WiP Paper: BUSted Second Stop! A First Step for Breaking Cryptographic Applications on MCU-based IoT Devices

André Barbosa, Cristiano Rodrigues, Tiago Gomes, and Sandro Pinto
Centro ALGORITMI / LASI, Universidade do Minho
{pg50216,id9492}@uminho.pt,{mr.gomes,sandro.pinto}@dei.uminho.pt

ABSTRACT

IoT devices have become ubiquitous in modern society. However, the security of the newly connected devices, typically powered by microcontrollers (MCUs), has not been thoroughly investigated. One of the overlooked types of attacks on MCU-base IoT devices are microarchitectural side-channel attacks. Until recently, microarchitectural side-channel attacks on MCUs were deemed unlikely due to their perceived simplicity. The BUSted attack has challenged this assumption by unveiling a novel timing-based microarchitectural side channel on the bus interconnect, impacting all mainstream MCUs. BUSted introduces two variants, single-run and multi-run, however only the former was implemented against a simple Smart Lock application. This paper takes the first steps toward exploiting more advanced targets, such as cryptographic applications, by exploring the BUSted multi-run variant. We introduce practical methods for profiling cryptographic applications and address the limitations of the single-run variant, particularly the impractical high profiling time. We propose a new BUSted interrupt-based multi-run methodology to trace the victim on a per-clock basis. Our approach includes a trampoline and sliding window technique to reduce stack interference and enhance profiling granularity. We demonstrate the effectiveness of this solution by successfully profiling the TinyCrypt implementation of AES-CBC.

CCS CONCEPTS

• **Security and privacy → Embedded systems security**.

KEYWORDS

Side-Channels, Microarchitecture, Microcontrollers, Bus, Multi-Run

1 INTRODUCTION

The Internet of Things (IoT) has connected common everyday-use devices, such as refrigerators, watches, and light bulbs, to the Internet. While this connectivity has introduced new functionalities, it has also exposed these devices to remotely exploitable attacks. One of the stealthiest types are the so-called microarchitectural side-channel attacks. This class of attacks exploits the side effects of computational properties to extract sensitive information by leveraging secret dependent traces left in the processor's microarchitectural states. Besides being prominent in powerful processors, such as application processor units (APUs), with several attacks unveiled in the last decade [1, 4, 5, 7–10, 14, 21], these attacks were often overlooked in microcontrollers (MCUs), which are the core of the majority of IoT devices, being Nemesis [18] and BUSted [16] the only know attacks unveiled so far. The Nemesis attack [18] demonstrated information leakage from an MCU by leveraging the timing differences inflicted by the CPU interrupt logic. However, it was only demonstrated in the not-very-widespread MSP430 MCU.

BUSted [16] exploits timing differences in the arbitration logic of the bus interconnect through the contention inflicted when multiple bus mains[1] try to access the same bus secondary. It has been demonstrated scalable across the full MCU spectrum, including PIC, RISC-V, and Armv6/7/8 MCUs. This attracted the attention of major security-oriented news outlets [12, 15] and prompted Arm [2] and WolfSSL [19] to release security advisories. The authors of the BUSted attack proposed two variants: the single-run and multi-run variants. The first variant was implemented to exploit a Smart Lock application deployed in the secure world of two Armv8-M MCUs (Arm Cortex-M23 and -M33). The multi-run variant was theorized and speculated but still remains to be demonstrated. We argue that implementing the multi-run variant of the BUSted attack targeting cryptographic algorithms is more challenging than the authors claim to be. However, a methodology capable of compromising cryptographic-hardened applications holds the power to jeopardize all IoT infrastructure. Thus, we decided to investigate: *What are the underlying challenges of implementing the BUSted attack against a cryptographic algorithm?*

In this paper, and as a first step towards answering this question, we address the limitations of the BUSted single-run variant when applied to compute-intensive code, such as cryptographic applications, and show that the BUSted multi-run variant is a viable solution. Our primary observation is that the BUSted single-run methodology is unsuitable for compute-intensive code due to its requirement for profiling each victim clock cycle. This results in a linear increase in profiling time as the clock cycles of the victim application increase. To substantiate our findings, we conducted preliminary experiments profiling two cryptographic applications: AES tinyCrypto and RSA WolfSSL. The results demonstrate the impracticality of the single-run attack methodology for compute-intensive code, for instance, the total time to collect a single trace on RSA WolfSSL is in the the order of magnitude of years.

