{1024}.

Fig. 12. Detection results of *LSTM* models on ARM traces.

step sizes set from 1 to 20. We find that both models can perfectly detect injected malicious functions when the sliding step size is set to 1, with no errors in 1450 cases (100% accuracy). However, as the sliding step size increases, the detection accuracy shows a decreasing trend due to the reduced computational workload with increasing step size, resulting in faster detection but at the expense of accuracy. When device resources are not extremely limited, setting the sliding step size to 1 in the CNDSW framework seems to be a good solution for detecting individual malicious functions on ARM devices. We plot the results of how detection accuracy changes of different models from Figs. 9 to 12(b) on five evaluation programs implemented on ARM devices, with sliding step sizes set from 1 to 20. We can find that the MLP model and the CNN model demonstrate a relatively poor robustness capacity compared to other models when the step size is set to a large number (>7). Besides, it can be observed from the figures that the robustness of the FLCN model is significantly stronger than other models, even with less than 20 training samples for each function, demonstrating its potential for industrial applications. This indicates that the FLCN model is feasible to be the optimal neural network deployed in the CNDSW side-channel monitor in the upcoming practical applications.

Afterwards, we attempt to use previously trained models for spectrum-based detection of traces but find that the CNDSW detection framework performs poorly on frequency-domain signals, particularly evident in CPA analysis where correct high correlation positions cannot be obtained. Fig. 14(a) present the detection accuracy results, and Fig. 14(b) illustrate our spectrum analysis process. From Figs. 14(a) and 14(b), we can find that there is no noticeable correlation between the FFT trace and the pre-stored template, which indicates that the FFT traces are not suitable for malicious code intrusion detection at this stage. We present this part to demonstrate the completeness of the experimental process.

For RISC devices, we conduct experiments similar to those on ARM devices. Figs. 15(a) and 15(b) show the detection results. Due to the relatively lower robustness of the models on RISC devices, the best detection results on RISC devices for models with 1024 points (except for the FLCN model) can only reach 90%. For each trace collected by the evaluation program, we use models trained in the previous stage separately and adopt the CNDSW detection framework for detection. On the RISC board, we conduct detection experiments on 1050 traces of length 20,000 points. The detection region surrounds the highest point of the correlation analysis results. For each trace, we conduct detection experiments over a range of step sizes from 1 to 20. In Figs. 15(a) and 15(b), it can be observed that the average detection accuracy at step size 1 is above 90%. From step size 1 to step size 20, the overall detection accuracy of the model shows a decreasing trend, because there are fewer original library function features in overlapping windows. However, we can observe that compared to other algorithms, the FLCN algorithm has stronger robustness. In both figures, the FLCN algorithm exhibits the best performance. The amount of experiments conducted on RISC devices is the same as that on ARM boards. In all ten detection scenarios, the FLCN model has the best performance, and this model has not been re-tuned. The difference between this model and the one trained on ARM boards lies in the fact that it is not trained on a small sample; each sample consists of 500 traces.

In Figs. 16(a)–16(d), we illustrate the detection time different models on traces captured from both ARM and RISC implementation of AES-128 algorithm for the detection step size ranges from 1–20. Overall, the detection time of each model varies slightly. Although the FLCN model has a longer detect time when the step size is set to 1, but its robustness is better, and even with an increase in step size, the FLCN model still performs well. As the step size increases, the detection time of the FLCN model also decreases significantly, demonstrating the usability of the FLCN model. By collaboratively considering the detection accuracy and the cost, it might be a better choice for our CNDSW framework to apply the FLCN model for the malicious code intrusion detection.

In Table 8, we illustrate the differences in accuracy and time between the CNDSW detection framework and using deep-learning alone for detection. The neural network model's batch size is set to 256. We found that the detection time using only the neural network model is significantly longer, more than two hundred times that of CNDSW. The rapidity of detection directly impacts the system's ability to promptly identify malicious intrusions, underscoring its critical importance in practical applications.

