Shielding Performance Monitor Counters: A double edged weapon for safety and security

Shielding Performance Monitor Counters: a double edged weapon for safety and security

Alberto Carelli, Alessandro Vallero and Stefano Di Carlo
Politecnico di Torino, Department of Control and Computer Engineering,
{alberto.carelli | alessandro.vallero | stefano.dicarlo}@polito.it

***Abstract*—Recent years have witnessed the growth of the adoption of Cyber-Physical Systems (CPSs) in many sectors such as automotive, aerospace, civil infrastructures and healthcare. Several CPS applications include critical scenarios, where a failure of the system can lead to catastrophic consequences. Therefore, anomalies due to failure or malicious attacks must be timely detected. This paper focuses on two relevant aspects of the design of a CPS: safety and security. In particular, it studies how performance monitor counters (PMCs) available in modern microprocessors can be from the one hand a valuable tool to enhance the safety of a system and, on the other hand, a security backdoor. Starting from the example of a PMC based safety mechanism, the paper shows the implementation of a possible attack and eventually proposes a strategy to mitigate the effectiveness of the attack while preserving the safeness of the system.**

***Index Terms*—safety, security, cyber physical systems, performance counters.**

I. Introduction

The increasing adoption of Cyber-Physical Systems (CPSs) employing IoT devices is a matter-of-fact. A CPS is a system controlling a physical process. A CPS integrates processing units, sensors and actuators enabling the interaction of the computing infrastructure with the physical world. All devices of a CPS are interconnected in order to create a network enabling the different nodes to exchange information. The main task of such devices is to acquire knowledge on a physical process.

The application of CPSs is becoming pervasive and will drive innovation in many sectors such as aerospace, automotive, chemical process, civil infrastructures, energy, healthcare, manufacturing and transportation [1], [2]. In this context, CPSs are becoming a significant and mandatory element of several critical infrastructures [3]. Critical infrastructures constitute the technological backbone of our society. However, the consequences of a misbehavior of such complex systems can lead to catastrophic consequences. The misbehavior can be the result of a failure of one of their components, or an intentional attempt to corrupt their behavior.

In this context, both safety and security aim at the same goal. Safety aims at avoiding hazards due to accidental failures, while security focuses on protecting the system from intentional attacks [4]. The boundary between safety and security is becoming thinner and there is a recognized request to jointly strengthen both of them. This is exacerbated by the fact that security solutions may have a negative effect on the safety properties of the system and vice versa [5].

Given this premise, this paper focuses on the safety and security implications introduced by the availability of different Performance Monitor Counters (PMCs) in modern microprocessors employed in CPSs. PMCs are an effective instrument to detect timing violations in multiprocessor systems and other types of physical failures. However, at the same time, they have been exploited to perform different classes of attacks (see Section II for related work on the use of PMCs in the safety and security domain).

In particular, this paper analyzes the application of the safety technique proposed in [6] in the context of a complex CPS. This technique exploits PMCs to monitor the duration of a process in order to prevent deadline violations during the execution of different tasks. However, the need to access the PMCs introduces a serious security vulnerability. The same PMCs used to guarantee safety of the system can be used to implement the attack described by Bonneau in [7]. This attack exploits the PMCs to discover the secret key of an Advanced Encryption Standard (AES) cipher potentially violating the confidentiality of a system using this encryption algorithm. Eventually, this paper proposes a strategy to alter the readings of the PMCs at the user level in order to mitigate the side-channel attack, while preserving the effectiveness of the considered safety mechanism.

In our experiments, we evaluated the impact of the proposed security mitigation technique on the safety and security of a CPS executing 7 MiBench benchmarks [8] and the AES encryption, victim of the attack. The ultimate goal of our experiments is to show how the proposed technique enables to maintain the safety level of the system while mitigating the effect of the considered attack.

II. Background

PMCs are special registers available on most microprocessor architectures. They are used to monitor events such as branch predictions, cache hits/misses, process timing, etc.. The set of events that can be monitored depends on the target microprocessor. The privileges required to access these counters depend on both the processor architecture and the operating system (OS).