To address this issue, we propose a new BUSted profiling interrupt-based approach, similar to Nemesis [18] and SGX-Step [17]. This method interrupts the victim at each clock cycle[2], gathers the side-channel information, and triggers the mechanism to interrupt the next clock cycle. Transitioning from a single-run to an interrupt-based multi-run methodology posed two significant challenges: (i) the automatic stacking by the CPU upon each interrupt interferes with the detection of contention, and (ii) the interrupt-based approach reduces trace granularity from one clock cycle to multiple clock cycles. To overcome the former challenge, we introduced a novel approach using a trampoline setup code, a code interposed between the interrupt handler and the victim code, to avoid stack interference. For the latter, we applied a sliding window technique,

[1] Authors replaced master and slave terminology with keywords main and secondary.
[2] This approach expands BUSted assumptions to consider the victim can be interrupted.

which fuses several partial traces into one full trace, mitigating the granularity degradation caused by the interrupt-based approach.

We validate the effectiveness of the proposed BUSted multi-run methodology by tracing the TinyCrypt [11] implementation of the AES-CBC cryptographic algorithm. We obtained distinct traces for different plaintexts, demonstrating early evidence that the BUSted multi-run methodology can be applied to cryptographic targets. This research provides the first steps towards the full exploitation of cryptographic applications, by providing promising empirical evidence on the applicability of BUSted to compute-intensive code. In summary, the paper makes the following contributions: **1)** We address the inherent challenges in implementing the multi-run variant of the BUSted attack; **2)** We provide the design of the BUSted multi-run variant profiling phase tailored for cryptographic applications; and **3)** We evaluate the effectiveness of the proposed solution by profiling a cryptographic algorithm with promising results.

2 OVERVIEW, MOTIVATION, AND CHALLENGES

In this section, we present the adversary model and provide an overview of the BUSted attack and its variants: single-run and multi-run. We then motivate this work by conducting a preliminary study on the limitations of the BUSted single-run attack methodology.

2.1 Scope and Adversary Model

Scope. Similar to BUSted, we target devices with single-core MCUs, pervasive in modern IoT devices. MCUs are specialized processing units with limited computational power and memory, designed with simple microarchitectures featuring 2-3 pipeline stages, small caches, and no advanced predictors. This design prioritizes energy efficiency and cost effectiveness in resource-constrained environments. MCUs do not support virtual memory and are highly deterministic, consistently producing the same results under identical conditions. Prominent examples include the MSP470, Arm Cortex-M4/7, and new security-oriented Arm Cortex-M23/33.

Adversary model. In line with BUSted, we assume the attacker and the victim operate in different security domains. The attacker has knowledge of the victim's code, such as an open-source cryptographic library containing secret-dependent control flows. During the profiling phase, we assume the spy has control over the victim and the environment. Profiling is conducted offline in a lab setting, replicating a real attack scenario with the spy controlling all variables. The spy and the victim share the same physical memory bank, and the victim operates upon the attacker's requests. We also assume the attacker has full access to and control over one isolated domain and its resources, including peripherals (e.g., timer) and bus mains (e.g., DMA). Unlike BUSted, we assume that interrupts are enabled, the attacker can interrupt the victim's execution, and there are multiple opportunities to extract the secret. We do not consider physical attacks such as power side channels or fault injections, which require physical access to the target device.

2.2 BUSted Attack

BUSted is a side-channel attack that explores the side effects of the MCU bus interconnect arbitration logic to bypass security guarantees enforced by memory protection primitives. BUSted leverages

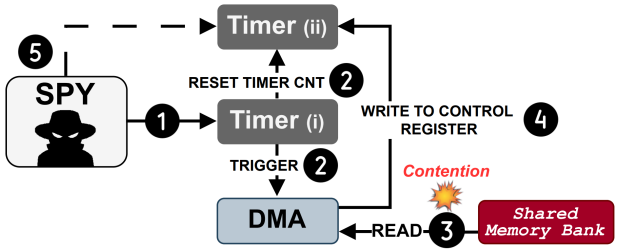

Figure 1: Illustration of the profiling SGN using BUSted single-run attack variant methodology.

timing differences created by contention on the bus interconnect arbitration logic, to capture the victim's memory access patterns. The principle behind this attack is that when multiple bus mains (e.g., CPU or DMA) access the same secondary resources concurrently, one main's access will be delayed in time. This delay is what creates the microarchitectural channel leveraged by BUSted. The overall attack is divided into (i) *profiling phase* and (ii) *exploitation phase*. In the profiling phase, the spy creates a template (i.e., side-channel patterns) by tracing each vulnerable path[3] in the victim. In the exploitation phase, the spy compares the trace of the victim's execution with the template to determine the secret.

