A path to scale up proven hardware-based security in constrained objects

Nicolas Dejon, Chrystel Gaber, Gilles Grimaud

Compas'2020 : Parallélisme/ Architecture/ Système / Temps Réel
MILC - Lyon, France, du 30 juin au 3 juillet 2020

A path to scale up proven hardware-based security in constrained objects

Nicolas Dejon[†‡], Chrystel Gaber[†] et Gilles Grimaud[‡]

[†]Orange Labs,
Châtillon, France
name.surname@orange.com

[‡]Univ. Lille, CNRS, Centrale Lille, UMR 9189
CRIStAL - Centre de Recherche en Informatique Signal et Automatique de Lille,
F-59000 Lille, France
name.surname@univ-lille.fr

Abstract
The Internet of Things is revolutionizing the established embedded systems market. However, embedded system designers aren't traditionnally considering security as a top concern and consider maintaining their current tools and technologies while opening to connectivity. It is thus expected the embedded systems sector will suffer massively from cyberattacks with the risk to break down traditional IT systems with them. This article demonstrates how this situation is a plausible scenario given the current context and exposes how the existing means to build and secure constrained devices are not sufficient enough to reduce the risks. As a countermeasure, we propose the creation of isolated hardware-enforced trustworthy environments targetting constrained devices in need of strong guarantees by adapting the formally proven Pip protokernel. We explore as well what considerations must be beared in mind to scale up the adoption of the proposed solution.

Keywords: constrained objects ; hardware-enforced isolation ; proven kernel

---

1. Introduction

The Internet of Things (IoT) revolution is happening and, as a consequence, various sectors undergo restructuration among which the embedded systems. IoT devices compete with legacy systems, shimmer a better connected future and are increasingly popular in the society. This paper proposes to adapt a proven kernel that would make the embedded systems and the IoT worlds both benefit of hardware-enforced isolation. Section 2 explores the current embedded system design situation and how they can benefit from partitioning to establish Trusted Execution Environments (TEEs). Next, in section 3, we will see how the embedded world is settling down in the IoT and how the IoT is currently struggling with security. Section 4 demonstrates how current tools and technologies at disposal for constrained devices on the move for connectivity are not sufficient to face the threats presented in the previous section. Then, section 5 highlights why relevant solutions are not adapted for constrained devices. Lastly, section 6 introduces our proposal to adapt a proven kernel which enables hardware-enforced isolation

properties. We establish as well the conditions under which it could scale up for an effective, almost directly applicable and massive injection of the proven isolation capabilities for constrained objects. Section 7 wraps up the content of the statements made in this paper.

2. Embedded systems: low-costs but expensive expectations

Marketing and production time are driving many sectors, which also applies to the design of embedded systems. With a short product lifetime, market penetration drives the schedules, with early adopters winning the biggest market shares. Time-to-market is thus a critical factor conducting the development of embedded systems.

2.1. The battle to reduce the costs: grouping and partitioning

Tightly coupled with time-to-market are costs. They need and are expected to be as low as possible during the complete (potentially long) life cycle of the systems including design, manufacturing, purchase, installation, use and maintenance. Higher costs are to be found in distributed systems where system functions are spread on several devices, multiplying this way the space, weight, power, costs (SWaP-C factors) as much as the size of the distributed system. This traditional approach has the advantage of inherent fault-containment, but neglects poorly coordinated control, and complex and fault-prone pilot interfaces, inducing a safety cost, "mode confusion" and causing fatal airplane crashes [18]. Instead, grouping all these system functions on a same platform would drastically reduce the costs but raises some concerns: there could be design bugs leaving the system unfunctional or, even worse, unreliable.

Rushby introduces therefore the notion of a separation kernel [19]: it should reproduce a distributed system on a single device with the same initial guarantees. The separation kernel relies on partitioning, where no partition hosting system functions should behave differently as if they were physically separated. In other words, a failure in one partition should not propagate to other partitions, so the purpose of partitioning is fault-containment.

2.2. Challenges for a separation kernel

Spatial isolation protects partitions running in a shared environment from accessing or altering each other's content. Temporal isolation should also be enforced to ensure real-time services stay unaffected from the behavior of other partitions.