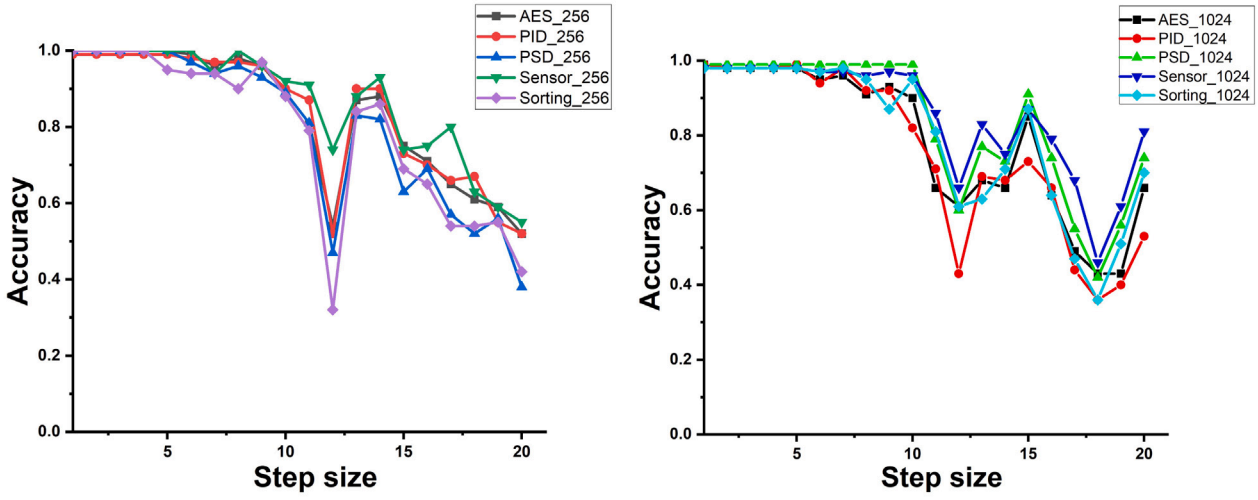
Considering the lightweight nature of the deep-learning algorithms used, particularly enhanced by the CNDSW framework's processing capabilities on traces, we argue that effective deployment and high-performance detection results can be achieved both at the edge and on larger centralized processors. These experimental findings indicate that while accuracy remains high, the feasibility of near real-time operation at the edge or within centralized processors warrants further exploration. The time factor not only affects responsiveness but also influences operational costs and scalability in real-world deployment scenarios. By leveraging lightweight deep-learning models and the efficient processing enabled by CNDSW, our approach demonstrates promising potential for robust intrusion detection across diverse computing environments.

Next, we conduct detection on captured traces inserting multiple library function trace segments. For each evaluation program trace, we randomly selected 2–3 library function traces and inserted them into different positions of the evaluation program. Similarly, using the CNDSW framework, we conduct the detection for the step size ranging from 1 to 20. For the ARM device, the experimental results show that the average detection accuracy is consistently higher than 99% when the sliding step size set to 1. However, when it comes to the case of the RISC device, the average detection accuracy drops to 85.7%. Table 9 summarizes the detailed detection results of the MLP-based CNDSW framework on captured traces with multiple malicious activities injected when the step size set to 1.

7.3. Detection on real-time traces

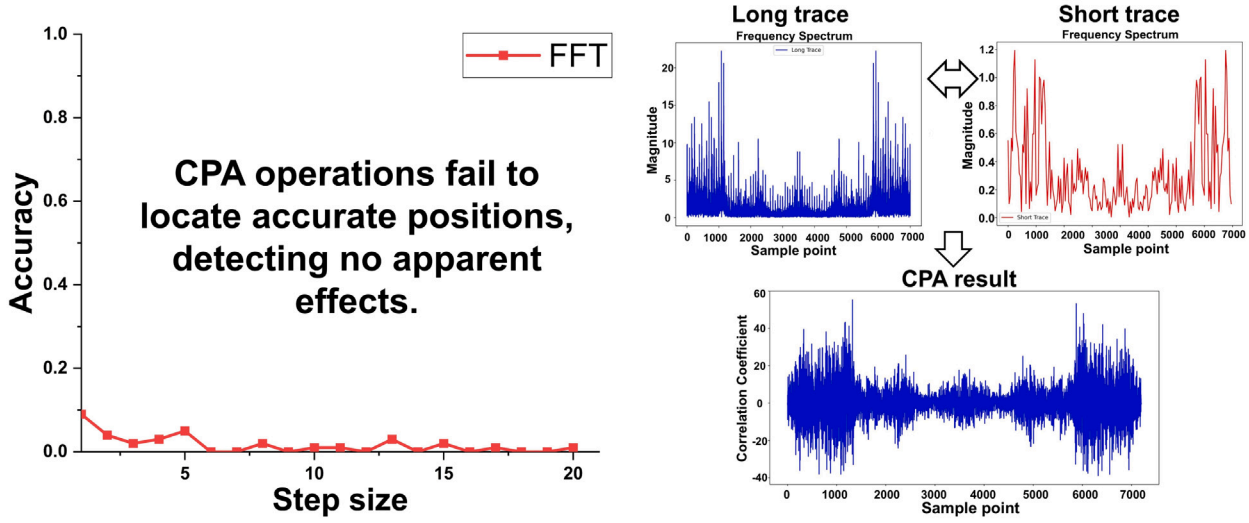
In this experiment, we further test the CNDSW framework on traces captured from running devices with malicious code injected. We first modify the firmware of the evaluation programs by randomly inserting library function operations. We randomly select one or multiple C library functions as the malicious activities to insert into some key locations of the firmware of evaluation programs and meanwhile run the CNDSW framework to monitor the generated real-time traces. For example, in AES, we place malicious library functions before the initial *AddRoundKey* operation, in the middle of two encryption rounds, and after the final encryption round. Afterwards, we capture real-time traces of evaluation programs with malicious function embedded from running devices.