PMCs are commonly used for detailed run-time profiling of application/OS software. An accurate software profiling enables performance improvement or prediction of performance degradation [9]. PMCs have been employed in [10] to monitor the control flow integrity of software applications by analyzing

branch instructions, thus detecting deviations from the correct control flow. In [6] PMCs are used to detect faults causing deadline violations in multicore systems. The authors of [11] focus the estimation of WCET through PMCs for safety-critical applications. In [12], WCET-aware PMUs are proposed for safety certification in the automotive domain.

Considering the security domain PMCs were adopted to detect firmware modifications [13]. However, different threats are linked with PMCs. Indeed, each monitored event represents a possible source of side-channel information that can be exploited by a malicious attacker. Most related works in this domain, focuses on the cache behavior during the encryption with the AES algorithm. The author of [14] is able to recover the complete AES key remotely exploiting timing information related to cache accesses. The timing attack performed in [7] requires a reduced number of samples to recover the AES key when applied to Intel architectures. PMCs have been employed as source of side-channel information also to attack encryption algorithms on AMD platforms in [15]. Side-channel attacks are possible also for asymmetric key cryptography, as reported in [16]. The attack, carried out on Intel platforms, targets a 1024 bit key of RSA and exploits branch-miss events. Proper defense measures can be taken if the attack is detected. The authors of [17] propose a generic detection mechanism, using a pre-trained classifier, able to deal with a variety of microarchitectural side-channel attacks, including also cache-based attacks.

III. CPS Architecture

A. CPS and node architecture

Fig. 1 shows the general CPS architecture considered in this paper. It is composed of a master node, called *monitor*, linked to a set of slave nodes. The monitor is the central unit controlling the network. It constantly communicates with the slave nodes, exchanging data with them. The monitor is responsible for checking the responsiveness of each slave node. Moreover, it decides the actions the CPS must carry out when interacting with the physical world. This is accomplished by sending commands to the actuators of the slave nodes based one the received sensor data.

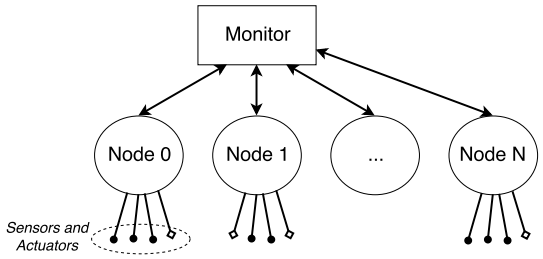

Fig. 1: The CPS architecture scheme

Fig. 2 shows the conceptual architecture of the slave *nodes*. They are computational units able to communicate with the monitor. Each node is directly connected to a set of *sensors* and *actuators*. The sensors read the raw data of the cyber-physical system, while the actuators execute the commands received from the node. Each slave node is equipped with an OS in charge of providing a set of services required to accomplish different real-time tasks. Each node has a certain number of tasks to complete, which is an intrinsic property of the node itself. The number of tasks depends on the available sensors and actuators and on the function the node has to satisfy. The data exchanged with the monitor is encrypted with a symmetric key by an appropriate service module integrated in the OS. In general, any task running at the application level can request the OS services.

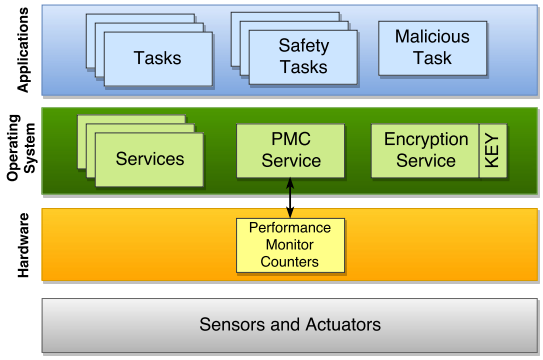

Fig. 2: The node architecture

Different safety mechanisms to control the correct operation of the node can be implemented as additional safety tasks at the application level, potentially exploiting PMCs to carry out their activity. The next section will introduce the specific safety mechanism considered in this paper.