Variants. Depending on the victim's application, the exploitation phase follows different methodologies, creating two possible BUSted variants, **single-run** and **multi-run**. In the single-run variant, the attacker collects all side-channel data on the fly to reveal the secret while the victim is computing over it. As the secret is ephemeral (i.e., it changes in each victim run), the attacker just has one chance to leak it. In the multi-run variant, the attacker repeatedly executes the target victim to collect multiple side-channel observations. The same secret (e.g., private key) is used in each execution, and the corresponding side-channel data is collected for every run. The attacker analyzes this data to identify patterns related to the secret data, which will reveal its content.

Hardware gadgets and SGNs. BUSted introduced the concept of hardware gadgets and smart gadget networks (SGNs) to enable side-channel attacks using stateless microarchitectural components (i.e., bus interconnect) on single-core MCUs. Hardware gadgets are memory-mapped peripherals commonly found in MCUs (e.g., timers and DMAs) that can be programmed to perform specific spy logic (e.g., memory accesses). The key observation is that these hardware gadgets can operate without CPU intervention, allowing them to automate concurrent spy logic while the victim's code runs on the (single-core) CPU. They can monitor victims' memory access patterns at clock cycle granularity. SGNs interconnect multiple hardware gadgets to perform more advanced attacker logic.

2.3 Preliminary Study and Limitations

The BUSted attack was evaluated using a simple Smart Lock Application, i.e., single-run variant. To assess BUSted's performance on compute-intensive code, we profiled three different applications: the Smart Lock Application exploited by BUSted (as reference), the TinyCrypt AES-CBC algorithm [11], and the wolfSSL RSA algo-

[3]Vulnerable path refers to any victim's control flows dependent on secret data.

rithm [20]. To collect the traces, we implemented an SGN mimicking the BUSted single-run methodology.

Single-run profiling SGN. Figure 1 depicts a high-level view of the SGN architecture and its inner workings. Firstly, the spy starts timer*(i)*. Once timer*(i)* completes counting the desired cycles, it triggers ② the DMA and resets timer*(ii)* to zero. Then, the DMA reads ③ from a predefined shared memory bank, the same one accessed by the victim, causing contention. The DMA transfers data ④ into the control register of the free-running timer*(ii)*, stopping it immediately. After timer*(ii)* stops ⑤, the spy reads its counter value. A higher-than-normal value in timer*(ii)* indicates that the DMA took longer to stop it, signaling contention. This process repeats until the victim executes the last instruction.

Results and limitations. The results[4], in Table 1, demonstrate that the BUSted single-run methodology is unsuitable for mounting attacks against compute-intensive code since the profiling time grows linear O(n) with the growing number of clock cycles of the victim. The BUSted single-run approach measures contention in one clock cycle per victim run, requiring the profiling SGN to execute exactly as many times as the victim, e.g., 41476 times for AES-CBC on STM32F407. This limitation renders this BUSted variant ineffective for applications with long execution times. For instance, getting a single trace of wolfSSL RSA would require more than a year on STM32F407 and around fourteen years on STM32L552.

2.4 Challenges

To implement an effective profiling methodology for cryptographic algorithms, the spy must meet three requirements: **(R1)** The spy must **create contention** in a shared memory bank between the spy and the victim. **(R2)** The spy must **record contention** on the bus interconnect caused by the victim's secondary port. **(R3)** The spy must perform **multiple traces** within a reasonable timeframe.

Challenges. We share two challenges with the BUSted attack, **C1** and **C2**, and found a new challenge **C3**. ***(C1)*** The spy cannot access past microarchitecture states. The bus interconnect is stateless, requiring timing differences to be assessed in real-time. ***(C2)*** The spy and the victim cannot run concurrently since MCUs are single-core. ***(C3)*** BUSted single-run methodology is not suitable for compute-intensive code. We aim at significantly more complex victims with a different attack approach (multi-run), which requires assessing multiple traces that would be prohibitively time-consuming with the current BUSted single-run profiling approach.

3 DESIGN

Our goal is to develop an effective and reliable profiling method that is capable of tracing compute-intensive applications. In this section, based on the profiling methodology defined in BUSted, we present the design of an augmented profiling method that can be used to profile a cryptographic application.