At the same time, we conduct the monitoring process by using the CNDSW framework on real-time traces. For each evaluation program, we run 50 tests to get the average with the step size ranging from 1 to



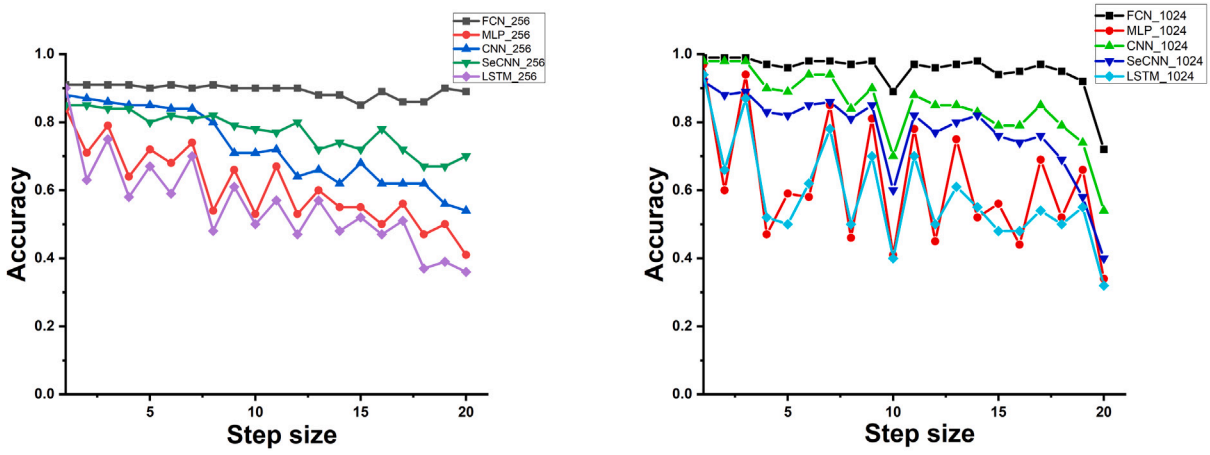
(a) Detection results of F_{256}. (b) Detection results of F_{1024}.

Fig. 13. Detection results of *FLCNN* models on ARM traces.



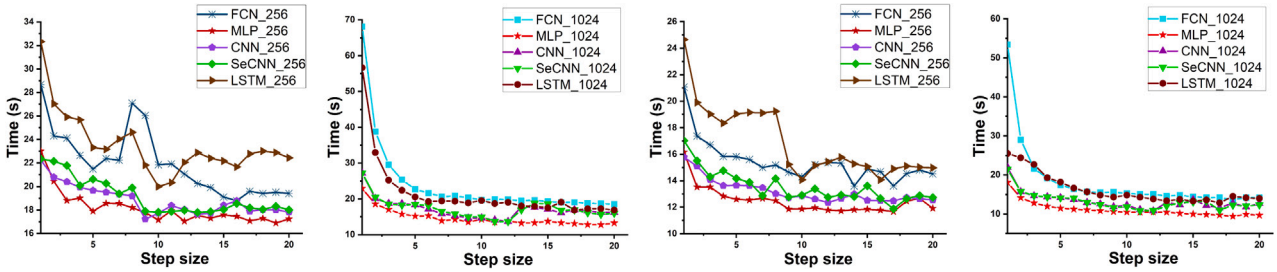
(a) Detection results on ARM traces with *FFT*. (b) Spectrum analysis process.