B. Safety task

The safety task considered in this paper is based on the process monitoring technique presented by Esposito et al. in [6]. The goal of the implemented technique is the online detection of faults potentially able to cause deadline misses of the tasks executed by the node. Overall, the considered safety technique consists of two phases:

1) *Off-line phase*: all critical tasks executed on the node are profiled off-line in order to generate their execution time profile. This is accomplished by recording the PMCs over several concurrent executions of the different tasks. From the analysis of the application profiles a set of thresholds for the considered PMCs is generated. As an outcome of this phase, a set of thresholds for the PMCs are computed.
2) *On-line phase:* all critical tasks are monitored through the values of their PMCs, based on the thresholds computed during the off-line phase. Every time a task violates one of the thresholds a proper warning or error is issued to highlight a potential hazard. This may in turn trigger the activation of a recovery mechanism.

Further details about the implementation of this safety mechanisms will be provided in section IV where mitigation techniques against PMCs-based security attacks will be introduced.

C. Attack model

Securing the CPS architecture presented in Fig. 1 is an important task. This paper focuses on an attacker interested in recovering the encryption key of a node to carry out malicious actions. We suppose the attacker is not interested in denial-of-service attacks, because they disrupt the offered services and prevent the control of the CPS. A successful attack on a node may spread the infection to every node, thus compromising the whole system. In the case all nodes share the same secret key, the whole system would be immediately compromised when a single node is compromised. When the secret key is different for every node, the same malicious task could target another node and the attack could be repeated until all nodes composing the CPS are under control of the attacker.

Looking at the architecture of the node reported in Fig. 2, we assume that enough effort has been carried out to secure the hardware and software architecture (OS and application level tasks) of the node. This includes securing the secret key and the encryption/decryption service. Nevertheless, we assume that the attacker may exploit user level vulnerabilities to inject a malicious task (e.g., a virus or a malware) within a node. The attack could be undertaken on a specific node because the attacker could have gained physical access to it, or because that specific node offers unique vulnerabilities. The malicious task is a user application that can exploit the computational resources of the node as well as the services offered by the OS of the node, thus it can probe the PMCs and trigger the encryption process.

The possibility to access the PMCs makes the system vulnerable to timing attacks, a category of side-channel attacks that exploit time measurements as source of side-channel information [14]. PMCs are the force point of safety mechanisms, although they represent a weak point of the system from a security perspective. They can therefore be considered as a double edged weapon.

This paper focuses on the side-channel attack presented by Bonneau et al. in [7], which targets the AES encryption algorithm. The attack, exploits timing information related to the memory access in the final round of the AES encryption. More specifically, the encryption time is influenced by the data locality of the final round S-box in the cache memory. In [7] the attack is implemented looking at the L1 data or L2 cache access profile. The attacker chooses a random initial key. It then runs several iterations evolving the key, performing and collecting a large number of encryption processes. The key evolves according to the encryption time and to the ciphertext. Each encryption process is defined as a sample. A large number of samples is collected in order to give the evolution a statistical meaning. The process ends when the correct key is found.

The possibility to access the PMCs by the malicious task, that includes timing and cache access information, opens a path to properly implement this attack in the node architecture presented in Fig. 2.

IV. Attack mitigation

The success of the Bonneau attack introduced in section III-C depends on the PMCs readings carried out by the malicious task. A possible mitigation strategy for this attack consists of corrupting these readings of a proper quantity in order to make the attack fail. However, this corruption may jeopardize the capability of the safety tasks that rely on the PMCs to detect anomalies due to system's failures.

This section proposes a methodology to corrupt the PMCs in order to mitigate the Bonneau attack, while keeping under control the impact of the corruption on the detection capability of the safety task introduced in section III-B.

To explain how the PMCs can be corrupted, let us start from the description of how the considered safety mechanism can be implemented in the proposed scenario.

