Towards Malicious Exploitation of Energy Management Mechanisms

Safouane NOUBIR, Maria MENDEZ REAL, Sébastien PILLEMENT
Institut d'Éléctronique et de Télécommunications de Rennes (IETR), UMR CNRS 6164, Université de Nantes
Email: safouane.noubir@etu.univ-nantes.fr

***Abstract*—Architectures are becoming more and more complex to keep up with the increase of algorithmic complexity. To fully exploit those architectures, dynamic resources managers are required. The goal of dynamic managers is either to optimize the resource usage (e.g. cores, memory) or to reduce energy consumption under performance constraints. However, performance optimization being their main goal, they have not been designed to be secure and present vulnerabilities. Recently, it has been proven that energy managers can be exploited to cause faults within a processor allowing to steal information from a user device. However, this exploitation is not often possible in current commercial devices. In this work, we show current security vulnerabilities through another type of malicious usage of energy management, experimentation shows that it is possible to remotely lock out a device, denying access to all services and data, requiring for example the user to pay a ransom to unlock it. The main target of this exploit are embedded systems and we demonstrate this work by its implementation on two different commercial ARM-based devices.**

I. Introduction

In the embedded system industry, energy management has become a necessity to optimize system performance and reduce energy consumption, improve system portability and increase battery life.

An efficient energy manager is composed of a software algorithm accompanied by a hardware counterpart. The software part is generally a decision-making algorithm based on the CPU load while the hardware part is responsible for applying the decisions. The most used energy management mechanism is Dynamic Voltage & Frequency Scaling (DVFS) [1]. This mechanism allows to dynamically control the voltage and frequency to reduce energy consumption by controlling the processing speed. However, the voltage and frequency are strongly linked to the processor stability and by maliciously modifying them it has been proven that it is possible to inject faults into the system without any physical access [2]. Fault injection is a well known attack that was first introduced in 1996. By injecting faults during specific instructions of an encryption algorithm it is possible to use Differential Faults Analysis (DFA) to dramatically decrease the possible number of keys to test. Using the built-in DVFS to force the device into an unstable operation mode can result into injecting faults into the device operation. However, it does not provide the required precision to use DFA. In this work, we explore malicious usages of DVFS and we show that if stealing secret information exploiting energy management mechanisms is hardly possible in today's commercial devices, it is still possible for an attacker to remotely lock out a device and possibly ask for a ransom to make the device services and data accessible again.

This work is proven through its implementation on two current commercial ARM-based devices. First, this work is implemented in a Samsung chip, using Exynos 5422 architecture [3] based on ARM v7 big.LITTLE [4] architecture. This chip is widely used in the smartphone industry. Second, this work is also implemented on a very recent Huawei chip (from 2018); Kirin 960 System-on-Chip (SoC) using ARM v8 technology. This later is a more sophisticated chip which uses a separate co-processor to dynamically use DVFS. Our implementations show the feasibility of the proposed technique on both systems. The main contributions of this work are:

- The exploration of the feasibility of a malicious manipulation of DVFS in current commercial ARM-based devices for stealing secret data and the highlighting of different true challenges.
- The introduction of a malicious exploitation of energy management mechanisms able to lock a device making services and data inaccessible.
- The implementation of this technique onto two different recent and widely used commercial devices.

The remainder of this paper is as follows: In Section II we provide background related to our work by explaining the importance and principle of DVFS mechanisms, as well as previously presented work on possible malicious usage of DVFS. In Section III we first elaborate the threat model, the considered scenario and targeted system are presented and the conditions and assumptions of this work are explained. Next, in Section IV the main steps to exploit the technique presented in this work until the device locking are presented. Section V presents the implementation of the proposed technique on two different current commercial ARM-based devices as well as the obtained results. Section VI discusses the limitations of this work and gives some leads for possible countermeasures. Finally, Section VII concludes this paper.