Fig. 14. The right figure shows the results of CPA detection after spectrum transformation, and the left figure shows the results of spectrum detection, with no obvious effect detected.



(a) Detection results on RISC traces (256). (b) Detection results on RISC traces (1024).

Fig. 15. Library function detection results of different models on traces captured from the RISC board.



(a) Detection time of models with the input size 256 on ARM traces. (b) Detection time of models with the input size 1024 on ARM traces. (c) Detection time of models with the input size 256 on RISC traces. (d) Detection time of models with the input size 1024 on RISC traces.

Fig. 16. Comparison of the library function detection time of different models on traces captured from the ARM board and the RISC board implementation of *AES-128*.

Table 8
Comparison of the detection accuracy and time between the CNDSW framework and the neural-network-only scheme on traces captured from the ARM board implementation of AES-128.

The CNDSW framework				The neural-network-only scheme			
CNDSW	Number of traces	Step 1 Acc	Detection time	Only NN	Number of traces	Step 1 Acc	Detection time
MLP	50 × 29	99.5%	22.96”	MLP	50 × 29	65%	4805”
CNN	50 × 29	99.5%	24.72”	CNN	50 × 29	76%	5208”
SeCNN	50 × 29	99.5%	24.79”	$SeCNN$	50 × 29	78%	5168”
LSTM	50 × 29	99.5%	44.48”	$LSTM$	50 × 29	81%	9641”
FLCN	50 × 29	99.5%	48.40”	$FLCN$	50 × 29	85%	10219”

Table 9
Detection results of the CNDSW framework with the MLP model on captured traces with multiple malicious activities injected when the step size set to 1.

Accuracy / Algorithm / Architecture	AES	PID	PSD	Sensor20	SensorSorting
ARM	100%	100%	98%	100%	100%
RISC	92.6%	86%	84.8%	80.2%	85%

Table 10
Detection results of the CNDSW framework with the MLP model on real-time traces with multiple malicious activities injected when the step size set to 1.

Accuracy / Algorithm / Architecture	AES	PID	PSD	Sensor20	SensorSorting
ARM	100%	99.8%	99.2%	98.2%	98%
RISC	89.6%	82%	80%	74%	78%

20. Table 10 summarizes the averaged detection accuracy of the MLP-based CNDSW framework with sliding step size of 1 on real-time traces from two target devices.

For the ARM device, we observe that the CNDSW framework still exhibits good performance on real-time traces. When the step size for dual sliding set to 1, the average detection accuracy for AES evaluation program in 10 real insertion cases reaches 100%. Besides, the average detection accuracy for the other four evaluation programs in 10 sets of test is above 98% on average. For the RISC device, we conduct the same experiment. The average detection accuracy of the CNDSW framework with the sliding step size set to 1 reaches 89.6%. In our study, the baseline model was originally trained on traces collected from ARM boards. Despite the specific nature of trace data from different platforms, we applied the same deep-learning model architecture directly to traces from RISC boards. This approach yielded satisfactory results across both platforms, indicating robustness in model transferability. However, optimizing performance further may necessitate platform-specific model training and structural adjustments through redesigning and fine-tuning. This strategy could potentially enhance detection accuracy, particularly in scenarios involving sensor-related traces.

In summary, the proposed CNDSW framework shows a great potential for control flow monitoring with an effective detection accuracy. By identifying malicious code intrusions at the function level, the scope of the detection objects are not limited to known attacks anymore.

7.4. Related work

Recent advancements in physical side-channel monitors have significantly addressed escalating security challenges posed by covert channels in embedded systems. Table 11 provides a comparative analysis of cutting-edge methodologies and their effectiveness in detecting malicious activities across various platforms.

The listed works showcase a range of innovative approaches, including power (Liu et al., 2016; Ding et al., 2020), EM (Khan et al., 2019), and impedance-based side-channel analysis (Awal and Rahman, 2023), alongside sophisticated techniques in the code intrusion detection and instruction classification. For instance, deep-learning models like LSTM of the Zeus scheme in Han et al. (2017) and conventional machine learning approach like SVM in Awal and Rahman (2023) illustrate the industry’s trend towards more robust detection frameworks. Zeus achieves an exceptional 98.9% accuracy in distinguishing between legitimate and malicious operations in Allen Bradley PLCs, highlighting its efficacy in practical scenarios. Similarly, Awal et al.’s SVM-based approach boasts a 98.6% detection accuracy with a remarkably low false-positive rate of 0.40%. However, as shown in Table 11, all existing works conduct the code intrusion detection at the block level while ignoring that a small function-level instruction intrusion may cause a significant damage in many security-critical fields. Although some works (Iyer et al., 2024; Awal and Rahman, 2023) do focus on the side channels generated during the execution of specific instructions, they only explore to which extent the captured traces can be classified to different instructions, which is still far away to the practical detection scenarios.