During the offline phase (see section III-B), the user tasks are profiled and information about their execution time is collected looking at the PMCs (e.g., clock cycle counter). The values measured by a PMC for the different processes can be considered as a random variable X, characterized by an empirical Probability Density Function (PDF), $f_X(x)$, and an empirical Cumulative Distribution Function (CDF), $F_X(x)$. This assumption is justified by the intrinsic non-determinism of the computing platform. In fact, modern microprocessors implement out-of-order execution models in which instructions are executed in a non-deterministic order. Moreover, the memory hierarchy featuring different levels of cache memories determines non-deterministic data access profiles. Finally bus arbitration, memory controllers and other architectural components are characterized by non-deterministic behaviors.

Based on the collected profiles, each task can be associated to three operating areas reflecting the state of the system: *safe area*, *critical area* and *warning area*.

Two thresholds are defined to separate the aforementioned operating areas: T_W and T_C. These thresholds are chosen starting from two confidence levels C_W and C_C. Confidence levels can be chosen arbitrarily during the design phase. Strict confidence levels can increase the number of false positives and the performance overhead due to a larger number of recovery operations. Wide levels may not detect all failures of the system. In details, T_W and T_C can be computed by solving the following inequalities looking at the collected profiles:

$$P(X > T_W) < C_W \Rightarrow F_X(T_W) > 1 - C_W \tag{1}$$

$$P(X > T_C) < C_C \Rightarrow F_X(T_C) > 1 - C_C \tag{2}$$

During the on-line phase (see section III-B), the safety task monitors the execution time of every task in order to determine in which area it is located and if a recovery action is required. The decision process relies on the value of the PMCs and on the values of the thresholds identified during the off-line phase. The system state is then classified as follows:

- *safe*: if the value of the PMCs is below the warning threshold. In this case the system is considered safe and it keeps working normally;

- *critical*: if the value of the PMCs is above the critical threshold. In this case the system is considered unsafe and a proper recovery actions must be issued;
- *warning*: if the value of the PMCs is between the warning and the critical threshold. In this case the system is considered safe and it keeps working normally. However, if the system is classified as *warning* α consecutive times, it is then considered as *critical*.

The choice of α is related to the probability that the system is in a *safe* state after α consecutive *warning* classifications denoted as $P(FP_\alpha)$. This indicates a false positive event. $P(FP_\alpha)$ is obtained during the process profiling of the off-line phase. We report from [6] the formula to compute α:

$$\alpha = \frac{ln(1 - P(FP_\alpha))}{ln(F(T_C) - F(T_W))} \tag{3}$$

Starting from this premise, we propose to modify the PMC service implemented at the OS level of Fig. 2 in order to expose a corrupted value of each PMC defined as:

$$\overline{PMC} = PMC + c \tag{4}$$

where $\overline{PMC}$ is the corrupted PMC reading, PMC is the correct PMC reading, and c is the corruption level. It is worth to recall here that, in the proposed architecture, the PMC service is considered secure (see section III-C) and represents the only user access point to the PMCs.

The corruption level c is computed as a uniformly distributed random variable defined as:

$$c = U(0, s \times (\mu - T_W)/2) \tag{5}$$

where μ is the average value of the measured PMC of a user process profiled during the off-line phase and s is a scaling factor for the middle point between μ and T_W. As a result, every time a performance counter related to a user process is read, the corruption level changes. This consequently randomizes the differences between the PMC readings, which are at the base of the implementation of the Bonneau attack described in section III-C.

The corruption level is tailored on the μ and T_W of each process and it is bounded by the scaling factor s. This limits the increase of false positives and the impact on the performance of the system. The adoption of s plays a key role in the proposed methodology allowing to trade-off security and safety as will be described in the next section presenting experimental results. Simpler value alterations are possible, however they might not adequately counteract the discussed attack.

