# Perspectives on security kernels for IoT

Nicolas Dejon
Orange Labs, Châtillon, France,
Univ. Lille, CNRS, Centrale Lille,
UMR 9189 - CRIStAL -
Centre de Recherche
en Informatique Signal
et Automatique de Lille,
F-59000 Lille, France
nicolas.dejon@orange.com

Chrystel Gaber
Orange Labs,
Châtillon,
France
chrystel.gaber@orange.com

Gilles Grimaud
Univ. Lille, CNRS, Centrale Lille,
UMR 9189 - CRIStAL -
Centre de Recherche
en Informatique Signal
et Automatique de Lille,
F-59000 Lille, France
gilles.grimaud@univ-lille.fr

*Abstract*—**IoT market's growth surge encouraged developers to focus on fast delivery rather than security resulting in several major attacks. Efforts to provide secure-by-design applications or IoT devices rely on trusting physical secure elements or on the lower software layers. Thus, the entire system roots its overall security in the kernel given it is the first software layer above the hardware. However, constrained objects often struggle to combine functionality and security due to inherent low resources and few mechanisms address this problem. In this article, we explore existing approaches and highlight the need for a minimal and formally proven root of trust for constrained objects while presenting the challenges this implies.**

## I. Introduction

Today, Internet of Things (IoT) enters a popularization phase and impacts the professional and personal spheres of a growing number of users. IoT is a paradigm that evolved steadily since its apparition in 1999, and according to most forecasting reports, it will become generalized in the years to come [1].

The pace of change and the associated consequences will be pushed by the number of connected devices expecting to explode in the next few years. Indeed, Gartner, which closely follows the trends in IoT, expects a high market adoption by 2023. Gartner calculated that 8.4 billion IoT things were in use in 2017 and forecasts 14.2 billion connected things in 2019 (almost double in two years) and will eventually reach 25 billion by 2021. These devices will conquer homes and industries and the market will mostly be driven by consumer devices [2].

This substantial twist in the IT market may revolutionise the business environment: costs reduction through the collected data, new business opportunities, and new services. However, IoT products and services are disadvantaged compared to non-connected devices, because of the concern over cybersecurity risks.

This clearly shows that the ecosystem is at risk and must be protected in order to build a resilient user trusted system. The research community is particularly concerned about the IoT security topic, given the number of devices to be connected and the lack of effective countermeasures preventing the current security issues and attacks. By its very nature, the whole IoT ecosystem could fail from internal (unintentionally?) flawed devices that could lead to business disasters, user threats or even life threats.

The explosion of the IoT devices brings a plethora of IoT device applications. In the recent years, these applications leverage advanced security mechanisms and the scientific community made significant efforts on the static and dynamic security analysis of IoT applications [3], [4], [5]. However, they all need to trust the software stack underneath and ultimately the layer in interaction with the hardware (CPU, memory, devices). The latter feature is usually accomplished by the kernel and becomes in such way the first software Trusted Computing Base (TCB) of the firmware. If the kernel is badly designed or compromised, no assumptions can be made about the applications that run above, weakening the security trust in the firmware and potentially compromising the whole IoT ecosystem as a consequence.

This paper explores how to build a minimal kernel for constrained devices with strong security guarantees. We show that IoT devices could benefit from formal methods to reach the expected security level and we identify the current challenges. Section I discusses the requirements for a secured IoT ecosystem built up from trusted devices, which roots in the firmware and its kernel security. Then, section II analyses the state-of-the-art for secure kernels with a specific attention on formally proven kernels. Section III describes the IoT device specificities and the obstacles hindering the use of the methodologies described previously. Section IV identifies future works and perspectives in the design of a formally proven kernel. Finally, section V concludes this paper.

## II. State-of-the-Art

### A. Kernels

Kernels vary in size and exposed features. Two main categories are frequently opposed: monolithic kernels and microkernels.

Monolithic kernels are usually used by general-purpose operating systems, like Linux for UNIX-like operating systems. They concentrate all the functionalities required to run exclusively the applications and the user interface in user mode. The kernel itself exposes a large API that surfaces an important code base (Linux is composed by more than 36 millions of lines of code [6]). However, customization can reduce drastically the code base (e.g. MuLinux [7]) and modules can be loaded at runtime which makes it a modular kernel.

Microkernels reject outside the kernel space most of the core features of the monolithic kernels. Thus, they provide only a minimal set of system calls, automatically reducing the attack surface. These system calls are then used by outside servers (programs) that will reproduce the lost features of the monolithic kernels. This way, the kernel code base scales down a lot. However, due to frequent privilege switches between kernel and user spaces, the performances were quite low. The L4 microkernel family [8] solved this issue by identifying four major features: virtual memory management, threading, scheduling and inter-process communication.