In contrast, our research introduces CNDSW, a pioneering method designed for detecting malicious activities at the library function level. Unlike traditional approaches focused on binary classification or sequence-to-sequence models, our method monitors the control flow at the function level, allowing for the detection of unknown attacks specifically involving library function operations. When an unexpected function call is detected, the system automatically triggers an alert, providing timely warnings to personnel. From the detected abnormal calls of library functions in the control flow, the user may preliminary forecast the potential threat type of the malicious activities without having to know the details.

Table 11
Comparison of related work in physical side-channel monitor.

Ref.	Side-channel	Target	Detection level	Model	Detection accuracy	Detected attack	Citation
Liu et al. (2016)	Power	8051 MCU	Code block	Markov+Viterbi	99.9%	Known	97
Han et al. (2017)	EM	Allen Bradley PLC	Legit/malicious	Zeus: LSTM	98.9%	Known	124
Khan et al. (2019)	EM	Arduino UNO	Legit/malicious	Idea	99.5%	Known	55
Ding et al. (2020)	Power	IoT Camera	Code block	DeepPower: Seq2Seq	90.4%	Known	41
Iyer et al. (2024)	EM	AT89S51 MCU	Instruction (Classification only)	Binary	99.0%	Known	2
Awal and Rahman (2023)	Impedance	ATmega328P	Instruction (Classification only)	SVM	98.6%	Unknown	4
Our Work	Power	STM32F3, ATmega128P	Function	CNDSW: CNN+CPA	99.0%	Unlimited	

CNDSW represents a significant advancement by enabling precise detection of malicious behavior within library functions, enhancing the system's ability to counter sophisticated attack vectors. Its successful deployment in operational environments underscores its scalability and adaptability, facilitating seamless integration into existing industrial control systems.

8. Conclusion

In this paper, we conduct an extensive investigation into physical side-channel analysis techniques, focusing on detecting malicious activities at the function level in embedded systems. Our research underscores the critical importance of precise detection methodologies in safeguarding both legacy and modern IIoT infrastructures against sophisticated cyber threats.

Besides, we introduce the CNDSW framework, a pioneering approach that integrates advanced correlation analysis techniques with neural networks. CNDSW leverages dual-sliding window technology to analyze real-time traces of power consumption, enabling it to accurately identify malicious behaviors at the function level on ARM and RISC-based microcontroller platforms. Importantly, our experiments have validated CNDSW's robust performance, consistently achieving detection accuracies exceeding 99%, surpassing traditional methods in IIoT security.

Our evaluation highlights CNDSW's capability to monitor and detect malicious activities with unprecedented granularity, ensuring the integrity and security of critical systems where firmware updates are impractical or impossible. Despite its successes, challenges remain in enhancing dataset diversity and optimizing model performance under varying environmental conditions. Ongoing research will focus on expanding our dataset to encompass a wider range of attack scenarios, refining algorithm-specific models, and investigating the root causes of power consumption leakage to fortify CNDSW against evolving cyber threats.

Looking forward, our research agenda aims to advance embedded system security by further refining CNDSW and exploring its applicability in diverse IIoT environments. We anticipate deploying CNDSW in real-world control systems, where it can be integrated as a component of the industrial IoT ecosystem. Specifically, CNDSW can be deployed on host systems to analyze power consumption signals collected from various endpoints, thereby detecting malicious software. Additionally, it could be adapted to run on small edge computing boards, such as Google Coral Dev Board, which would function as micro computational units. This approach would reduce computational demands while continuously uploading detection information.

When considering the practical deployment of physical side-channel monitors in real-world systems, CNDSW appears to be a promising option, particularly for critical infrastructure. The power consumption data collection is relatively straightforward, allowing for the system to be deployed without requiring modifications to the existing circuit setups. This approach reduces the complexity of retrofitting current systems and enhances security in key environments. However, large-scale deployment remains challenging, especially in complex and widely distributed IIoT environments. In these cases, an alternative approach, such as using systems that detect malicious software through electromagnetic signal collection, might be more suitable for broader implementation. This method could offer similar detection capabilities without interfering with existing circuits and provide greater flexibility and scalability.