Moreover, as Rushby shows, fault-containment shares common fights with cybersecurity. Indeed, Software-Of-Unknow-Provenance (SOUP) or Commercial-Off-The-Shelf (COTS) Software, that could be faulty or malicious and most of the time proprietary protected black-boxes, mixed with mission-critical software could lead to non-functioning system and/or system exploited to perform attacks and information leakage. While a separation kernel protects the sensitive assets from the previous concerns, achieving isolation is a critical prerequisite.

2.3. Achieving isolation

Isolation creates a trusted environment where the integrity, confidentiality and availability of isolated components are protected. From there, a Trusted Computing Base (TCB) can be composed by any components that need to be trusted (and not necessarily trustworthy) for the system to be reliable for software execution. This isolated partition could be the base of a Secure Execution Environment (SEE), prerequisite for a Trusted Execution Environment (TEE) [20].

Isolation techniques vary in costs, performance and heaviness and could be mainly classified in two categories: compile-time and runtime techniques. Among compile-time techniques, we find namespaces but mixed in the final compiled executable, strongly typed languages like Rust or unit code testing however depending on the quality of the tests and the code coverage. Rutime isolation techniques could be further sub-divided in software runtime isolation and hardware runtime isolation. On the software side, it could for example be assertions or byte-

code verifiers. Hardware-based isolation consists of hardware support dedicated to achieve isolation [4]. A taxonomy of hadware-based isolation used in mainstreams OSs and kernels for contrained devices is proposed in Table 1.

3. Upheavals caused by the IoT emergence

IoT means world wide connected embedded systems [6, 7]. But bringing connectivity to legacy systems also exposes these devices to cyberattacks as never before, which competes with the expected reliability of IoT devices; sometimes we even trust them with our lifes. For example, in 2015, two researchers managed to remotely hack a Jeep Cherokee, taking control of some inboard systems and critical physical systems such as the steering and braking, foresighting tragical scenarios [14]. Moreover, IoT security is not only about attacks the way down to the devices, but also the other way around. For example, the Mirai botnet which took place in 2016 broke down major parts of the internet by taking control of a large number of devices [13].
However, as exposed previously, time-to-market drives the embedded development and figures constrast sharply speaking about security: security is the top IoT developer concern whereas it ranks last in the embedded world. Furthermore, only 4% of the design time of an embedded system is spent on security/privacy threat/risk assessment. However, the embedded world seems to acknowledge the need for security in their systems since security is the top 3 greatest technology challenge [6].
Overall, on the 25 billion of IoT devices expected on the market in 2021 [8], the great majority of them will suffer from cyberattacks and the new business ecosystem will be the playground of cybercrimes capturing 300 billion to 2 trillion dollars worldwide and every year [17].
With potentially high impacts, the ecosystem is in a stringent need of high-level guarantees.

4. Composing the TCB for secure embedded systems

4.1. Market insights and available technological bricks

IoT devices and embedded systems have common grounds, reflected in their use of the technologies. One of the major similarities is the choice of the CPU architecture: Arm dominates thanks to its power efficiency. Arm is driven by the constrained device market capturing 67% of the market shares thanks to the ARMv7 architecture [7]. Arm estimates one trillion of devices to be on the market in 2035 [22] (40 billion in 2025) and gains popularity among the embedded vendors [6] even with a dominant position. 32-bit architectures win more than 60% of the market [7], but 64-bit architectures gain popularity in the recent years [6]. Developers of the two worlds choose about the same operating systems. On the IoT side, FreeRTOS dominates constrained devices, followed by Contiki-NG, MbedOS, RIOT OS and QNX [7, 21]. On the embedded side, Embedded Linux, FreeRTOS, Android and custom/inhouse OSs are mostly used [6].

