Université de Rennes

Direction de la Recherche et de l'Innovation

Appel à projets - Année 2026

Contrats doctoraux «Ordinaires»

ECOLE DOCTORALE : MATISSE **POLE DE RECHERCHE : Mathématiques-numérique**

Unité de recherche d'accueil

Unité de Recherche : IRISA, UMR 6074

Equipe : DiverSE / Sushi

Encadrement

Directeur de thèse : Olivier ZENDRA **Unité de recherche** : UMR 6074

Employeur/Établissement d'origine : Inria

HDR : X oui **Nb de thèses en cours** : 4

Mail : Olivier.Zendra@irisa.fr **Tél**. : 0299847552

Co-directeur de thèse : Frédéric TRONEL **Unité de recherche** : UMR 6074

Employeur/Établissement d'origine : CentraleSupélec Rennes

HDR : X non **Nb de thèses en cours** : 4

Mail : Frederic.Tronel@irisa.fr **Tél.** : +33 2 99 84 45 78

Sujet

Titre : Interfaces hardening in mixed code operating system kernels (MkLISafe)

Mots clés : cybersecurity, operating systems, Linux, IOCTL, Rust, interface hardening, secure languages, programming rules

Cotutelle internationale (oui/non) : Non.

Descriptif (15-30 lignes) :
Contexte socioéconomique et scientifique ; Les hypothèses et questions posées ; Les grandes étapes de la thèse et démarche ; Approches méthodologiques et techniques envisagées

Résumé du projet
We propose to design and implement a hybrid hardening approach for software interfaces in legacy operating system kernels implemented with a mix of safe and unsafe programming languages. The approach is based on types in a memory-safe, strongly-typed programming language and fine-grained hardware compartmentalization features expected in near-future CPUs. The main use case considered is ioctl handlers written in C in the Linux kernel with the Rust safe programming language [1] and CHERI-enabled CPUs (ARM Morello or a RISC-V IP) [2]. The results should pave the way for secure compartmentalization [3] inside legacy operating system kernels.

Description des objectifs et originalité de la thèse
A major part of the Linux kernel attack surface is hidden behind the ioctl system call [4]. This system call is used by many drivers to expose custom services to userspace programs. However, the highly driver-dependent semantics of these services make it difficult to ensure the robustness of this interface. First, driver code is more likely to have vulnerabilities because of less peer-reviewing and scrutiny from the community than core kernel code. Second, the exposed interfaces and their semantics differ between drivers, mandating driver-specific expertise for manual reviewing. This variety makes it difficult to automate bug finding [5].
The Rust-for-Linux project [1] progressively integrates support for Rust-coded drivers in the mainline Linux kernel [6]. It aims at improving the quality of drivers, especially reducing the number of vulnerabilities. However, Rust-for-Linux is mostly effective at preventing memory safety bugs and does not address higher-level semantics-violations [7]. In the context of Rust-for-Linux, ongoing work in the SUSHI team has proposed wrapper types for driver developers to specify input data checks and prevent access to actual data before all checks are validated. Since the specifications on input data are declared as types in the source code, they can be close to the types of the driver's interface. Thus, they are more explicit than traditional checks spread in the call graph of the handlers. The design of these wrapper types extends and transposes to Rust the types introduced in RLBox for C++ [8]. The typestate design pattern [9] and its easy applicability with high assurance in safe Rust [10] are leveraged to let the Rust compiler guarantee that the specified checks are done.
Description des principaux verrous et techniques envisagées