In future work, we plan to explore electromagnetic signal-based side-channel analysis to complement CNDSW's existing capabilities and evaluate its applicability in more extensive industrial environments. By addressing these challenges and advancing detection capabilities, we contribute to the resilience of critical industrial systems against emerging cyber threats. In Table 11 we list the results on physical side-channel monitors in recent years and a comparison of the various aspects.

In conclusion, this study introduces CNDSW as a state-of-the-art detection framework that sets a new standard in embedded system security. By integrating advanced correlation analysis techniques, CNDSW not only enhances detection accuracy but also paves the way for future innovations in securing IIoT infrastructures worldwide.

CRediT authorship contribution statement

Dalin He: Writing – original draft, Methodology, Investigation, Formal analysis, Data curation. **Huanyu Wang:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Conceptualization. **Tuo Deng:** Writing – review & editing, Validation, Investigation. **Jishi Liu:** Writing – review & editing, Investigation. **Junnian Wang:** Writing – review & editing, Validation, Supervision, Resources, Project administration, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgment

This research was supported by the Postgraduate Scientific Research Innovation Project of Hunan Province (No. CX20240881).

References

Abadi, M., Budiu, M., Erlingsson, U., Ligatti, J., 2009. Control-flow integrity principles, implementations, and applications. ACM Trans. Inf. Syst. Secur. 13 (1), 1–40.

Astrachan, O., 2003. Bubble sort: An archaeological algorithmic analysis. ACM SIGCSE Bull. 35 (1), 1–5.

Awal, M.S., Rahman, M.T., 2023. Disassembling software instruction types through impedance side-channel analysis. In: 2023 IEEE International Symposium on Hardware Oriented Security and Trust. HOST, IEEE, pp. 227–237.

Bhamare, D., Zolanvari, M., Erbad, A., Jain, R., Khan, K., Meskin, N., 2020. Cybersecurity for industrial control systems: A survey. Comput. Secur. 89, 101677.

Boyes, H., Hallaq, B., Cunningham, J., Watson, T., 2018. The industrial internet of things (IIoT): An analysis framework. Comput. Ind. 101, 1–12.

Bühlmann, P., Van De Geer, S., 2011. Statistics for High-dimensional Data: Methods, Theory and Applications. Springer Science & Business Media.

Cao, P., Zhang, H., Gu, D., Lu, Y., Yuan, Y., 2022. AL-pa: Cross-device profiled side-channel attack using adversarial learning. In: Proceedings of the 59th ACM/IEEE Design Automation Conference. pp. 691–696.

Das, T.K., Adepu, S., Zhou, J., 2020. Anomaly detection in Industrial Control Systems using Logical analysis of Data. Comput. Secur. 96, 101935.

Di Pinto, A., Dragoni, Y., Carcano, A., 2018. TRITON: The first ICS cyber attack on safety instrument systems. Proc. Black Hat USA 2018, 1–26.

Ding, F., Li, H., Luo, F., Hu, H., Cheng, L., Xiao, H., Ge, R., 2020. DeepPower: Non-intrusive and deep learning-based detection of IoT malware using power side channels. In: ASIA CCS '20, Association for Computing Machinery, New York, NY, USA, pp. 33–46.

F-Secure Labs, 2016. BLACKENERGY and QUEDAGH: The convergence of crimeware and APT attacks. Accessed on: Date you accessed the report. URL https://www.f-secure.com/v-descs/backdoor-w32-blackenergy.shtml.

Falliere, N., Murchu, L.O., Chien, E., et al., 2010. W32. stuxnet dossier, White paper, Symantec corp, Security response https://docs.broadcom.com/docs/security-response-w32-stuxnet-dossier-11-en.

Feng, J., Jacques, T., Abari, O., Sehatbakhsh, N., 2023. Everything has its Bad Side and Good Side: Turning processors to low overhead radios using side-channels. In: Proceedings of the 22nd International Conference on Information Processing in Sensor Networks. pp. 288–301.

Han, Y., Chan, M., Aref, Z., Tippenhauer, N.O., Zonouz, S., 2022. Hiding in Plain Sight? On the efficacy of power Side Channel-Based Control Flow Monitoring. In: 31st USENIX Security Symposium. pp. 661–678.

Han, Y., Christoudis, I., Diamantaras, K.I., Zonouz, S., Petropulu, A., 2019. Side-channel-based code-execution monitoring systems: A survey. IEEE Signal Process. Mag. 36 (2), 22–35.