4.2. Focus on the Memory Protection Unit

A computer system usually has two or more privileged levels. The most privileged ones are able to disable interrupts, change system settings, switch privileged modes, and reconfigure the Memory Protection Unit (MPU) among other things. This prevents lower privileged levels to do so and potentially give untrusted code rights for full system control. However, non-privileged code still can access any memory locations, peripheral devices and have executable rights over unwanted regions like out of RAM causing unacceptable threats. The MPU is there to overcome this unwanted scenario.
The MPU can be traced back as far as in the ARMv4t. It is a programmable unit inside the processor, configurable at runtime by a privileged software (typically an OS kernel). It can be used to protect the system address space by dividing the memory into memory regions. The

MPU defines these memory regions to have specific access permissions (read, write, execute) and memory attributes (cacheable, shareable...). The number of memory regions that can be configured is implementation-defined, generally 8 to 16 regions.
In the Cortex-M series, architectures ARMv6/v7/v8 support respective Protected Memory System Architectures (PMSA) as an optional extension. The MPU implements this extension to protect a 4GB address space. As such, all micro-architectures of the Cortex-M series propose an optional MPU, except Cortex-M0 and Cortex-M1 [1].
Since Arm dominates the mobile, embedded and IoT markets, it means the MPU is de facto the most widely available hardware component to protect memory assets and already used in many OSs for constrained devices. However, the reality is different, the MPU is not present in all implementations or, when it is present, is set aside. Several reasons push device manufacturers not to use the MPU: too high memory footprint, too high energy consumption, too high performance overhead, too high time-to-market pressure to take the time to integrate, compatibility issues, limited regions and thus flexibility or no guarantee of system protection [23].
The MPU is thus suitable enough to protect the memory space if properly configured and if the architecture had been thought to ensure isolation of the system components at any time.

4.3. Perspectives and needs for constrained objects

Constrained objects have inherently very constrained resources: low memory capacities, low energy, low computational power, low operational frequencies. This would correspond to the Arm Cortex-M family (32-bit), MSP430 (16-bit) and AVR (8-bit) for example.
With the IoT revolution pressing the manufacturers, but the reluctance and maybe helpless situation to ensure security guarantees for their products, big players of the sector announce almost turnkey security solutions. For example, Arm created the TrustZone technology to provide a hardware-based isolation mechanism that would split the system in a secure world and a normal world. For many years, the embedded systems community seldom used the TrustZone, because of a lack of clarity in its usage [16]. Legacy designed systems can run in the unsecure world and make use of the secure functions in the secure world. TockOS, pushed by Google, shows the interest of heavy actors for security in this sector. Created in 2018, it promises to give each application an isolated space, also isolated from the kernel code. The kernel isolation is carried out by the use of Rust and the application isolation is performed by the MPU. While isolation is hardware-based and allows the creation of a TEE, the guarantees are weak. Open-source code and a large community is enough to create trust but not enough to avoid vulnerabilities, as can be experienced with Linux [5].
However, new technology adoption is not straightforward in the embedded world, and system designers have their expectations [6]. It comes out that the majority of them rely on a 32-bit architecture with ARM architectures prevailing in this category and they usually use the same processor across their projects. For the operating system, they use less and less inhouse/custom OSs. If they have an OS, they are happy with the current solutions and have no reason to switch, also to be able to maintain software compatibility and make use of the expertise and familiarity. The reason to change operating system are mostly because the hardware or processor changed or designers were chosen an OS without any decision taking. It shows they are not willing to change their development environment without any major factor, mostly independent of their will. Among the most important factors to choose an OS is the full access to the source code, the availability of the technical support, the compatibility with other software, systems and tools, no royalties and real-time performances.
Based on Table 1, the MPU seems to be the minimal hardware component that can be used to acquire isolation properties. While TrustZone is heavily pushed to be adopted and ready-

to-use tools made available to accelerate the adoption, it goes against the designers' desires to keep the same processor since the majority of devices are ARMv7 based.