3.1 Augmented Profiling Methodology

To address ***C3***, we introduce an interrupt-based trace approach. Figure 2 depicts the SGN we implemented in §2.3, now adapted for the new multi-run approach. The victim code is interrupted at each clock cycle, allowing the SGN to gather side-channel information. We enhanced the single-run SGN with additional hardware, highlighted in Figure 2 with a red watermark. This hardware includes a new timer*(iii)*, which uses hardware interrupts to stop the victim code and switch to the side-channel information-gathering routine (i.e., ISR). Timer(iii) enables the SGN to shift from generating contention in **one** clock cycle to creating contention in **every** clock cycle per victim execution, allowing for much faster profilings, making the SGN meet **(R3)**. The multi-run SGN flow is similar to the single-run, but in this case, when timer*(i)* finishes counting the desired cycles, it also ② triggers the new timer*(iii)*. Timer*(iii)* then starts countdown, and upon completion, it triggers an interrupt that halts the victim code and switches to the ISR. The ISR restarts the hardware gadgets and triggers another interrupt right after the processor switches from the ISR back to the victim code. This repeats in a loop until the victim code executes the last instruction.

Victim	STM32F407 (@168 MHz)		STM32L552 (@110 MHz)	
	Execution Cycles	Profiling Time	Execution Cycles	Profiling Time
Smart Lock App.	2637	39ms	7480	508ms
AES (TinyCrypt)	41476	16 s	57895	30 s
RSA (wolfSSL)	81931195	432 days	220545898	14 years

Table 1: Profiling using BUSted single-run methodology on STM32F407 (Cortex-M4) and STM32L552 (Cortex-M33).

3.2 Dealing with Stack Interference

Design challenge. On Arm Cortex-M architecture, after an interrupt event is triggered, the processor first saves (pushes) several registers onto the stack before entering the ISR. This register state-saving operation is called stacking. At the end of the ISR, these registers are restored (popped) to the register bank, allowing the processor to resume its previous execution state, an operation called unstacking. Both operations are done automatically by hardware and cannot be disabled by software. Both stacking and unstacking operations are performed in the SRAM, which interferes with SGN memory accesses and prevents it from creating contention on the bus, precluding **(R1)** requirement. This issue is caused by the bus arbitration logic, i.e., round-robin policy. When multiple bus mains (e.g., CPU and DMA) request access to the bus simultaneously, the arbiter grants access in a cyclic order, ensuring each main gets a fair chance to use the bus. The SGN's DMA is configured inside the ISR to access memory and create contention in the first clock after the ISR ends execution. However, due to the unstacking (which happens on the ISR exit and is performed by the CPU), the SGN's DMA is always granted access to the bus (right after unstacking), thus not experiencing any delay hence not seeing contention.

Solution. We introduce a **Trampoline** approach, illustrated in Figure 3, to eliminate unstacking interference by running a specific intermediary logic before returning to the victim code. Without the trampoline, execution flows directly from the ISR Ⓐ to the victim code Ⓒ. With the trampoline, execution moves from the ISR Ⓐ to the trampoline setup code Ⓑ (where unstacking occurs) and then to the victim code Ⓒ. This intermediary step allows us to remove stack interference and create contention satisfying **(R1)**

[4]For the wolfSSL RSA application, we extrapolate the total profiling time from the average profiling time of the first 10k clock cycles.

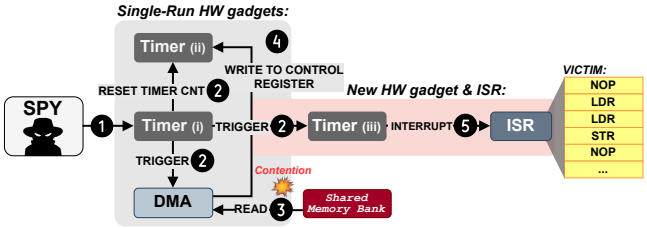

Figure 2: SGN setup with the interrupt-based trace approach.