Han, Y., Etigowni, S., Liu, H., Zonouz, S., Petropulu, A., 2017. Watch me, but don't touch me! contactless control flow monitoring via electromagnetic emanations. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1095–1108.

Hemsley, K., Fisher, R., 2018. A history of cyber incidents and threats involving industrial control systems. In: Staggs, J., Shenoi, S. (Eds.), Critical Infrastructure Protection XII. Springer International Publishing, Cham, pp. 215–242.

Huang, T., Yang, G., Tang, G., 1979. A fast two-dimensional median filtering algorithm. IEEE Trans. Acoust. Speech Signal Process. 27 (1), 13–18.

Iyer, V.V., Thimmaiah, A., Orshansky, M., Gerstlauer, A., Yilmaz, A.E., 2024. A hierarchical classification method for high-accuracy instruction disassembly with near-field EM measurements. ACM Trans. Embed. Comput. Syst. 23 (1).

Jayalaxmi, P., Saha, R., Kumar, G., Alazab, M., Conti, M., Cheng, X., 2023. Pignus: a deep learning model for ids in industrial Internet-of-things. Comput. Secur. 132, 103315.

Johnson, M.A., Moradi, M.H., 2005. PID Control Technology: New Identification and Design Methods, Chapters 1 and 2. pp. 1–46.

Khalil, S.M., Bahsi, H., Korõtko, T., 2023. Threat modeling of industrial control systems: A systematic literature review. Comput. Secur. 103543.

Khan, H.A., Sehatbakhsh, N., Nguyen, L.N., Callan, R.L., Yeredor, A., Prvulovic, M., Zajić, A., 2019. IDEA: Intrusion detection through electromagnetic-signal analysis for critical embedded and cyber-physical systems. IEEE Trans. Dependable Secure Comput. 18 (3), 1150–1163.

Kocher, P.C., 1996. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Advances in Cryptology—CRYPTO'96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings 16. Springer, pp. 104–113.

Kocher, P., Jaffe, J., Jun, B., 1999. Differential power analysis. In: Wiener, M. (Ed.), Advances in Cryptology — CRYPTO' 99. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 388–397.

Kuang, B., Fu, A., Zhou, L., Susilo, W., Zhang, Y., 2020. DO-RA: Data-oriented runtime attestation for IoT devices. Comput. Secur. 97, 101945.

Ligatti, J., Abadi, M., Bidiu, M., Erlingsson, U., 2005. Control flow integrity. In: Proceedings of the 12th ACM Conference on Computer and Communications Security. p. 1.

Liu, Y., Wei, L., Zhou, Z., Zhang, K., Xu, W., Xu, Q., 2016. On code execution tracking via power side-channel. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 1019–1031.

Maillard, J., Hiscock, T., Lecomte, M., Clavier, C., 2023. Side-Channel Disassembly on a System-on-Chip: A practical feasibility study. Microprocess. Microsyst. 101, 104904.

Mekala, S.H., Baig, Z., Anwar, A., Zeadally, S., 2023. Cybersecurity for industrial IoT (IIoT): Threats, countermeasures, challenges and future directions. Comput. Commun. 208, 294–320.

Nazari, A., Sehatbakhsh, N., Alam, M., Zajic, A., Prvulovic, M., 2017. EDDIE: EM-based detection of deviations in program execution. In: Proceedings of the 44th Annual International Symposium on Computer Architecture. pp. 333–346.

O'flynn, C., Chen, Z., 2014. Chipwhisperer: An open-source platform for hardware embedded security research. In: Constructive Side-Channel Analysis and Secure Design: 5th International Workshop, COSADE 2014, Paris, France, April 13–15, 2014. Revised Selected Papers 5. Springer, pp. 243–260.

Qin, X., Jiang, F., Dong, C., Doss, R., 2024. A hybrid cyber defense framework for reconnaissance attack in Industrial Control Systems. Comput. Secur. 136, 103506.

Rendón-Segador, F.J., Álvarez-García, J.A., Varela-Vaca, A.J., 2023. Paying attention to cyber-attacks: A multi-layer perceptron with self-attention mechanism. Comput. Secur. 132, 103318.

Rijmen, V., Daemen, J., 2001. Advanced encryption standard. vol. 19, National Institute of Standards and Technology, p. 22.

Sengupta, J., Ruj, S., Bit, S.D., 2020. A comprehensive survey on attacks, security issues and blockchain solutions for IoT and IIoT. J. Netw. Comput. Appl. 149, 102481.