II. Background and related work

A. Dynamic Voltage & Frequency Scaling

Dynamic Voltage & Frequency Scaling was first introduced in 1994 [1] and today it is one of the most used energy saving mechanisms in embedded systems such as in the smartphone industry. DVFS is based on the ability to control frequency

978-3-9819263-4-7/DATE20/©2020 EDAA

and voltage of a certain core or of a cluster in a SoC in order to dynamically trade processing speed for energy according to the system requirements. This mechanism encompasses a hardware component to control voltage and frequency and an energy aware software.

The hardware component, as shown in Fig.1, includes at least one voltage regulator and a frequency generator. Voltage regulators are part of the Power Management Integrated Circuit (PMIC). In ARM big.LITTLE architecture, for instance, the PMIC uses the same voltage for all the cores in a cluster. However, it is rarer but possible to find some devices using individual regulators for each core (e.g., Qualcom implementation of Snapdragon 800). The frequency is controlled by the Phase Locked Loop (PLL) circuit built in the SoC.

The software side, as shown in Fig.2, is composed of two layers. The first one is the energy aware application, which chooses the right voltage and frequency couple (also called operating point) depending on the processor load. The second layer is composed of the kernel module and the driver responsible for communicating new values of the operating point to the hardware. Most of the manufacturers use their own drivers with the PMIC, while the frequency is mostly controlled by software like *cpufreq* often available in Linux-based Operating Systems (OS). Cpufreq can also control the voltage through the vendor driver.

In most of the cases, a device equipped with DVFS has predefined operating points of frequency and voltage where the vendor guarantees the processor works in perfect and stable conditions. The role of cpufreq is to switch between those operating points depending on the CPU usage. However, the user can develop its own drivers in order to directly control the regulators either by writing on the register or by using kernel Application Programming Interfaces (API). By maliciously manipulating the voltage and frequency levels, it is possible to force the processor into unstable operating conditions that can lead to the introduction of faults in its operation.

B. Security Vulnerabilities of Energy Management Mechanisms

In a sequential circuit, when selecting a frequency level, the clock period must be greater than the critical path; the time for a flip-flop to lock on, plus the time for the data to stabilize. Failing to respect this timing constraint might result in input and output flip-flop desynchronization that can cause the introduction of faults in the CPU operations.

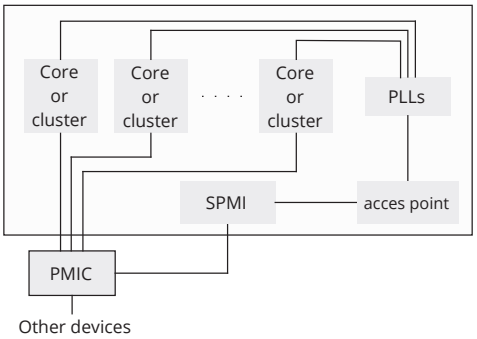

Fig. 1: Implementation of the DVFS mechanisms hardware

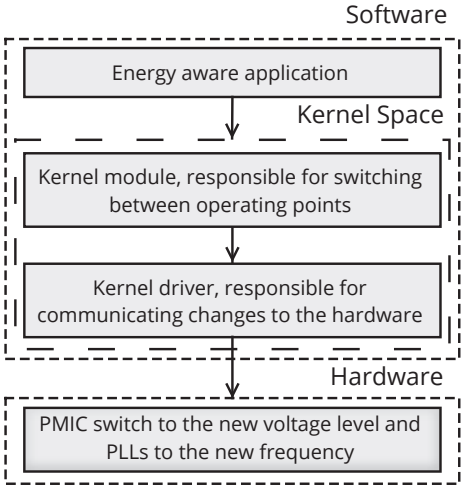

Fig. 2: Implementation of the DVFS mechanisms software

In an integrated circuit, the voltage has a direct influence to the critical path. By decreasing the supply voltage, the critical path increases. By simultaneously increasing the frequency, it is then possible to break the previous timing condition forcing the system into an unstable mode.