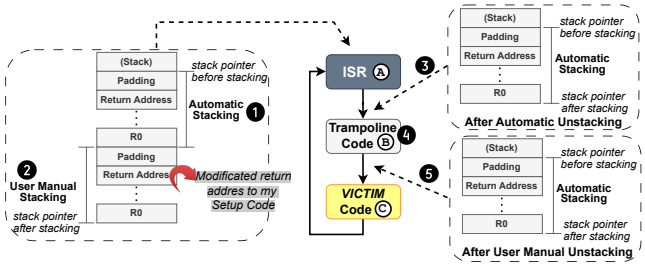

Figure 3: SGN interrupt-based with trampoline setup code.

requirement. The trampoline process is as follows. On ISR entry, the core automatically saves the state context on the stack (1), i.e., xPSR, Return Address, R12, and R0-R3. On the ISR, we emulate the stacking, creating a second copy of it. In this process, the return address is changed to return to our trampoline setup code (2). Upon ISR exit, the core does automatic unstacking from the copy of the stack we created, diverting execution to our trampoline setup code (3). Within our setup code (4), we reconfigure the profiling SGN, and before resuming the victim execution, we pop the stack, emulating the ISR exit. Lastly, a new profiling iteration is triggered after the victim advances its execution in one clock (5).

3.3 Dealing with Low Granularity

Design challenge. We need to obtain a high-granularity victim's trace to maximize the side-channel information obtained during the profiling phase. Ideally, we want to distinguish whether there was contention (or not) in every clock cycle. While not mandatory, this increases the attacker's chance of getting microarchitecture leakage from the victim's execution. However, the introduction of the interrupt-based approach and the trampoline reduced the overall granularity of the victim's trace, resulting in a loss of side-channel information in some of the victim clock cycles.

Solution. To address this challenge, we implemented a **Sliding Window** approach. Initially, a victim trace is performed in the first clock cycle of the victim. Then, based on the size of the sliding window, which corresponds to the granularity of the trace, the spy conducts additional traces, each shifted by one clock cycle, to cover all victim code. In the end, all partial traces are combined to create a single comprehensive trace that encompasses full victim execution with a granularity of one clock cycle. Figure 4 illustrates an example of a sliding window. If the trace granularity is four clock cycles, a sliding window of size four is required. This method enables monitoring of all the victim's clock cycles. As illustrated in Figure 4, a standard trace (A), without using the sliding window technique, can monitor only three out of twelve clock cycles of the victim. In contrast, the sliding window (B), by combining all traces, achieves full coverage of the victim's execution.

4 IMPLEMENTATION AND EVALUATION

We implemented and evaluated the BUSted multi-run variant on the STM32F407 and STM32L552, which feature Arm Cortex-M4 and Cortex-M33, respectively. From now on, we will refer STM32F407 as F4 and STM32L552 as L5. In line with the BUSted attack, we also focus on Arm MCUs due to their mass proliferation in IoT devices.

4.1 Profiling Implementation

Profiling setup. In our SGN setup, the spy leverages timers to generate precise intervals for initiating DMA transfers and measuring their durations. Furthermore, the spy uses the DMA as a malicious bus main to create contention on shared memory banks by repeatedly executing the same transfer. Considering the spy's complete control over the victim within its domain, the following steps outline our profiling setup: **1)** The spy configures the peripherals and builds the SGN; **2)** The spy triggers the SGN; **3)** The victim is invoked and begins executing its instructions; **4)** At a certain point, as the victim executes, contention arises and an interrupt is raised; **5)** Upon each interrupt, the control shifts back to the spy, allowing it to gather side-channel information (DMA latency) of the clock/instruction monitored; **6)** The spy triggers the SGN again in preparation for interrupting the next instruction. This cycle repeats until all instructions executed by the victim are traced.

Profiling Implementation. The timer's interrupt must arrive precisely at the first clock cycle of the victim. We configured timers on both platforms to generate interrupts upon reaching a pre-defined threshold. With the SGN fully configured, successful exploitation relies on achieving precise timing during SGN reconfiguration and resuming the victim execution. There is no margin for deviation to maintain accuracy and contention. The SGN was implemented on two different platforms with distinct architectures and processors, requiring us to manually fine-tune the SGN on each board.

Resources used. Our profiling SGN requires several platform resources, i.e., peripherals. On the F4 platform, the SGN uses 3 out of 14 timers and 1 out of 16 DMA channels, accounting for 9.3% of the total peripherals. Similarly, on the L5 platform, the SGN uses 3 out of 17 timers and 1 out of 16 DMA channels, representing 5.4% of the total peripherals. While our profiling SGN requires one more timer than BUSted's, this does not affect the feasibility of the attack, as most MCUs typically have an abundance of timers.

