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])
- https://blog.reverberate.org/2025/02/03/no-panic-rust.html

[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

maybe:

- https://nvd.nist.gov/vuln/detail/CVE-2024-46777 (overflow again)
- https://nvd.nist.gov/vuln/detail/CVE-2025-38415 (shift OOB)

CVE	problem	solution	
2011-0699	FIXME deutsch: 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	if \`procfs\` API were implemented per my concept, maximum path length could be	🟢

CVE	problem	solution	
	by the \`afs\` filesystem (\`afs\`) implementors, leading to a runtime error	encoded in the type, so compiler could warn/ error on oversight	
2024-55641	When quota reservation on XFS fails due to IO errors shutting down the filesystem, inodes were mistakenly not unlocked during cleanup.	Implementing inodes in Rust can make use of the `Drop` trait, automatically unlocking them as they go out of scope.[1]	🟢
2024-45003	Inside the the Virtual File System layer (VFS) layer, during inode eviction, a race condition can result in a deadlock when inodes that are marked for deletion get accessed by a filesystem.	Rust provides no builtin solution to race conditions. While data races are automatically prevented in safe Rust, preventing deadlocks still lies in the hands of the programmer.	🔴
2022-48869	A data race in the `gadgetfs` implementation can lead to use-after-free.	This is a classic example of a multithreaded program not using the correct synchronization primitives for shared mutable state. As long as safe Rust is used for access of this state — in a hypothetical Rust-only implementation —, data races are guaranteed by the language to not occur. (However, using unsafe Rust could limit this guarantee.)	🟢 /🟡
2024-42306	In the `udf` kernel module, when a corrupted block bitmap is detected, allocation is aborted but subsequent allocations will still not check the fail state and instead blindly use the allocation buffer, leading to undefined state. The solution was to use a "verified" bit to check the bitmap for validity.	This depends: if the "verified" bit existed before the issue was found, but was erroneously not checked, this can be prevented through stricter type modelling in Rust. However, if the case in question was simply not considered during implementation, and the mentioned bitflag was introduced as solution, then this constitutes a typical logic error where not all possible system states are considered, and cannot be prevented by our approach.	🟡
2024-46695	A root user on an Network File System (NFS) client can, under specific circumstances, change security labels on a mounted NFS file system. This happens because a mandatory permission check was overlooked, which was documented in the contract of the function `__vfs_setxattr_noperm()`.	Our approach would allow to enforce these permission checks as part of the type system, either by a type around `__vfs_setxattr_noperm()` performing them itself, or by only yielding the correct marker types when the permissions are checked.	🟢
2025-38663	A sanity check for invalid file types was missing from the `nilfs2` module.	As concretely shown within the `FileType` enum in our wrapper library, construction of the enum from an invalid file type ID would have automatically resulted in an error.	🟢
2023-52590	Due to an interaction with VFS, renaming a directory on `ocfs2` could result in filesystem corruption, because a lock was not properly aquired.	Although, in theory, this would be part of the VFS contract that we could try to encode in the type system, this looks like a logic error as result of a complex interaction of modules. This kind bugs are notoriously hard to avoid completely by software tooling, because they require a detailed high-level understanding of components and contracts that is usually very hard to express in a machine-understandable way.	🟡 /🔴
2024-50202	In `nilfs2`, errors were ignored in a procedure searching for directory entries. This could lead to a hang later on when the error are rediscovered.	Error handling is one of Rust's strengths, because they are wrapped into the return type and the language requires the programmer to give explicit instructions on how to react to the error case. This is different in C, where errors	🟢

[1]see also *RAII*

CVE	problem	solution	
		are usually hidden away in global state, or — while part of return value — there is no mechanism to force explicit handling.	
2024-47699	A potential null pointer dereference was found inside `nilfs` when dealing with a corrupted filesystem.	While unsafe Rust does not prevent accidental null pointer derefs, since we abstract away pointer access in our wrapper, and let the user deal only with native Rust owned values and references, this problem would be solved by The libfuse wrapper (Libfuse_wrapper)	🟢

\`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.

***NFS* – Network File System** 2, 2

***VFS* – the Virtual File System layer**: the part of the Linux kernel that abstracts between filesystems and the rest of the system 2, 2, 2

***\`afs\`* – the \`afs\` filesystem**: TODO 2

***libfuse_wrapper* – the libfuse wrapper**: TODO - our lil' project 3

***\`procfs\`* – the \`procfs\` filesystem**: TODO 1, 1, 3