Previously, in [5] this vulnerability has been exploited to inject faults into a processor core. By injecting faults during an AES encryption, authors were able to use DFA to reduce the number of possible encryption keys until it was possible to use brute-force to find the correct secret key. Moreover, this technique was also used to corrupt the execution of an RSA signature verification algorithm and force the verification to allow the execution of a malicious application within an ARM TrustZone. This kind of faults injection does not require any kind of physical access to the system as DVFS can be remotely manipulated. However, compared to classical fault injection, one of the main challenges of this technique is the necessity for very high fault injection precision. For instance, a fault must be injected within 65000 instructions in an algorithm lasting millions of instructions. Furthermore, accessing the voltage and frequency regulators requires for the attacker to have root privileges. Finally, as rooted smartphones are the main target of the work in [5], a second important challenge is today's devices safetynet API [6]. This API verifies the integrity of the system with a distant server and informs important applications if any modification has been made (e.g., if the smartphone has been rooted). Consequently, recent rooted smartphones would not be able to execute critical applications (e.g. Google services, bank applications, ...).

Same researchers started to work on possible leads to prevent malicious usage of energy managers. In 2019, they proposed a machine learning algorithm to detect malicious usage of the DVFS [7]. Moreover, a different work proposed a chip called Fame [8], in order to detect and mitigate

hardware faults. Both solutions are interesting but require to add an integrated circuit to the SoC. These and other possible countermeasures are further discussed in Section VI. In this work we target very recent commercial devices, that do not encompass any additional hardware for fault injection mitigation.

A different work [9] presents a different malicious usage of DVFS mechanism. The main goal of this usage is to secretly transfer information from a task executing in the secure world on an ARM TrustZone (or from a non-secure third-party IP), to a second malicious task executing in the normal world by using the DVFS mechanism as a hidden channel of communication. The main idea is the encoding of the information to be sent into different values of voltage and frequency. On the reception part, the second malicious task decodes this information. This technique requires however an already corrupted task executing within the secure world of a TrustZone processor.

In related works, when maliciously manipulating energy management mechanisms, it is either necessary to have high precision to target specific instruction in the victim task when introducing faults, or to rely on a malicious task that is already executing in the secure world of the Trusted Execution Environment (TEE). In this paper, we explore the possibilities of a malicious manipulation of DVFS mechanisms on recent commercial devices under realistic conditions. Contrary to these previous works, our goal is not to steal secret information on the device but to take control and make every device services and data inaccessible.

III. Threat model

A. Scenario

The main goal of this work is to explore the security vulnerabilities of today's commercial devices when DVFS mechanisms are maliciously manipulated. Our work shows that while stealing secret information with this technique is unlikely in today's devices, making a device inaccessible (services and data) is still possible in current sophisticated devices. In a possible exploitation scenario considering a smartphone for instance, the attacker is able to remotely lock the device and possibly ask the owner for a ransom. The device owner (victim) can then whether wipe out all the data and restart the device (loosing all data) or to pay a ransom to gain access to the device back. It is worth noting that in this work, the attacker locks out the device by generating a remote hardware fault and not by encrypting any data. This makes a great advantage, as OS and antivirus are more and more resistant to encryption-based cyberattacks such as *WannaCry* [10].

Furthermore, in order to explore the possibilities of a normal, yet malicious user, we consider that the targeted device is a black-box for the attacker. This assumption is realistic as SoC companies do not often release information about consumer devices. Therefore, only the publicly available information is used to perform the attack. Moreover, physical access to the system is not required for the attacker, instead the attacker must be able to execute on the victim's device a corrupted application. In the smartphone domain for instance, this application could be downloaded on the phone. Thus, only access to voltage and frequency regulators is required. This later condition can be fulfill through different techniques that will be presented in next Section. These attack conditions raise some important implementation challenges that are discussed in Section V.