Challenge 1: Identify patterns of input validation in the huge diversity of ioctl handlers. A comprehensive set of elementary checks should be identified as well as common sequences of checks and how they are spread in the call graph. The huge amount of ioctl handlers (thousands) makes it mandatory to automate the analysis, for example with taint analysis techniques and semantic pattern matching tools like Coccinelle [11].
Challenge 2: Interpose a mandatory Rust-based input-validation layer. Hardening ioctl handlers using wrapper types involves three sub-challenges. (i) Intermediate stages along the validation paths access already validated parts of the input, which requires identifying valid patterns of access to partially validated values in wrapper types. (ii) The interposition adaptation process should scale and thus be automated and guaranteed to not affect the correctness of the handlers. Interposition patterns could be written as Coccinelle semantic patches [11]. (iii) Using the input-validation layer should be enforced at build time by combining types and lightweight static analysis.
Challenge 3: Make the input-validation layer lightweight and robust despite mixed code. To keep the safety guarantees of the Rust compiler although Rust and C code are executed in the same address space of the kernel, ioctl handlers and the input-validation layer could be isolated from each other using fine-grained hardware compartmentalization. CHERI capabilities [2] could protect the integrity of the input-validation layer and prevent ioctl handlers from accessing unvalidated input. This isolation should be set automatically and add a negligible performance overhead.
Challenge 4: Make the input-validation layer user-friendly enough to promote adoption. The integration of the Rust-based wrappers of the input-validation layer into common software engineering practice will have to be evaluated. To favor easy adoption by developers, user friendliness must be kept in mind when designing the wrappers. Integration in the most used development environments [12] [13] will be explored.

Approche méthodologique et critères de qualité des résultats obtenus

To start addressing the four challenges early in the project, the problem could be first simplified in two ways before gradually adding automation: by only considering a small set of ioctl handlers and only handlers which check their input in a single place of their call graph.
The CHERI Alliance maintains mature CHERI-enabled versions of the Linux kernel [14] and the LLVM toolchain. However, combining Rust and CHERI is only work in progress [15] [16] [17]. This can be worked around using wrappers managing CHERI capability changes at the boundary between handlers and the input-validation layer. Evaluations requiring real execution can use QEMU/RISC-V as well as ARM Morello boards, one of which will be lent by the CHERI Alliance.
Evaluation criteria will consider performance impact, code quality metrics as well as past CVEs that the approach could have prevented. The results should be submitted in major operating system conferences like EuroSys, ASPLOS, ATC, OSDI, DSN and workshops like HotOS, APSys, KISV and PLOS, as well as security conferences like USENIX Security, NDSS, CCS, AsiaCCS, ACSAC, S&P, EuroS&P and workshops like SysTEX and EuroSec.

Etat de l'art s'appuyant sur un choix de publications importantes liées au sujet

Because of the huge size of the Linux kernel, static-analysis-based techniques to secure system calls must scale and thus be kept simple. The Linux kernel source tree features the ad-hoc sparse tool to annotate pointers to userspace memory and detect unwanted access to such memory [18]. Its use is, however, not mandatory, and it does not enforce any control on the actual input. Coccinelle has been used for years to automatically detect and often fix Linux bugs based on descriptions of C source code patterns and semantic patches [11]. Its potential use to enforce input validation from userspace has not been studied though.
The most popular dynamic-analysis techniques to secure the system call interface are based on fuzzing, typically with syzkaller [19]. However, most ioctl operate on device drivers and testing them requires the actual devices to be present or to have a model allowing to simulate the device.
DIFUZE [20] and later SyzDescribe [21] fuzz device drivers by first using static analysis to automate the description of ioctl data structures and API. Similar static analysis techniques could help interposing an input validation layer and thus secure ioctl handlers without requiring the hardware.
Ongoing work in the Rust-for-Linux project introduces an untrusted data API like RLBox [8] for drivers written in Rust [22]. Although this work is close to the basis of the wrapper types under work in the SUSHI team, it only targets Rust code and does not allow to specify the validations to be made beyond a single validation function to provide.

Liste des publications

[1] «Rust for Linux,» [En ligne]. Available: https://rust-for-linux.com. [Accès le November 2025].

[2] R. N. M. Watson, D. Chisnall, J. Clarke, B. Davis, N. W. Filardo, B. Laurie, S. W. Moore, P. G. Neumann, A. Richardson, P. Sewell, K. Witaszczyk et J. Woodruff, «CHERI: Hardware-Enabled C/C++ Memory Protection at Scale,» *IEEE Security & Privacy,* vol. 22, n° %14, p. 50–61, 2024.