5. Related works

Many operating systems are already widely used for constrained devices and some of the most famous ones are recalled in Table 1. While most of them have been chosen by their popularity and isolation mechanisms, the majority require a manual intervention to set up isolation between components and they are almost all written in an unsafe programming language like C. ProvenCore-M [2] is one, if not the only, to target the microcontroller sector and to expose strong guarantees of isolation by formal proofs. It proposes an OS architecture close to PSA, either to sandbox applications or to set at disposal secure functions. It makes use of the TrustZone if available or a dual-CPU architecture where it runs in a separate CPU. It is available to the industry. The main drawback is its proprietary classification which is why few inner workings have been disclosed. For bigger systems, seL4 [9] is also a kernel OS that exposes strong guarantees with formal methods and available for the industry, however open-source. The major drawback is that it doesn't target small objects for now, using the MMU as a hardware-enabler isolation, hardware not present in tiny objects. Also, while the TCB is kept minimal, it exposes the features of a complete kernel like memory capabilities, making the learning curve less obvious. In the same sector, Pip [11] is instead a protokernel, open-source as well, drastically lowering down the size of the TCB with only memory management and context switching features. Isolation is also formally proven and is MMU-based. The small exposed feature set makes it very flexible for upper implementations, so that designers could keep their actual project while soaking the system with strong isolation guarantees. Again, the major drawback is that it doesn't fit constrained devices. The MINION [12] architecture also identified the reluctance of system designers to use the MPU and proposed a way to automatically analyse statically a firmware and reorder the memory sections to group the ones of similar nature and isolate processes and kernel. However, this time, it is not formally proven and subject to more vulnerabilities because of memory view over-approximation.

6. Creating the TCB for secure constrained objects with the Pip protokernel

Following the remarks and enlightments of the previous sections, it does not seem possible to build a system for constrained objects which provides strong guarantees of isolation with current tools and technologies while completely satisfying system developers' expectations. Because it is open source, formally proven, flexible, but using the MMU, we propose to adapt the Pip protokernel so that it fits the imagined solution depicted in the Table 1 (*Pip-MPU*).

6.1. The Pip protokernel

Far from the monolithic kernels like Linux which exposes a huge TCB (36 millions of LoC [15]), the microkernels are reducing this TCB to some specific modules, the remaining becoming external services in the user space. Pip belongs to the protokernel family which reduces the TCB to two mechanisms: memory and control flow management. This means only these two kernel features can run in a privileged level (kernel space), the remaining, and so the applications and OS, run in an unprivileged level (user space) and can use the minimal kernel API of 9 system calls. This is enough to create simple applications but can be complemented by a rich OS to retrieve full OS features. Pip supports a recursive partitioning schema where each partition can donate a part of its memory to another partition. The relation created by this memory donation creates a hierarchical view where each partition is part of a partition tree. In that respect, a root partition is created during the system initialization and will be the parent of child partitions, themselves possibly parents to other paritions and so on. By ensuring recursive memory attri-

bution, the kernel never experiences memory exhaustion and respects the availability security pilar. Furthermore, the kernel and the partitions are isolated, just as well as between the partitions themselves, ensured by formal proofs on each kernel API. Each partition is thus a trusted environment with strong guarantees of isolation.
While the formal proofs are conducted in the Coq Proof Assistant [10], the Pip API is available as a C library, thus understandable and open to investigation by any embedded developer since C and C++ are the most used languages in that area [6].

6.2. The challenges to port Pip-MPU and future work

The port from an MMU-based Pip to an MPU-based Pip raises some challenges. The proven isolation principle remains unchanged since it builds up an effective trusted environment. As a consequence, the properties should stay the same and most of the kernel API as well. What is more uncertain is the modification of the kernel code. Indeed, while MMU and MPU share the memory protection concept, they work differently. First, the MPU works directly on physical addresses while the MMU enables paging to fragment the memory in smaller equally sized pages (usually of 4KB). Since the proofs in Coq make use of properties that relate to the MMU's inner workings, this part could be heavily impacted. Also, Pip uses dedicated structures to keep trace of the partition tree and accelerate the MMU's table configuration which depends again on the MMU. The challenge is to combine the two perspectives and investigate how straightforward it is to recycle code and proofs to cover both.
Next to investigate is its compliance with constrained devices. As shown, the MPU is suitable for this type of devices. What is to show is that the design of such a system answers the expectations of the system designers expressed earlier. First of all, the TCB created by Pip is minimal and as such has a very low footprint. Pip is also open source and will still be after the port. At the opposite of security-by-obscurity, its trustworthiness emanates from the proofs and the simple architecture comprehension. Pip is perfectly compatible with the developers claim to keep the same development environment, notably their OS. Pip, as a protokernel, lies below the OS layer which will just be deprived from the memory management and context switching features. For the application developers, the port would be almost transparent. However, this means the port should go along with reference implementations of modified Pip-compatible OSs. Such an approach has already been demonstrated with FreeRTOS and Linux [3]. Lastly, many device users expect real-time performances. On the one hand, as the kernel code is static and atomic, the determinism of the operations can be ensured and, as for the current version, has reasonable performance overhead. As embedded systems are generally statically designed for real-time concerns, the number of times the kernel APIs will be called and thus this overhead can be prior evaluated. On the other hand, context switching determinism is still under study and needs to be evaluated to check its compatibility with real-time scenarios. The transition to the industrial scale of the proposed technology will need these appropriate measures as key indicators of an adoption possibility to quickly and massively inject isolation.