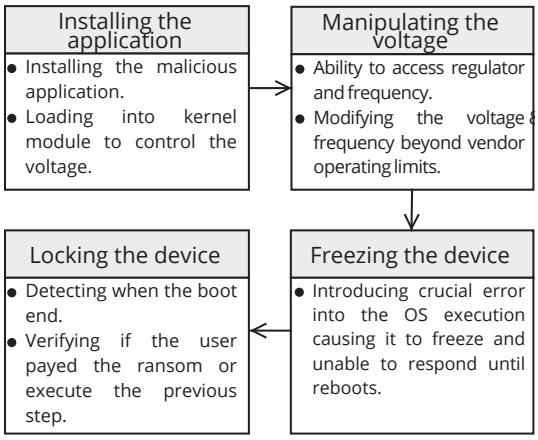

Fig. 3: Required steps from the attacker point of view

B. Targeted System

In today's multi-core systems cores are generally grouped according to their type into different clusters. One example is the ARM big.LITTLE technology which is widely used in industrial SoCs. This architecture includes 2 (or more) clusters of homogeneous cores:

- the big cluster includes 4 high performance cores for heavy load tasks.
- the LITTLE cluster implements 4 small cores for low load tasks and less power consumption.

The main advantage of this architecture is it's ability to dynamically adapt the computation load to higher or lower performance cores according to the needs in order to reduce the overall energy consumption. Most of the SoC using ARM big.LITTLE technology are equipped with per cluster DVFS (see Fig. 1). Consequently, all cores within a cluster share the same voltage and frequency level and are all impacted by voltage/frequency modifications. Rarer implementations such as the Qualcom Snapdragon 800 [11], support per-core DVFS.

IV. Malicious DVFS manipulation technique

The main steps to implement the proposed attack are summarized in Fig. 3.

Installing malicious application: the proposed attack does not require any physical access to the victim device. Instead the access to the device is done through a corrupted application which executes into the device. This later can be installed by the user (the victim), downloaded on an app's store for example.

Manipulating the voltage: Target devices are supposed to be equipped and to support DVFS, granting the user the

ability to modify the voltage and frequency. Usually the vendor will specify some operating voltage and frequency points (operating points) where the device runs in perfect and stable conditions. However, these operating points are not hardware enforced. Our purpose is then to manipulate the voltage beyond those limits, while the frequency is fixed in order to force the system into an unstable operating mode. This step can also be done by fixing voltage and modifying frequency. However, the target device is a black-box and most of the registers are unknown. Thus, modifying the frequency can only be done through cpufreq which will add latency between each change. Note that there is also no safeguard limit when choosing a voltage level due to the high difference of a critical path between different chips, even between dies on the same wafer, making it very difficult to set a physical limit for the regulators.

Freezing the device: Modifying the voltage beyond the SoC operating limits results in the introduction of faults that might be critical in the execution of the OS. These can cause the device to freeze and to be unable to recover until it reboots.

Repeatedly locking the device: The attacker must have the ability to permanently keep the device outside its stable Voltage/Frequency zone either by changing them during boot or just after the booting phase. Thus, gaining the ability to modify the voltage once the device finishes booting.

In the next Section, these different steps and the faced challenges are explained.

V. Implementation and results

A. Targeted system

For proving the feasibility of the presented attack technique, we chose two different recent commercial SoCs widely used in the smartphone domain. The Odroid Xu4, that uses an octacore ARM big.LITTLE architecture (same chip than the Samsung S5 phone and the Exynos 5422), and the Hikey960 board using a more recent chip and sharing the same SoC that in the Kirin 960 architecture (from 2018) used in the Huawei Mate P10 smartphone.

There is an important difference between these two boards related to our work: In the Odroid Xu4, it is possible to control the frequency and the voltage independently, while for the Hikey board, the voltage can only be configured for a certain frequency, which means that the voltage will automatically change according to the frequency value. Moreover, the Hikey board is more recent and sophisticated and includes a co-processor responsible for the dynamic frequency and voltage level selection. These devices are widely used in the smartphone industry and we then choose to deploy the attack using an Android software stack which is build on an UNIX-based OS (namely Linux).

B. Implementing the attack

We assume that a malicious application is executing into the victim device (e.g., smartphone users). This application requires to gain access to the control of the device voltage. There are two ways to access voltage (and frequency) regulators in a Unix kernel-based OS:

By directly accessing the hardware voltage/frequency registers: However, in most of the cases, commercial SoCs used in the smartphone industry cruelly lack from technical documentation and vendors rarely provide the source of their OS. Consequently, most of the registers addresses are unknown.