V. Experimental Results

To show the proposed approach at work, we evaluated as a case study a slave node running 7 tasks based on different MiBench benchmarks [8]: *cjpeg*, *djpeg*, *fft*, *qsort*, *susan smoothing*, *susan edges* and *susan corners*. The system runs on top of a Linux OS, for which a PMC service and an AES encryption service (which is the victim of the Bonneau attack) were implemented. The hardware employed for the analysis is an Intel Core i7 CPU Q720 running at 1.6GHz. The PMC considered for the experiments is the clock cycle counter, in order to detect longer-than-expected execution time.

Fig. 3 reports the CDFs of the considered PMC for the different tasks (off-line profiling). Results were obtained collecting 100K samples for each profiled task in order to provide statistical meaning. For every profiled process the average value μ and the two thresholds, T_W and T_C, are reported. The confidence levels chosen for each process are $C_C = 0.6\%$, to determine the T_C, and $C_W = 5\%$ to determine T_W. Each process is characterized by a different CDF. Therefore, the distance among μ, T_W and T_C changes accordingly. Based on this setup, we expect different behaviors of the safety task for each process and a different behavior when the readings of the PMC are corrupted.

According to section IV, the classification of the state of each task is influenced by two main parameters: s and α. To evaluate the impact of these two factors, we started by comparing, for a constant $\alpha = 3$, the behavior of the safety task with our without PMC corruption for s between 0.2 and 0.8. Since the PMC corruption is randomly generated, its effect for each task has been evaluated averaging results of 1K repetitions on the collected 100K samples.

Fig. 4 reports the percentage of task samples classified as *safe*. As expected, all tasks follow a similar trend: increasing values of s underestimate the percentage of *safe* classifications. However, the degradation is not the same from a quantitative point of view. For instance, the degradation of *fft* and *cjpeg* is very different. Both tasks have a percentage of safe samples very close to 99% in the case without corruption (red bars). However, *fft* seems to be very resilient to corruption, loosing less than 0.04 percentile points (p.p.) in the case $s = 0.8$, while *cjpeg* decreases of more than 1.5 p.p..

A better insight to analyze the causes of this behavior is provided in Fig. 5. The percentage of samples classified as error is reported as a stacked bar graph, considering the samples classified directly in the *critical* state (Err) and the ones first classified as *warning* and later evaluated as *critical* (WrnToErr). On average, the contribution to the total amount OF samples in *critical* state is distributed equally between *Err* and *WrnToErr*, with a slight predominance of *Err*. This is expected since both parameters grow proportionally with the increase of the corruption factor s. However, in some benchmarks *Err* grows faster than *WrnToErr*. For example, they increase by 0.8 p.p. for *Err* and 0.2 p.p. for *WrnToErr* in *djpeg*. The opposite happens in other cases, e.g., in *cjpeg*, 0.3 p.p. for *Err* and 1.2 p.p. for *WrnToErr*. Such different behaviors are attributed to the trend of the CDF in the region close to the left of the two thresholds: T_W and T_C. Looking at the CDF of *djpeg* near T_C, the curve is not flat. This means that there are several samples in this region. When the PMC is corrupted, the CDF is shifted to the right by a small quantity, while the thresholds do not move. As a result, there is a moderate increase of *Err* (0.8 p.p. in *djpeg*). For other tasks whose CDF is almost flat near T_C, the increase of *Err* is very small. This is the case of *edges* (0.03 p.p.).