4.2 Profiling Evaluation

Profiling metrics. Table 2, provides the profiling metrics (i.e., granularity and contention detection rate) from tracing a simple victim comprising 1000 memory accesses. Victim execution times and single-run contention points serve as references. We profiled without a sliding window to evaluate the performance of a unique trace. We found that on F4, the profiling (as expected) is optimal with the trampoline, achieving a detection rate of 18.8%, with a low granularity of 16. Surprisingly, L5 is more effective without

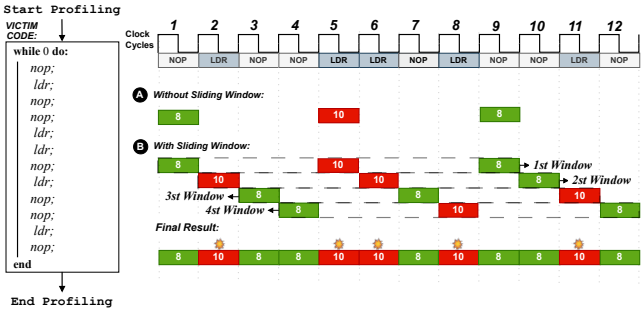

Figure 4: Illustration of a trace, with (A) and without (B) sliding window. In green no contention and red contention.

the trampoline, with a high granularity of 2 and detecting more points of contention, resulting in a detection rate of 16.9%. F4's low detection rate may appear problematic; however, it will be mitigated when the sliding window technique is applied. The direct drawback of a low detection rate is the increased number of windows required for a full trace, which in turn raises the total profiling time.

Time Reduction. We conducted a series of experiments to measure the profiling time of AES-CBC mode from TinyCrypt and RSA from wolfSSL on both the F4 and L5 MCUs, using a 32-bit timer. Table 3 presents the profiling times of the multi-run (MR) methodology, highlighted in bold, compared to the single-run (SR) methodology detailed in §2.3. The multi-run profiling times show significant reductions compared to the single-run on both platforms and cryptographic algorithms. For instance, on L5, the profiling time for AES was reduced by 600 times, and for RSA, approximately 300,000 times. These results hint at the suitability of the multi-run methodology to profile compute-intensive applications.

Real Victim Trace. Figures 5 presents the traces obtained from AES-CBC TinyCrypt (while RSA profiling was also done, discussion focuses on AES-CBC due to space constraints) for the L5 and F4. Each vertical line denotes the DMA latency at a specific clock cycle. The orange line represents the contention threshold. The traces were generated using two different plaintexts while keeping the cryptographic key constant. Profiling for the L5 was conducted without a trampoline, requiring 12 windows. Profiling for the F4 was conducted with a trampoline, requiring 4100 windows. Both profiles exhibit distinct differences in the traces for each plaintext, indicating a potential information leak. Being able to observe differences raises the likelihood of a cryptographic leakage. In AES-CBC, trace variations may expose S-Box lookups, XORs, and CBC chaining operations. Exploiting these trace variations is out-of-scope of this paper and deferred for future work.

5 DISCUSSION AND NEXT STEPS

Microarchitectural influences on the Profiling mechanisms. The target platform's architecture and the CPU's microarchitecture influence the results of multi-run profiling. The F4 achieves optimal results using a trampoline, while the L5 performs best without one. The trampoline is effective for boards where stack interference, bus arbitration, and DMA characteristics prevent hardware gadgets from observing contention. However, these findings are

	F4 (@168 MHz)				L5 (@110 MHz)			
	T	Gra	CP	Detect	T	Gra	CP	Detect
No Trampoline	424	3	10	3%	1075	2	124	16.9%
Trampoline	65	16	62	18.8%	22	81	17	2.3%
Exectuion Time	1010				1783			
Single-Run CP	329				732			

Table 2: Multi-run profiling metrics from tracing 1000 memory accesses, without sliding window. Trace in clock cycles (T), Granularity (Gra), Contention Points (CP), and percentage of multi-run CP relative to single-run CP (Detect).

Victim	F4 (@168 MHz)			L5 (@110 MHz)		
	SR	MR	Reduction	SR	MR	Reduction
AES (TinyCrypt)	16 s	**1.6 s**	10x	30 s	**50 ms**	600x
RSA (wolfSSL)	432 days	**6 s**	6220800x	14 years	**24 min**	306722x

Table 3: Profiling time of TinyCrypt AES and wolfSSL RSA using BUSted multi-run methodology (MR) in comparison with single-run methodology (SR), from §2.3.

preliminary. **Further research** is necessary to evaluate the performance of our multi-run methodology on MCUs with more complex microarchitectures (e.g., the Arm Cortex-M7).