By using the Linux regulator API [12]: In this case, only the name assigned to the regulator by the driver is needed. This last can be found in */sys/class/devices/regulator/regulator.number/name*. In this work, this later solution was chosen.

In the Odroid XU4 board, the regulator for the big and LITTLE cluster are called "vdd_arm" and "vdd_kfc" respectively. In the Hikey board, even if there is a coprocessor responsible for the DVFS management, we were able to directly communicate through messages the desired new values for voltage (as well as for frequency) without any further control or required access rights.

After ensuring that the malicious application can access these regulators, the next step is to set the exclusive access and denying any other thread from accessing it (e.g., *cpufreq*). Exclusive access to regulators is possible as long as no other modules are currently using it. In order to ensure this last condition, modules using them must be first disconnected, for instance, by binding and unbinding the PMIC driver. This procedure will reset the regulators class structure, and free them.

Finally, the malicious application must request an exclusive access again and bind it to the kernel module. All those steps must be done in the kernel space, consequently, the attacker application must have privileged access. In android systems, the root mode is deactivated. However, *rooting* an android or *jailbreaking* an iPhone is a common and popular practice for users who want to gain full control on their smartphones and push them beyond the limits. This practice is a type of *hacking of consumer electronics*. For this purpose a great number of applications or custom ROMs were developed (e.g., *magisk* [13]), able to unlock root privileges in the system without requiring any password. This consumer community can count up to hundreds of thousand users (e.g., up to 25 million *magisk* downloads in the last 5 years [14]). Those users can all be considered as potential target of the malicious usage of DVFS presented in this paper.

C. Characterization results

The first step to maliciously exploit DVFS mechanisms is to characterize the SoC in order to determine the voltage and frequency limits. These limits can be determined experimentally. For this purpose, the victim and the malicious applications must be executed on two different voltage/frequency islands. This will allow the malicious application to control the voltage/frequency level of the victim island without self-faulting. In our case this condition implies that the victim and malicious applications must be executed into different clusters (for instance by task pinning them into different cluster resources). For these experiments, the malicious application controls the

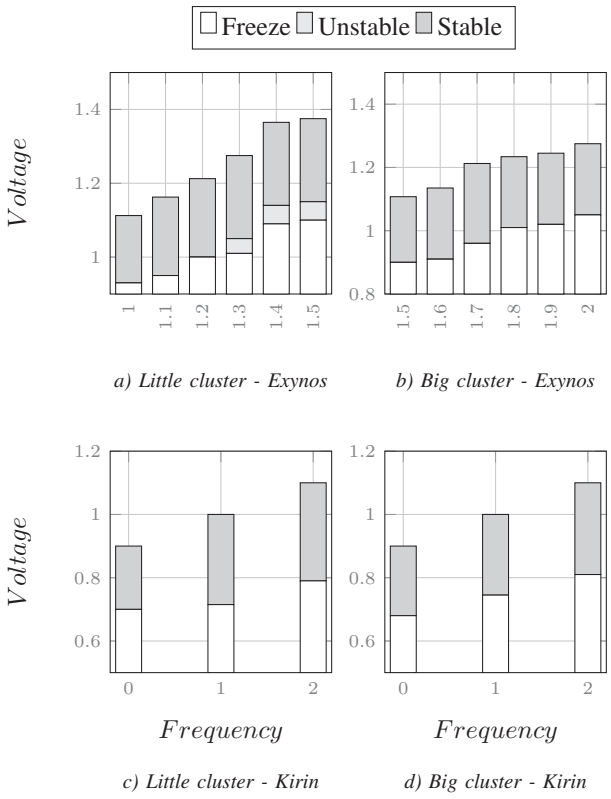

Fig. 4: Characterization of the Exynos 5422 and Kirin 960 clusters behavior when maliciously manipulating the voltage/frequency.

victim's cluster energy management by fixing the frequency to a certain value while setting the voltage level beyond the vendor limits. Results, gathered in Fig. 4 for Odroid and Hikey boards respectively, show different behaviors according to two distinct voltage thresholds:

- First critical threshold: the victim application execution is interrupted due to introduced faults , in most of the cases *segment faults* showing an attempt to access files in the memory with a NULL pointer. In rarer cases, the introduced faults entail *illegal instructions*. Notice that the malicious application is unaffected.
- Second critical threshold: after a second threshold reached, the entire system no longer responds making all services and data inaccessible until the system reboots.