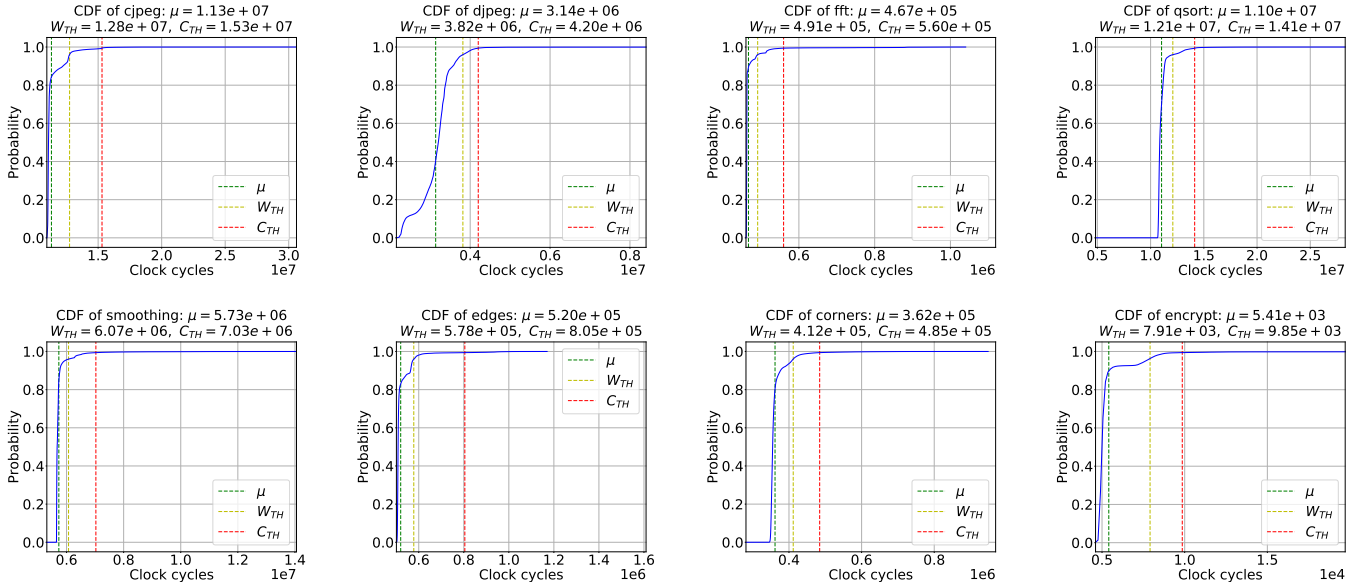

Fig. 3: Cumulative density function of the task execution timing distributions collected during the off-line phase.

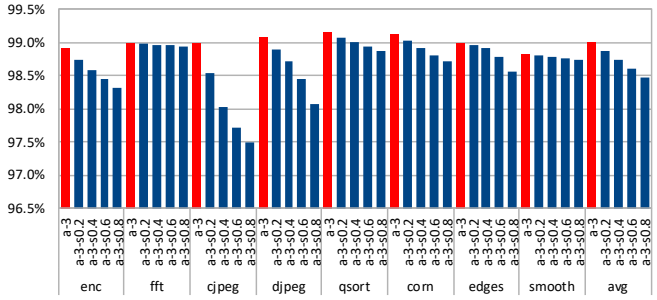

Fig. 4: Percentage of safe classifications without corruption (red bars) and with corruption (blue bars).

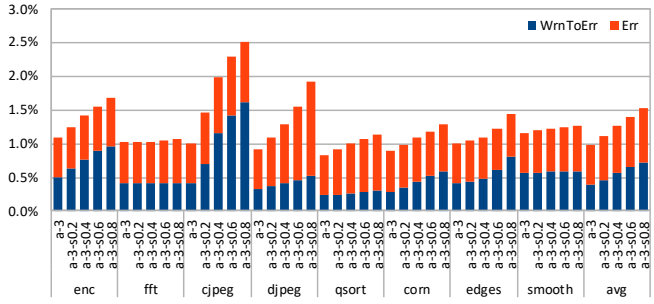

Fig. 5: Percentage of samples classified as *critical*.

The distribution of those samples that were not directly classified in the *safe* state is shown in Fig. 6. Samples are divided into three categories: *Err*, *WrnToErr* and the samples classified as *warning* at first and later evaluated as *safe* (WrnToOk). Corrupting the PMCs translates into a right shift of their CDFs increasing *Err* and *warning*. In general, since the left side of T_W is more populated than the one of T_C, the increase of *warning* is higher than the one of *Err*. However, a small part of *warning* is later evaluated as *critical*. This is deduced by *edges*, where the number of *warning* raises by 4 p.p. when the corruption factor is equal to 0.8, while *WrntoErr* grows by just 0.4 p.p.. This is a good aspect of the adopted safety mechanism where the effect of α mitigates the negative effect of the PMC corruption.