7. Conclusion

Embedded devices become more and more connected and will most probably suffer from massive cyberattacks. Indeed, current tools and technologies seldom include security mechanisms, and even less propose better protecting hardware-based mechanisms. Making use of the MMU, the Pip protokernel implements the isolation principle for a separation kernel with formally proven kernel APIs. We showed in this article that a modified version of Pip, based on the MPU instead of the MMU, could be suitable to secure constrained devices while following the expectations and needs of the current embedded developers. For a massive adoption of the technology, future works will embrace the direction overcoming the identified challenges.

Bibliographie

1. Arm . – Website of : Arm cortex-m differences (arm). – https://developer.arm.com/ip-products/processors/cortex-m/cortex-m0/, 2020. [Online; accessed March 27, 2020].
2. Prove&Run . – Website of : Provencore-m (prove&run). – https://www.provenrun.com/products/provencore/, 2020. [Online; accessed March 27, 2020].
3. Boyer (F.), Grimaud (G.) et Cartigny (J.). – *Thèse Quentin Bergougnoux*. – Thèse de PhD, Université de Lille, 2019.
4. Copy (D.), Viswanathan (A.) et Neuman (B. C.). – A survey of isolation techniques. *Information Sciences*, pp. 1–16.
5. CVE. – Website of : Cve, linux kernel vulnerabilties. – https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=linux+kernel, 2020. [Online; accessed January 17, 2020].
6. EETimes. – 2019 Embedded Markets Study. noMarch, 2019.
7. Foundation (E.). – IoT Developer Survey 2019 Results. vol. 0, nApril, 2019.
8. Gartner. – Top strategic iot trends and technologies through 2023. – https://www.gartner.com/en/documents/3890506-top-strategic-iot-trends-and-technologies-through-2023, 21 September 2018. [Online; January 17, 2020].
9. Heiser (G.). – Secure embedded systems need microkernels. *USENIX; login*, vol. 30, n6, 2005, pp. 9–13.
10. INRIA. – Website of : Coq. – https://coq.inria.fr. [Online; accessed January 17, 2020].
11. Jomaa (N.), Nowak (D.) et Torrini (P.). – Formal Development of the Pip Protokernel. 2018.
12. Kim (C. H.), Kim (T.), Choi (H.), Gu (Z.), Lee (B.), Zhang (X.) et Xu (D.). – Securing Real-Time Microcontroller Systems through Customized Memory View Switching. noFebruary, 2018.
13. Krebs on Security. – Website of : Ddos on dyn impacts twitter, spotify, reddit. – https://krebsonsecurity.com/2016/10/ddos-on-dyn-impacts-twitter-spotify-reddit/, 2016. [Online; accessed March 27, 2020].
14. Miller (C.) et Valasek (C.). – Remote Exploitation of an Unaltered Passenger Vehicle. *Defcon 23*, vol. 2015, 2015, pp. 1–91.
15. Openhub. – Website of : Linux statistics. – https://www.openhub.net/p/linux, 2020. [Online; accessed January 17, 2020].
16. Pinto (S.) et Santos (N.). – Demystifying arm trustzone: A comprehensive survey. *ACM Computing Surveys*, vol. 51, n6, 2019.
17. Radanliev (P.), De Roure (D.), Cannady (S.), Montalvo (R.), Nicolescu (R.) et Huth (M.). – Economic impact of IoT cyber risk - analysing past and present to predict the future developments in IoT risk analysis and IoT cyber insurance. *Living in the Internet of Things: Cybersecurity of the IoT - 2018*, 2018, pp. 3 (9 pp.)–3 (9 pp.).
18. Rushby (J.). – Partitioning in avionics architectures: requirements, mechanisms and assurance. *Work*, noMarch, 2000, p. 67.
19. Rushby (J. M.). – Design and verification of secure systems. *Proceedings of the 8th ACM Symposium on Operating Systems Principles, SOSP 1981*, vol. 15, n5, 1981, pp. 12–21.
20. Sabt (M.), Achemlal (M.) et Bouabdallah (A.). – Trusted execution environment: What it is, and what it is not. *Proceedings - 14th IEEE International Conference on Trust, Security and*

Privacy in Computing and Communications, TrustCom 2015, vol. 1, 2015, pp. 57–64.