D. Locking out a device

After the system characterization, it could be possible to precisely control the injected faults by setting the voltage to a value extremely close to the second threshold for a very small and precise instant. This will allow to control the exact instruction that will be affected by the voltage glitch. However, we are limited by the minimum time required for the regulator to set a new voltage value. To investigate this point, we developed a kernel module to periodically modify the voltage level and we monitored the SoC supply voltage to determine the latency required between two consecutive voltage changes. Our experiments in Odroid board show that 2.5ms are required to set a new voltage level, and this independently of the frequency value selected. During these 2.5ms if the CPU is running at the lowest frequency (200MHz) around 500.000 instructions are executed. This is 10 times higher than the required amount of instructions needed to succeed in classical fault injection. This experiment proves that there is no control nor high precision possible over the fault injection in this scenario. However, after further investigation we show that it is still possible to maliciously exploit DVFS in this scenario. Indeed, if the device voltage is set to a value between the two thresholds, this will prevent some applications from working properly, and after the second voltage threshold, all system's services and applications stop working and become inaccessible until the system reboots.

Finally, in order to prevent the device owner from taking control of the device after rebooting, solutions were developed for each targeted board. First, for Odroid running Ubuntu OS, the malicious kernel module responsible for accessing the system regulators and used by the malicious application, is permanently loaded during the booting process. This was implemented without any control nor further required rights. Second, for the Hickey running Android Open Source Project, it is not possible to permanently load a kernel module. However it can successfully be loaded just after the booting process each time the system reboots preventing the user to take control of the system. This has been implemented through the Android *Intent* functionality which allows communication between threads. *Intent* works as interruptions, in our case the malicious application is able to detect the end of the booting phase (through the *ACTION_BOOT_COMPLETED* event) which triggers the load of the malicious kernel module.

A possible exploitation of the malicious manipulation of DVFS presented in this work is illustrated in Fig. 5. After detecting the end of the booting phase, the load of the malicious kernel module is triggered. At the same time, the attacker can verify if a certain ransom for instance has been

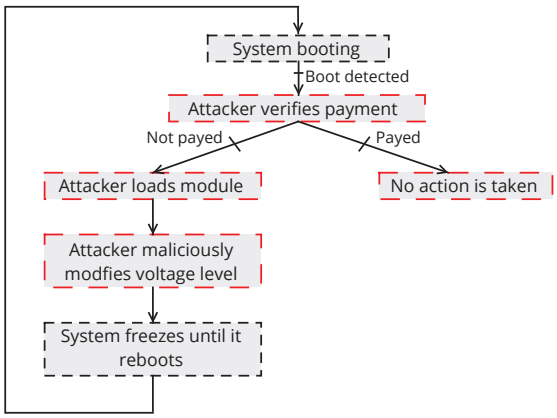

Fig. 5: Summarized steps required for the attacker to successfully implement the attack

payed which will allow the user to take control of the device back again. Otherwise, the malicious application will take control of the energy management of the system in order to manipulate the voltage and frequency levels and force the system to freeze until it reboots again.

VI. Discussion

This type of malicious usage of DVFS was developed in the case where stealing information is not possible, either due to Android SafetyNet API or due to the inability to inject faults with the required precision and the lack of documentation on the target. This work proves that it is still possible to implement a simple yet sophisticated attack to permanently lock out a device, potentially until the user pays a ransom. The main objective of this work is to highlight the security flaws of energy management of today's commercial devices.

While in this work the frequency was fixed and the voltage was dynamically modified, the contrary is also possible if the board documentation is available and gives information to be able to directly access the registers of the PLL. Notice that Unix *cpufreq* does not provide the required control for setting frequency values (the minimum frequency step being 100MHz).