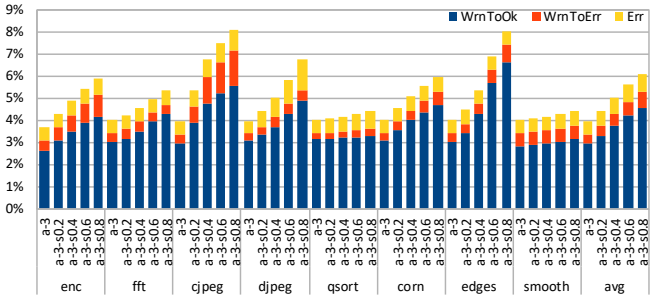

Fig. 6: Percentage of samples not directly classified as *safe*.

The proposed methodology increases the number of predicted *critical* samples. This translates into a growth of recovery actions to be performed even if not required. Since recovery takes time, a performance degradation is introduced. Fig. 7 quantifies the percentage of recovery actions due to false positives with respect to the number of required recovery actions when the PMCs are not corrupted. As expected, the degradation rises as the corruption is higher. The *cjpeg* has the highest degradation (44.5% with $s = 0.2$ and 149.5% with $s = 0.8$), the degradation is almost negligible for *fft* (0.7% with $s = 0.2$ and 4.2% with $s = 0.8$), while in the average case the degradation is in between 12.9% and 55.8%.

Finally, experiments were repeated keeping the corruption

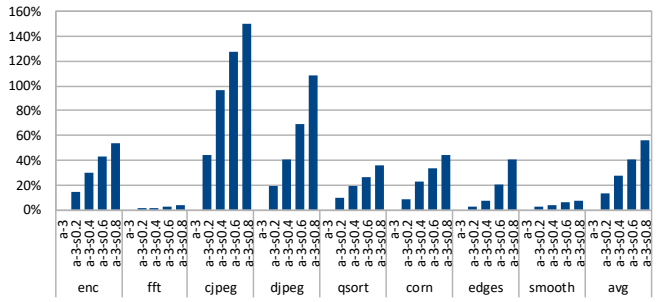

Fig. 7: Ratio between recovery actions due to false positives and recovery actions detected without corrupting the PMCs.

fixed ($s = 0.5$) and changing the value of α. The percentage of recovery actions due to false positives with respect to the number of required recovery action when the PMCs are not corrupted is reported in Fig. 8. A common trend cannot be identified. Therefore, the impact of α on the performance degradation introduced by the proposed methodology must be computed individually for every benchmark. However, the maximum difference we measured is lower than 20.3p.p. for *cjpeg*, and in the average case is 4.6p.p..

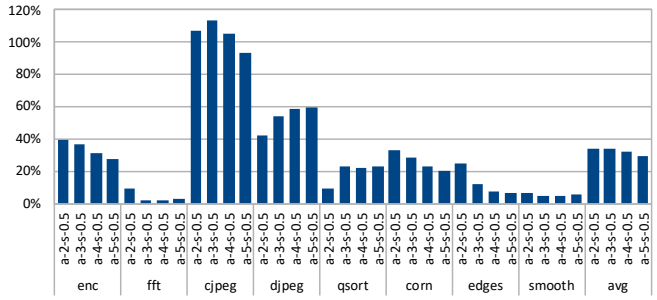

Fig. 8: Ratio between recovery actions due to false positives ($s = 0.5$ and changing α) and in the case without corruption.

From a security perspective, we analyzed the ability of performing the attack both on the unprotected and on the protected version of the node. The malicious task was consistently able to perform the attack using about 65M samples. Given the computational complexity required to perform the attack that limits the number of possible repetitions, we simulated the protected system with the lowest PMC corruption $s = 0.2$ and $s = 0.4$, representing the best conditions for the attacker (lower corruption values). In the first case the attack was able to recover the key in the protected application with 163M samples, while with 204M samples in the second case, thus proving the effectiveness of the PMCs shielding technique.

