Sources

INBOX

https://www.cs.hmc.edu/%7Egeoff/classes/hmc.cs135.201001/homework/fuse/fuse_doc.html (from https://unix.stackexchange.com/questions/325473/in-fuse-how-do-i-get-th e-information-about-the-user-and-the-process-that-is-try)

- https://security.googleblog.com/2022/12/memory-safe-languages-in-android-13.html (from [1])

[2]

Difference:

- “safety concerns go beyond type systems”

Notes

Introduction

[2] “Even worse, pervasive use of pointer aliasing, pointer arithmetic, and unsafe type casts keeps modern systems beyond the reach of software verification tools.”

CVE data

[3], [4] (ch-04, “unsafe language”)

methodology:

- for now, only linux kernel CVEs (then we can filter by CPE). this one paper also primarily cites a CVE analysis targeting linux kernel. EXTRA also consider other sources.
- maybe, for now, just focus on the CVE categories and maybe pick out some examples. looking at 3500 CVEs would be overkill anyways.
- [3] says that, in the last 10 years, 3439/3918 == 87.8% of linux kernel CVEs are memory-related (which we would atleast partially solve with Rust)
- EXTRA overflows would be interesting, but require more investigation into Rusts overflow behavior.

beispiel-CVEs

sources:

- https://nvd.nist.gov/vuln/search#/nvd/home?cnaSourceIdList=386&sortOrder=5&sortDirection=2&offset=125&rowCount=25&keyword=filesystem&cpeFilterMode=applicability&cpeName=cpe:2.3:o:linux:linux_kernel:*:*:*:*:*:*:*:*&resultType=records

CVE	problem	solution	
2011-0699	unerwartete signed/unsignedness von operatoren führt zum overflow (und AFAICS zu negativen kmalloc sizes)	ich habe wrapper `Wrapping<Num>` und `Saturating<Num>`, und ich kann trivialerweise zB noch `Checked<Num>` bauen. damit gebe ich gleichzeitig das verhalten des systems vor (kein surprise overflow) und habe zusätzlich eine klare annotation ggü der programmierperson, welches verhalten auftreten wird.	🟢
2025-21646	the `procfs` filesystem (`procfs`) expects maximum path length of 255, this was overseen by the `afs` filesystem (`afs`) implementors, leading to a runtime error	if `procfs` API were implemented per my concept, maximum path length could be encoded in the type, so compiler could warn/error on oversight	🟢

`procfs`

maybe:

- https://nvd.nist.gov/vuln/detail/CVE-2025-21646
- https://nvd.nist.gov/vuln/detail/CVE-2026-23147
- ▸ problem:
  ▸ rust löst: :green

Implementation

The Rust Project Developers. 2017. Implementation of Rust stack unwinding. https://doc.rust-lang.org/1.3.0/std/rt/unwind/.

Bibliography

[1] L. Seidel and J. Beier, "Bringing Rust to Safety-Critical Systems in Space," in *2024 Security for Space Systems (3S)*, 2024, pp. 1–8. doi: 10.23919/3S60530.2024.10592287.

[2] A. Balasubramanian, M. S. Baranowski, A. Burtsev, A. Panda, Z. Rakamarić, and L. Ryzhyk, "System Programming in Rust: Beyond Safety," in *Proceedings of the 16th Workshop on Hot Topics in Operating Systems*, in HotOS '17. Whistler, BC, Canada: Association for Computing Machinery, 2017, pp. 156–161. doi: 10.1145/3102980.3103006.

[3] cvedetails.com, "CVEs on Linux Kernel Machines." Accessed: Feb. 15, 2026. [Online]. Available: https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33

[4] H. Chen, Y. Mao, X. Wang, D. Zhou, N. Zeldovich, and M. F. Kaashoek, "Linux kernel vulnerabilities: State-of-the-art defenses and open problems," in *Proceedings of the Second Asia-Pacific Workshop on Systems*, 2011, pp. 1–5.

`*afs*` – the `afs` filesystem: TODO 1

`*procfs*` – the `procfs` filesystem: TODO 1, 1, 1