It is also worth noting that this attack can be implemented to any smartphone or architecture using DVFS or similar energy managers as long as the attacker can control the voltage and frequency and drive regulators beyond the recommended voltage. Therefore, the most straightforward countermeasure would be to add hardware or software limits. However, chips usually have different operating limits and there are factors that might affect the critical path (*e.g.*, temperature). Therefore, margins must be taken into account when voltage limits are fixed. Moreover, these limits would require to be bond to given values of frequency for each board.

A second possible countermeasure would be adding more protection over the control of the DVFS. For example only trusted applications running on a higher privileged level than the root user or OS (in case TEE is supported) could access the voltage and frequency regulators. This countermeasure will protect the user against the attack. However, the DVFS, which was developed to reduce the power consumption, will loose some of its benefits.

While some efforts have been done to propose additional chips in order to detect potential faults injection ([7][8]), they aim at classifying malicious glitches where the objective is to gain access to secret data. However they do not aim at preventing them, and therefore they do not have any impact on malicious manipulation of DVFS able to lock the device such as the technique presented in this work.

VII. Conclusion

Today's embedded devices generally support energy management mechanisms. The main objective of this work was to investigate the feasibility and the capabilities of the malicious manipulation of DFVS in today's commercial devices. We considered the example of smartphones. After the implementation of the proposed technique on two recent, widespread ARM-based multi-core boards (Odroid and Hikey), our experiments show that it is possible to maliciously manipulate DVFS mechanisms through software in order to introduce faults into the system operation that can result in the lock out of a device making all system services and data completely inaccessible. With this work we highlighted important security flaws in the energy management supported by most energy constrained embedded systems and the necessity to address them. In future work we plan to investigate possible software and hardware leads for mitigation.

Acknowledgement

This PhD work is founded by Pays de Loire in the frame of the RFI WISE program.

References

[1] D. P. JL Hennessy, "Computer architecture: A quantitative approach. morgan kaufmann publishersinc." San Francisco, CA, USA, 1990.

[2] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, and C. Whelan, "The sorcerer's apprentice guide to fault attacks," *Proceedings of the IEEE*, vol. 94, no. 2, pp. 370–382, Feb 2006.

[3] "Exynos 5422 reference." [Online]. Available: https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-5-octa-5422/

[4] "Arm white paper, big.little technology: The future of mobile, https://www.arm.com/files/pdf/big_little_technology_the_futue_of_mobile.pdf," 2013.

[5] A. Tang, S. Sethumadhavan, and S. Stolfo, "CLKSCREW: Exposing the perils of security-oblivious energy management," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, 2017, pp. 1057–1074. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/tang

[6] "Safetynet api documentation." [Online]. Available: https://developer.android.com/training/safetynet

[7] S. Zhang, A. Tang, Z. Jiang, S. Sethumadhavan, and M. Seok, "Blacklist core: Machine-learning based dynamic operating-performance-point blacklisting for mitigating power-management security attacks," in *Proceedings of the International Symposium on Low Power Electronics and Design*, ser. ISLPED '18. New York, NY, USA: ACM, 2018, pp. 5:1–5:6. [Online]. Available: http://doi.acm.org/10.1145/3218603.3218624

[8] "Fame, fault-attack awareness using microprocessor enhancements." [Online]. Available: https://sites.google.com/view/famechip/

[9] L. B. El Mehdi Benhani, "Dvfs as a security failure of trustzone-enabled heterogeneous soc," *25th IEEE International Conference on Electronics Circuits and Systems*, 2018.

[10] "White paper - the wannacry ransomware attack," 2017. [Online]. Available: http://cert-mu.govmu.org/English/Documents/WhitePapers/WhitePaper-TheWannaCryRansomwareAttack.pdf

[11] "Snapdragon 800 reference." [Online]. Available: https://www.qualcomm.com/products/snapdragon-processors-800

[12] "Voltage and current regulator api." [Online]. Available: https://www.kernel.org/doc/html/v4.15/driver-api/regulator.html

[13] "Magisk costum rom." [Online]. Available: https://github.com/topjohnwu/Magisk

[14] "Download counts of magisk," 2019. [Online]. Available: https://forum.xda-developers.com/apps/magisk/official-magisk-v7-universal-systemless-t3473445