VI. Conclusion

This paper studied the interplay of two challenging aspects of the design of a CPS: safety and security. It focused on the role that the PMCs have when implementing mechanisms able to enhance the safety of the system and, on the other hand, the risks they introduce when looking at the security of the system. Starting from the example of a PMC based safety mechanism, and from the implementation of a security attack, the paper proposed an attack mitigation strategy. Experimental results showed the effectiveness of the proposed technique to increase the complexity of the attack while preserving the safeness of the system.

References

[1] L. Wang and X. V. Wang, *Cloud-Based Cyber-Physical Systems in Manufacturing*. Springer, 2018.

[2] C.-R. Rad, O. Hancu, I.-A. Takacs, and G. Olteanu, "Smart monitoring of potato crop: a cyber-physical system architecture model in the field of precision agriculture," *Agriculture and Agricultural Science Procedia*, vol. 6, pp. 73–79, 2015.

[3] J. Ding, Y. Atif, S. F. Andler, B. Lindström, and M. Jeusfeld, "Cps-based threat modeling for critical infrastructure protection," *SIGMETRICS Perform. Eval. Rev.*, vol. 45, no. 2, pp. 129–132, Oct. 2017.

[4] G. Sabaliauskaite and A. P. Mathur, "Aligning cyber-physical system safety and security," in *Complex Systems Design & Management Asia*. Springer, 2015, pp. 41–53.

[5] L. Piètre-Cambacédès and M. Bouissou, "Modeling safety and security interdependencies with bdmp (boolean logic driven markov processes)," in *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*. IEEE, 2010, pp. 2852–2861.

[6] S. Esposito, M. Violante, M. Sozzi, M. Terrone, and M. Traversone, "A novel method for online detection of faults affecting execution-time in multicore-based systems," *ACM Trans. Embed. Comput. Syst.*, vol. 16, no. 4, pp. 94:1–94:19, May 2017.

[7] J. Bonneau and I. Mironov, "Cache-collision timing attacks against aes," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2006, pp. 201–215.

[8] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No.01EX538)*, Dec 2001, pp. 3–14.

[9] A. Kandalintsev, R. L. Cigno, D. Kliazovich, and P. Bouvry, "Profiling cloud applications with hardware performance counters," in *The International Conference on Information Networking 2014 (ICOIN2014)*, Feb 2014, pp. 52–57.

[10] Y. Xia, Y. Liu, H. Chen, and B. Zang, "Cfimon: Detecting violation of control flow integrity using performance counters," in *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*. IEEE, 2012, pp. 1–12.

[11] J. Nowotsch, M. Paulitsch, D. Bhler, H. Theiling, S. Wegener, and M. Schmidt, "Multi-core interference-sensitive wcet analysis leveraging runtime resource capacity enforcement," in *2014 26th Euromicro Conference on Real-Time Systems*, July 2014, pp. 109–118.

[12] E. Mezzetti, L. Kosmidis, J. Abella, and F. J. Cazorla, "High-integrity performance monitoring units in automotive chips for reliable timing v and v," *IEEE Micro*, vol. 38, no. 1, pp. 56–65, January 2018.

[13] X. Wang, C. Konstantinou, M. Maniatakos, and R. Karri, "Confirm: Detecting firmware modifications in embedded systems using hardware performance counters," in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov 2015, pp. 544–551.

[14] D. J. Bernstein, "Cache-timing attacks on aes," [Online] https://cr.yp.to/antiforgery/cachetiming-20050414.pdf, 2005.

[15] L. Uhsadel, A. Georges, and I. Verbauwhede, "Exploiting hardware performance counters," in *2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*, Aug 2008, pp. 59–67.

[16] S. Bhattacharya and D. Mukhopadhyay, "Who watches the watchmen?: Utilizing performance monitors for compromising keys of rsa on intel platforms," in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2015, pp. 248–266.

[17] M. Alam, S. Bhattacharya, D. Mukhopadhyay, and S. Bhattacharya, "Performance counters to rescue: A machine learning based safeguard against micro-architectural side-channel-attacks."