Many other kernel types exist, such as the experimental pip ”protokernel” [9] that restricts even further the exposed features. It only provides virtual memory management and context switching, and leaves all the rest in userland.

### *B. Design of a secure kernel*

Each kernel comes with its own set of strengths and weaknesses and reasons to choose a particular design, but they all share the desire to build a secure kernel.

Security by memory isolation for a kernel can be understood in terms of confidentiality and integrity of the data between the programs and with the kernel. Separation kernels [10] create these isolated environments by simulating a distributed system within a single physical machine.

However, in IoT, common security mechanisms like kernel and process memory isolation are usually not employed. Indeed, many IoT devices have real-time constraints that can’t be met by using kernel memory isolation due to frequent privilege switching, or don’t possess the required hardware commonly used like the MMU (Memory Management Unit). Yet, recent efforts made it possible to meet all expectations of a real-time system together with the memory isolation capability using the MPU (Memory Protection Unit) [11]. MPU has also already been showed important to ensure the integrity and confidentiality of Java Card applets by confining the executable code into sandboxed interpreters [12].

Programs don’t necessarily require hardware mechanisms to be confined. For example, remote updates for low-end devices can be provided in the form of Javascript runtime containers [13].

### *C. Design of a formally secure kernel*

Despite the trust we can put in all these systems, the bound may still be broken by human mistakes during the design or the use of outdated or insecure components (e.g. top 10 IoT vulnerabilties [14]), and vulnerabilities are still discovered frequently [15].

Therefore, pip has been designed specifically to bring the highest guarantees on memory isolation by providing formal proofs. These proofs mathematically demonstrate the announced security properties and the functional correctness of the features exposed by the kernel, helped and stamped ”formally proven” by the Coq proof assistant [16]. The proofs consist in a set of system invariants that should be present before and after a kernel system call. They are all provided in the form of Hoare triples.

Since the properties that emanate from the kernel are the basis for the exposed security features, they also need to be trusted. These are made simple enough to convince their trustworthiness.

Memory isolation in pip is hardware-enforced by the use of an MMU. Indeed, by controlling the memory pagination, the system could implicitly build an access control mechanism, restricting the use of the memory per running entity (kernel, operating system, programs, threads...). Memory isolation is thus dependent on the correct configuration of the MMU tables, which is the only role assigned to pip as a protokernel.

The sole feature provided by pip creates a very small TCB making the cost of proof to the minimum possible for a kernel.

## III. Design of a formally secure kernel for constrained objects

IoT devices can be deployed for years and sometimes in difficult reachable areas. Recent works regarding remote accessibility and remote updates like SUIT [17] would benefit from memory isolation guarantees. Indeed, such guarantees would allow the coexistence of modules with different security and update needs. For example, one could design a minimal isolated kernel with little update needs and sandboxed applications with high update needs. Update of the business logic in the sandboxed applications could be more performant since a full update could be avoided. Such devices could also have strong real-time requirements which should cohabit with expected/needed security.

While the state-of-the-art proposals provide some suitable solutions for IoT devices, they are usually not formally proven. In the case they are, pip and other formally proven kernels like seL4 [18] from the L4 family strongly rely on the hardware mechanisms of the MMU to conduct their proofs. Due to their constrained resources such as a minimal memory, these IoT devices would most likely include an MPU instead of an MMU, which implies that these formally proven solutions can’t be totally usable in the pursued context. To the best of our knowledge, MPU-based proofs are not openly provided to the community (e.g. ProvenCore [19]) but convince us that such solutions are reachable.

## IV. Challenges and perspectives

Several scientific challenges and perspectives arise from this point.

Given the presented landscape, there is a possibility to bring native security to IoT devices to replace the trust we have in the kernel with a formal guarantee of security (in the sense of memory isolation).

During this process, new properties required to assure memory isolation in systems could be discovered. The formal verification could follow already employed methods, or explore other techniques like separation logic. Challenges are also to be found in the formulation of the proofs that will need to interface with the real-world, which includes hardware, people and execution environments in the IoT context.

Furthermore, the cost of proof of current solutions is high. IoT devices, even low-end devices with very constrained resources, still are complex systems. Efforts and techniques to bring down this cost of proof are needed to make it suitable even for low-cost devices.

In addition to that, no current formally proven kernels seem to fully cover all existing architectures composed by a different hardware (MMU and MPU together for example). Generalization efforts are needed to cover other hardware-enforced access control mechanisms.