Combining BUSTed Methodologies. We show that the single-run methodology is unsuitable for compute-intensive code. We proposed an interrupt-based multi-run BUSted methodology, which notably reduces profiling time but adds complexity to the spy software logic and reduces profiling accuracy. For simpler victims, the single-run approach is preferable due to its precision and manageable profiling time. However, for compute-intensive code, the multi-run methodology is a must. We consider single-run and multi-run methodologies not as competing approaches but as complementary ones. **We envision** a hybrid BUSted methodology where the multi-run performs a coarse-grained trace to identify potentially vulnerable pieces of code, and the single-run provides a detailed, clock cycle-accurate trace of a specific portion of the victim code.

BUSted Multi-run Exploitation Phase. This paper examines the Profiling phase of the BUSted attack. Traces from profiling TinyCrypt AES-CBC suggest that BUSted could exploit more compute-intensive code. The profiling phase reveals variations in traces from different plaintexts, indicating potential information leakage. However, it is unclear whether these variations result from secret-dependent or regular code. **Further research** is needed to evaluate the level of information leakage in the traces, and additional profiling is required for the multi-run Exploitation phase.

Hardware Gadgets and SGN complexity. The BUSted multi-run attack reduces the complexity at the hardware gadget level. Unlike the single-run variant, the multi-run does not need real-time side-channel data collection. In the multi-run approach, the secret remains constant, allowing the attacker to profile the same secret repeatedly. Consequently, compared to the BUSted single-run, the multi-run Exploitation phase becomes significantly simpler, relying solely on offline correlation of multiple traces obtained during profiling without the need for additional hardware. Supported by the evidence in §4.1, we argue that a comprehensive BUSted multi-run attack requires minimal additional hardware beyond what is

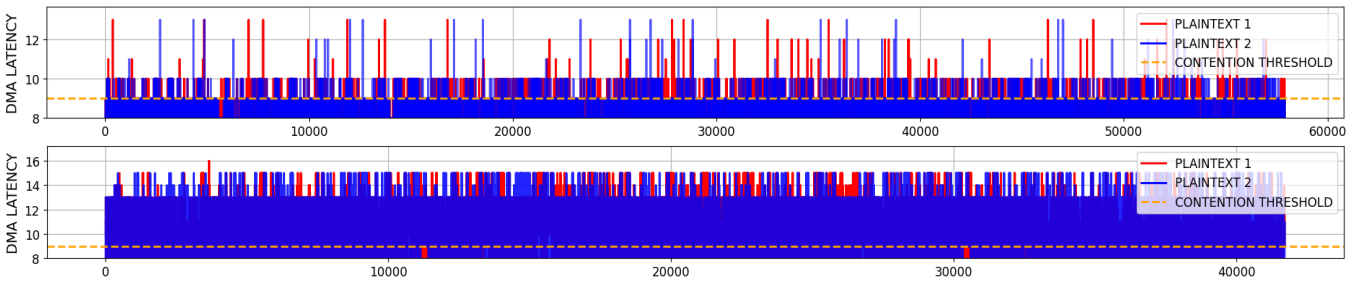

Figure 5: TinyCrypt AES-CBC trace on L5, no Trampoline 12 Windows, and F4, Trampoline 4100 Windows.

presented in this paper's Profiling phase. This **paves the way** for full attack automation, as the software is much easier to automate, and the hardware (SGNs) is far less complex.

6 RELATED WORK

Side-channel attacks have become powerful tools for attackers seeking to extract otherwise unavailable information. Despite extensive research efforts over the past decades to raise awareness of software-based side-channel attacks on APUs [1, 4, 5, 7–10, 14, 21], there is a notable shortage of reported software-based side-channel attacks on MCUs. The authors in [3, 6, 13] proposed powerful software-based side-channel attacks, but these focus on the power consumption of target devices rather than the timing differences inherent to the microarchitecture of MCUs. The only known software-based timing side-channel attacks on MCUs are Nemesis [18] and BUSted [16]. Nemesis exploits timing differences in the CPU interrupt logic of an MSP430 MCU. BUSted leverages timing differences in the bus interconnect arbitration logic to mount successful exploits on the Arm Cortex-M23/33 MCUs. Although BUSted introduced both single-run and multi-run attack variants, only the single-run approach has been proven and exploited, which is not suitable for cryptographic applications. Consequently, the BUSted multi-run variant remains to be thoroughly investigated.