Sifre, L., Mallat, S., 2014. Rigid-motion scattering for texture classification. arXiv preprint arXiv:1403.1687.

Staib, M., Moradi, A., 2023. Deep learning side-channel collision attack. IACR Trans. Cryptogr. Hardw. Embed. Syst. 422–444.

Timon, B., 2019. Non-profiled deep learning-based side-channel attacks with sensitivity analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. 107–131.

Ugurlu, E.M., Yilmaz, B.B., Zajić, A., Prvulovic, M., 2021. Pitem: Permutations-based instruction tracking via electromagnetic side-channel signal analysis. IEEE Trans. Comput. 71 (5), 1156–1169.

Van Bulck, J., Moghimi, D., Schwarz, M., Lippi, M., Minkin, M., Genkin, D., Yarom, Y., Sunar, B., Gruss, D., Piessens, F., 2020. LVI: Hijacking transient execution through microarchitectural load value injection. In: 2020 IEEE Symposium on Security and Privacy. SP, IEEE, pp. 54–72.

Vanhoef, M., Piessens, F., 2017. Key reinstallation attacks: Forcing nonce reuse in WPA2. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1313–1328.

Wang, H., 2024. Amplitude-modulated EM side-channel attack on provably secure masked AES. J.f Cryptogr. Eng. 1–13.

Wang, R., Wang, H., Dubrova, E., Brisfors, M., 2021. Advanced far field EM side-channel attack on AES. In: Proceedings of the 7th ACM on Cyber-Physical System Security Workshop. pp. 29–39.

Yang, X., Howley, E., Schukat, M., 2024. ADT: Time series anomaly detection for cyber-physical systems via deep reinforcement learning. Comput. Secur. 103825.

Yarom, Y., Falkner, K., 2014. FLUSH+ RELOAD: A high resolution, low noise, L3 cache Side-Channel attack. In: 23rd USENIX Security Symposium. pp. 719–732.

Youngworth, R.N., Gallagher, B.B., Stamper, B.L., 2005. An overview of power spectral density (PSD) calculations. Opt. Manuf. Test. VI 5869, 206–216.

Zhao, Z., Li, Z., Yu, J., Zhang, F., Xie, X., Xu, H., Chen, B., 2024. CMD: Co-analyzed IoT malware detection and forensics via network and hardware domains. IEEE Trans. Mob. Comput. 23 (5), 5589–5603.

Dalin He was born in Changsha City, Hunan Province, China, in 2000. He received the bachelor's degree in Optoelectronic Information Science and Engineering from the Hunan University of Science and Technology, Hunan, China, in 2022, where he is currently pursuing the master's degree in IC. His research interests include hardware encryption, side channel analysis, and malicious code intrusion detection.

Huanyu Wang was born in Jinchang City, Gansu Province, China, in 1995. He received the Ph.D. and M.S. degree in Information and Communication Technology from KTH Royal Institute of Technology, Stockholm, Sweden, in 2023 and the B.S. degree in Electronic Information Engineering from Dalian University of Technology, Dalian, China. He is currently an Assistant Professor at the School of Computer Science and Engineering at Hunan University of Science and Technology. His current research interest includes hardware security, side-channel analysis and deep learning based applications.

Tuo Deng was born in Qingyang City, Gansu Province, China, in 1998. He received the bachelor's degree in Optoelectronic Information Science and Engineering from the Hunan University of Science and Technology, Hunan, China, in 2021, where he is currently pursuing the master's degree in IC. His research interests include hardware encryption, side channel analysis.

Jishi Liu was born in Shaoyang City, Hunan Province, China, in 1999. He received his bachelor's degree in Electronic Information Science and Technology in 2021. Currently, he is pursuing his master's degree in IC at Hunan University of Science and Technology, Hunan, China. His research interests include image detection, edge computing and deep learning based applications.

Junnian Wang received the bachelor's degree from the Department of Modern Physics, Lanzhou University, in 1991, the master's degree in radio physics from the School of Information Science and Engineering, Lanzhou University, in 2000, and the Ph.D. degree in control theory and control engineering from the School of Information Science and Engineering, Central South University, in 2006. He has undertaken four projects of the National Natural Science Foundation of China and more than ten other provincial and ministerial level research projects. He has published more than 50 scientific papers, including more than 20 SCI/EI papers. His research interests include deep learning, intelligent information processing, and fault diagnosis. He received the Second Prize of Hunan Provincial Science and Technology Progress Award.