Lastly, the ARMv8-M architecture is currently under a formal verification process [20] and gives the opportunity to explore the proof even beneath the software stack and bridge the proof across the hardware.

## V. Conclusion

Security in kernels is of uppermost important to make any assumption about the overall security of a system. Being the first fundamental software component which interfaces with the hardware, the kernel is expected to have strong security guarantees. This can be achieved by applying formal methods as current solutions propose, however they are usually hardware dependent, not transferable between different architectures and hardware compositions, and not applicable in the context of IoT and its constrained devices because of limited resources or without compromising with the required performances. Future challenges thus include to identify the minimum set of required features to build-up a secure kernel, to push forward and extend current formal proofs to cover distinct architectures at the same time in a cost of proof effective manner, and to trim down the assumptions by exploring the potentials with on-going hardware verification efforts.

## Acknowledgment

The authors would like to address their heartful thanks to Narjes Jomaa.

## References

[1] P. Radanliev, D. C. D. Roure, J. R. C. Nurse, R. Nicolescu, S. Cannady, and R. M. Montalvo, “New developments in Cyber Physical Systems , the Internet of Things and the Digital Economy – discussion on future developments in the Industrial Internet of Things and Industry 4 . 0,” *Preprints*, no. March, 2019.

[2] 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].

[3] D. T. Nguyen, C. Song, Z. Qian, S. V. Krishnamurthy, E. J. Colbert, and P. McDaniel, “IotSan: Fortifying the safety of IoT systems,” *CoNEXT 2018 - Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, pp. 191–203, 2018.

[4] Z. B. Celik, P. McDaniel, and G. Tan, “Soteria: Automated IoT Safety and Security Analysis,” *Proceedings of the 2018 USENIX Annual Technical Conference*, 2018. [Online]. Available: http://arxiv.org/abs/1805.08876

[5] N. Dejon, L. Verderame, C. Davide, A. Armando, and A. Merlo, “Automated security analysis of iot software updates,” 2019.

[6] Openhub, “Website of : Linux statistics,” https://www.openhub.net/p/linux, 2020, [Online; accessed January 17, 2020].

[7] Michele Andreoli, “Website of : Linux statistics,” http://micheleandreoli.org/public/Software/mulinux/, 2020, [Online; accessed January 17, 2020].

[8] J. Liedtke, “Towards real microkernels,” in *Proceedings of the 13th International Conference on Information Security Theory and Practice*, ser. WISTP ’19. CACM, Sep 1996.

[9] N. Jomaa, D. Nowak, and P. Torrini, “Formal Development of the Pip Protokernel,” 2018.

[10] J. M. Rushby, “Design and verification of secure systems,” *Proceedings of the 8th ACM Symposium on Operating Systems Principles, SOSP 1981*, vol. 15, no. 5, pp. 12–21, 1981.

[11] C. H. Kim, T. Kim, H. Choi, Z. Gu, B. Lee, X. Zhang, and D. Xu, “Securing Real-Time Microcontroller Systems through Customized Memory View Switching,” no. February, 2018.

[12] G. Bouffard and L. Gaspard, “Hardening a Java Card Virtual Machine Implementation with the MPU,” 2018.

[13] E. Baccelli, J. Doerr, S. Kikuchi, F. A. Padilla, K. Schleiser, and I. Thomas, “Scripting Over-The-Air: Towards Containers on Low-end Devices in the Internet of Things,” *2018 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2018*, no. March, pp. 504–507, 2018.

[14] OWASP, “Website of : Owasp,” https://owasp.org/www-project-internet-of-things/, 2020, [Online; accessed January 17, 2020].

[15] CVE, “Website of : Cve, linux kernel vulnerabilties,” https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=linux+kernel, 2020, [Online; accessed January 17, 2020].

[16] INRIA, “Website of : Coq,” https://coq.inria.fr, [Online; accessed January 17, 2020].

[17] IETF, “Website of : Ietf suit draft architecture,” https://tools.ietf.org/html/draft-ietf-suit-architecture, 2020, [Online; accessed January 17, 2020].

[18] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, T. Sewell, H. Tuch, and S. Winwood, “SeL4: Formal verification of an OS kernel,” *SOSP’09 - Proceedings of the 22nd ACM SIGOPS Symposium on Operating Systems Principles*, pp. 207–220, 2009.

[19] S. Lescuyer, “ProvenCore : Towards a Verified Isolation Micro-Kernel,” no. January, 2015.

[20] A. Reid, “Who guards the guards? formal validation of the Arm v8-m architecture specification,” *Proceedings of the ACM on Programming Languages*, vol. 1, no. OOPSLA, pp. 1–24, 2017.