7 CONCLUSION

In this paper, we investigate the limitations of the BUSted single-run variant when applied to compute-intensive code, such as cryptographic algorithms. We propose a new interrupt-based multi-run approach, which poses two new design challenges due to stack interference and low trace granularity. We introduced two new techniques to overcome these challenges: trampoline setup code and sliding window. Using the proposed approach, we successfully profiled an AES algorithm from TinyCrypt. This research takes the first steps toward developing a side-channel attack methodology capable of tampering with MCU-based cryptographic algorithms and compromising the security foundation of modern IoT devices.

ACKNOWLEDGMENTS

This work is supported by national funds, through the Operational Competitiveness and Internationalization Programme (COMPETE 2020) Project nº 179491; Funding Reference: SIFN-01-9999-FN-179491]; This work has been supported by FCT— *Fundação para a Ciência e Tecnologia* within the R&D Units Project Scope UIDB/00319/2020 and grant 2020.08729.BD.

REFERENCES

[1] Onur Aciicmez and Werner Schindler. 2007. A Major Vulnerability in RSA Implementations due to MicroArchitectural Analysis Threat. *Cryptology ePrint Archive.*
[2] Arm. 2023. Clarification of Timing Side Channel Attacks on TrustZone enabled Cortex-M based systems. https://developer.arm.com/documentation/ka005578/.
[3] Alessandro Barenghi et al. 2021. Exploring Cortex-M Microarchitectural Side Channel Information Leakage. *IEEE Access.*
[4] Claudio Canella et al. 2019. A Systematic Evaluation of Transient Execution Attacks and Defenses. In *Proc. of USENIX Security.*
[5] Daniel Gruss et al. 2015. Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches. In *Proc. of USENIX Security.*
[6] Dennis Gnad et al. 2019. Leaky Noise: New Side-Channel Attack Vectors in Mixed-Signal IoT Devices. In *IACR Transactions on Cryptographic Hardware and Embedded Systems.*
[7] Moritz Lipp et al. 2018. Meltdown: Reading Kernel Memory from User Space. In *Proc. of USENIX Security.*
[8] Maria Mushtaq et al. 2020. Winter is here! A decade of cache-based side-channel attacks, detection mitigation for RSA. *Information Systems.*
[9] Paul Kocher et al. 2019. Spectre Attacks: Exploiting Speculative Execution. In *Proc. of S&P.*
[10] Xiaoxuan Louet et al. 2021. A Survey of Microarchitectural Side-channel Vulnerabilities, Attacks, and Defenses in Cryptography. In *Proc. of ACM.*
[11] Intel. 2019. TinyCrypt Library. https://docs.zephyrproject.org/2.7.5/guides/crypto/tinycrypt.html.
[12] Cyber Risk Leaders. 2023. From Spectre/Meltdown to side channel attacks on microcontrollers. https://cyberriskleaders.com/from-spectre-meltdown-to-side-channel-attacks-on-microcontrollers/.
[13] C. O'Flynn and A. Dewar. 2019. On-Device Power Analysis Across Hardware Security Domains.: Stop Hitting Yourself.. In *IACR Transactions on Cryptographic Hardware and Embedded Systems.*
[14] Colin Percival. 2005. Cache missing for fun and profit. In *Proc. BSDCan.*
[15] The Register. 2023. Arm acknowledges side-channel attack but denies Cortex-M is crocked. https://www.theregister.com/2023/05/15/mcu_side_channel_attack/.
[16] C. Rodrigues, D. Oliveira, and S. Pinto. 2024. BUSted!!! Microarchitectural Side-Channel Attacks on the MCU Bus Interconnect. In *Proc. of IEEE S&P.*
[17] Jo Van Bulck, Frank Piessens, and Raoul Strackx. 2017. SGX-Step: A Practical Attack Framework for Precise Enclave Execution Control. In *Proc. of SysTEX.*
[18] Jo Van Bulck, Frank Piessens, and Raoul Strackx. 2018. Nemesis: Studying Microarchitectural Timing Leaks in Rudimentary CPU Interrupt Logic. In *Proc. of ACM CCS.*
[19] WolfSSL. 2023. BUSted: Side-channel attacks to TrustZone-M separation. https://www.wolfssl.com/busted-side-channel-attacks-to-trustzone-m-separation/.
[20] WolfSSL. 2024. Algorithms - RSA. https://www.wolfssl.com/documentation/manuals/wolfssl/group__RSA.html.
[21] Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack. In *Proc. of USENIX Security.*