[3] H. Lefeuvre, V.-A. Badoiu, Y. Chen, F. Huici, N. Dautenhahn et P. Olivier, «Assessing the Impact of Interface Vulnerabilities in Compartmentalized Software,» chez *30th Annual Network and Distributed System Security Symposium, NDSS 2023*, San Diego, California, USA, 2023.

[4] H.-W. Hung, Y. Liu et A. A. Sani, «Sifter: protecting security-critical kernel modules in Android through attack surface reduction,» chez *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking, MobiCom '22*, New York, NY, USA, 2022.

[5] A. Kurmus, A. Mambretti, A. Sorniotti, V. Lenders, D. Pfammatter et B. Tellenbach, «SoK: automating kernel vulnerability discovery and exploit generation,» chez *Proceedings of the 19th USENIX Conference on Offensive Technologies, WOOT '25*, USA, 2025.

[6] J. Corbet, «A first look at Rust in the 6.1 kernel,» *LWN.net,* 13 October 2022.

[7] Z. Li, V. Narayanan, X. Chen et J. B. A. Zhang, «Rust for Linux: Understanding the Security Impact of Rust in the Linux Kernel,» chez *2024 Annual Computer Security Applications Conference (ACSAC)*, 2024.

[8] S. Narayan, C. Disselkoen, T. Garfinkel, N. Froyd, E. Rahm, S. Lerner, H. Shacham et D. Stefan, «Retrofitting fine grain isolation in the Firefox renderer,» chez *Proceedings of the 29th USENIX Conference on Security Symposium*, USA, 2020.

[9] R. E. Strom et S. Yemini, «Typestate: A programming language concept for enhancing software reliability,» *IEEE Transactions on Software Engineering,* Vols. %1 sur %2SE-12, n° %11, pp. 157-171, 1986.

[10] J. Duarte et A. Ravara, «Taming stateful computations in Rust with typestates,» *Journal of Computer Languages,* vol. 72, n° %1101154, 2022.

[11] J. Lawall et G. Muller, «Coccinelle: 10 years of automated evolution in the Linux kernel,» chez *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference, USENIX ATC '18*, USA, 2018.

[12] «Theia,» [En ligne]. Available: https://theia-ide.org/. [Accès le November 2025].

[13] «Visual Studio Code,» [En ligne]. Available: https://code.visualstudio.com/. [Accès le November 2025].

[14] «Securing Linux® with CHERI,» [En ligne]. Available: https://cheri-linux.org/. [Accès le November 2025].

[15] «Rust + CHERI Working Group,» [En ligne]. Available: https://cheri-alliance.org/who-we-are/working-groups/rust-cheri/. [Accès le November 2025].

[16] L. Revill, «A Rusty CHERI - The path to hardware capabilities in Rust,» chez *FOSDEM 2023*, Brussels, 2023.

[17] S. Harris, S. Cooksey, M. Vollmer et M. Batty, «Rust for Morello: Always-On Memory Safety, Even in Unsafe Code (Experience Paper),» chez *37th European Conference on Object-Oriented Programming (ECOOP 2023)*, Seattle, WA, United States, 2023.

[18] N. Brown, «Sparse: a look under the hood,» *LWN.net,* 8 June 2016.

[19] «Research work based on syzkaller,» [En ligne]. Available: https://github.com/google/syzkaller/blob/master/docs/research.md. [Accès le November 2025].

[20] J. Corina, A. Machiry, C. Salls, Y. Shoshitaishvili, S. Hao, C. Kruegel et G. Vigna, «DIFUZE: Interface Aware Fuzzing for

Kernel Drivers,» chez *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*, New York, NY, USA, 2017.

[21] Y. Hao, G. Li, X. Zou, W. Chen, S. Zhu, Z. Qian et A. A. Sani, «SyzDescribe: Principled, Automated, Static Generation of Syscall Descriptions for Kernel Drivers,» chez *Yu Hao et al., "SyzDescribe: Principled, Automated, Static Generation of Syscall Descriptions for Kernel Drivers," 2023 IEEE Symposium on Security and Privacy (SP)*, San Francisco, CA, USA, 2023.

[22] D. Alden, «Tracking trust with Rust in the kernel,» *LWN.net,* 3 September 2025.