21. Silva (M.), Cerdeira (D.), Pinto (S.) et Gomes (T.). – Operating Systems for Internet of Things Low-End Devices: Analysis and Benchmarking. *IEEE Internet of Things Journal*, vol. 6, n6, 2019, pp. 10375–10383.
22. Sparks (P.). – The route to a trillion devices The outlook for IoT investment to 2035. *ARM Whitepaper*, noJune, 2017, pp. 1–14.
23. Zhou (W.), Guan (L.), Liu (P.) et Zhang (Y.). – Good Motive but Bad Design: Why ARM MPU Has Become an Outcast in Embedded Systems. 2019.

Table 1: Taxonomy of the hardware-based isolation mechanisms natively present in constrained devices OSs and kernels.

					Hardware isolation possibilities				
OSs/kernels for constrained devices	*Full source code access*	*Maturity*	*Portability*	*OS flexibility*	*type*	*guarantees*	*possibilities*	*flexibility*	*hardware components*
`TrustedFirmware-M based systems (MbedOS, Zephyr, FreeRTOS Amazon)`	✓	Industry	ARMv8		enclaves		2 isolated domains (secure/non-secure)	PSA Level 1	TrustZone
	✓	Industry	ARMv8		enclaves		2 isolated domains (secure/non-secure)+ compile-time fixed number of isolated secure partitions in secure world	PSA Level 2&3	TrustZone with MPU in secure world
`MbedOS`	✓	Industry	ARMv6 ARMv7 ARMv8		apps enclaves		MPU Management (RAM execute lock + ROM write lock) + netsocket (EMAC drivers features)		MPU (auto enabled)
	✓	Industry	ARMv8		enclaves				TrustZone w/wo MPUs in secure world (see TF-M)
	✓	Industry	ARMv8		enclaves		For Musca_A1/B1 boards: SPM implementation (same possibilities as TF-M TrustZone with MPU)	(Non-secure world not protected)	TrustZone with MPU in secure world
	✓	Industry	ARMv8		enclaves		2 isolated domains + compile-time fixed number of isolated secure partitions in secure world + other OS/app isolation using MPU in non-secure world	PSA Levels 2&3 + non-secure app-defined MPU regions	TrustZone with MPU in secure world and MPU in non-secure world
`MbedOS2 + uVisor (deprecated)`	✓	Industry	ARMv6 ARMv7 ARMV8		enclaves		secure boxes		MPU
`RIOT-OS`	✓	Research and Industry	ARMv6 ARMv7 ARMv8 MSP430 RISC-V	Memory-limited thread number	✗	✗	✗	✗	✗
`ProvenCore-M`	✗	Industry	ARMv7 ARMv8		enclaves	Proven	secure functions (same as TF-M)	Memory-limited applications of same size	Probably MPU
	✗	Industry	ARMv8		enclaves	Proven	isolated processes secure functions (same as TF-M)	Same as TF-M	TrustZone
	✗	Industry	ARMv7		enclaves	Proven		Same as TF-M	dual CPUs
`Tock OS`	✓	Research and Industry	ARMv6 ARMv7 RISC-V	High memory context per process	apps enclaves		sandboxed processes	Memory-limited applications	MPU
`FreeRTOS (Amazon)`	✓	Industry	ARMv6 ARMv7 ARMV8 MSP430 RISC-V		✗	✗	✗	✗	✗
`FreeRTOS-MPU`	✓	Industry	ARMv7 ARMV8		enclaves		user defined regions	3 enclaves per task	MPU
`Pip-MPU (target)`	✓	Industry	ARMv7 ARMv8		recursive enclaves	Proven	Recursive hierachical partitions	Memory-limited hierarchical partitions	MPU