

Toolchain security features status update



Kees Cook <kees@kernel.org>

Justin Stitt <justinstitt@google.com>

<https://outflux.net/slides/2026/lpc/features.pdf>

Flashback! 2025 security features review

	GCC	Clang	RustC
zero call-used registers	yes	yes	needed
structure layout randomization	plugin	yes	needed
stack protector guard location	arm64 arm32 riscv ppc	arm64 arm32 riscv ppc	N/A
forward edge CFI	CPU in progress	CPU inline hash	yes
backward edge CFI	CPU SCS:arm64 inline	CPU SCS:arm64 inline	SCS:arm64
FAM counted_by attribute	yes	yes	???
pointer counted_by attribute	yes	yes	???
unexpected integer wrapping	broken	in progress	exists
-fbounds-safety	needed	in progress	N/A

2026 security features review

	GCC	Clang	RustC	
zero call-used registers	yes	yes	needed	🦀
structure layout randomization	plugin	yes	needed	🦀
stack protector guard location	arm64 arm32 riscv ppc	arm64 arm32 riscv ppc	N/A	
forward edge CFI	CPU in review	CPU inline hash	yes (arity)	😬
backward edge CFI	CPU SCS:arm64 inline	CPU SCS:arm64 inline	SCS:arm64	😭
struct member counted_by	yes	yes	needed	🦀
↑ ↑	↑ ↑	↑ ↑	↑ ↑	
unexpected integer wrapping	broken	in progress	exists	😡
-fbounds-safety	needed	in progress	N/A	🙋

RustC status

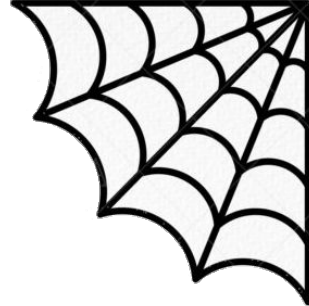
- With Rust in the Linux kernel, we need to keep RustC at parity with Clang and GCC so we avoid [cross-language attacks](#).
- Parity reached with C:
 - arm64 Software Shadow Call Stack
 - kCFI (needs `-Zmerge-functions=disabled`: [merged functions](#) with different hash type ids)
 - Software Tag-Based KASAN: `-Zsanitizer=kernel-hwaddress` (1.96+)
- Areas where Rust [hardening](#) needs attention:
 - zero call-used regs needs to happen in Rust code too
 - randstruct [in progress](#), so structs in Rust match those in C
 - kCFI arity: no FineIBT-BHI when Rust is enabled
 - counted_by: can Rust use this via bindgen?
 - arithmetic overflow handling exists, but how to wire up traps to use UBSan warning (not panic)
 - every hardening option is still `-Z` (unstable)

Parity: Miscellaneous

- Diagnostics for inscrutable warnings
 - GCC value range tracking would lead to mysterious warnings (e.g. `-Warray-bounds`) that provided no rationale about how a value had been reached:
 - `-fdiagnostics-show-context` ([16](#)+), Linux [v6.19](#)+
 - Clang inlining would not report from where a warning originated.
 - `-fdiagnostics-show-inlining-chain` ([23](#)+), Linux ([v7.1](#)+)
- `__builtin_counted_by_ref` works with `counted_by` with pointers
 - Implemented in Clang ([22](#)+)
 - Implemented in GCC ([16](#)+)
 - Used in Linux with new `kmalloc_obj_flex()` allocation helper ([v7.0](#)+)

Work needed: forward edge CFI (making progress?)

- Exploitation of func pointers easier than ever via automated gadget discovery
 - <https://www.usenix.org/conference/usenixsecurity19/presentation/wu-wei>
- CPU hardware support (coarse-grain: marked entry point matching): at parity
 - x86 ENDBR instruction, GCC & Clang (CONFIG_X86_KERNEL_IBT):
 - `-fcf-protection=branch`
 - arm64 BTI instruction, GCC & Clang (CONFIG_ARM64_BTI_KERNEL):
 - `-mbranch-protection=bti`
 - `__attribute__((target("branch-protection=bti")))`
- Software (per-function-prototype matching): `-fsanitize=kcfi`
 - Clang: existing: x86_64, arm64, riscv. new: [arm32](#) (22+)
 - GCC: [v16](#) under review (maybe for GCC 17?), covering x86_64, arm64, riscv, arm32



Work needed: backward edge CFI (no progress)

- CPU hardware support at parity
 - x86 Shadow Stack CPU feature bit and implicit operation: no compiler support needed
 - Userspace Shadow Stack working.
 - In-kernel Shadow Stack still not explored yet (and isn't expected).
 - arm64 PAC instructions, GCC and Clang (CONFIG_ARM64_PTR_AUTH_KERNEL):
 - `-mbranch-protection=pac-ret[+leaf]`
 - `__attribute__((target("branch-protection=pac-ret[+leaf]")))`
- Software (shadow stack)
 - x86: inline hash checking (like kCFI) **would be nice to have in both Clang and GCC**
 - arm64 shadow call stack: GCC ([12.1](#)+) and Clang (CONFIG_SHADOW_CALL_STACK):
 - `-fsanitize=shadow-call-stack`

Work needed: counted_by attribute for pointer members

```
struct {  
    struct info *p __attribute__((counted_by(count)));  
    int count;  
}
```

- Compiler parity reached in 2025
 - [Implemented](#) in GCC [16](#)+ (also with void *, [16](#)+) [1](#)
 - [Implemented](#) in Clang [21](#)+ (also with void *, [22](#)+) [2](#)
- Linux using `__counted_by_ptr()` as a distinct macro from `__counted_by()` to deal with the compiler version skew. ***Adoption is slow:***
 - **5** annotations in v7.0, **16** in v7.2, **33** in linux-next [3](#)
 - significantly harder to do bulk annotations since every allocation size needs an audit [4](#)

Work needed: unexpected arithmetic wrap-around

- For the Linux kernel, we need "[idiom exclusions](#)" to avoid instrumenting cases where wrap-around is either already checked, or is not part of program flow:
 - `if (var + offset < var)`
 - `while (var--)`
 - `-1UL, -2UL, ...`
 - Clang: [19+](#); GCC: **Needed**, for OBT below...
- "Overflow Behavior Type" annotations in kernel for explicitly declared overflow behavior: wrap or trap:
 - `typedef unsigned long __ob_trap size_t;`
 - Clang: [23+](#) ([-Xclang -fexperimental-overflow-behavior-types](#))
 - Linux adoption is stalled waiting on trap label support (Clang [debating](#) best approach)
 - GCC: **"Needed"**

Work needed: miscellaneous

- Handling nested structures ending in a Flexible Array Member
 - Fixed in GCC ([16](#)+)
 - **Needed** in Clang: <https://github.com/llvm/llvm-project/issues/72032>
- Coverage sanitizer stack depth tracking callback (kernel stack erasing)
 - GCC is still **plugin only** https://gcc.gnu.org/bugzilla/show_bug.cgi?id=121222
 - Implemented in Clang ([22](#)+)
- Partitioned allocations (Linux v7.2+ [CONFIG_KMALLOC_PARTITION_TYPED](#))
 - Clang: [-fsanitize=alloc-token](#) and [__builtin_infer_alloc_token\(\)](#) (22+)
 - GCC: **Needed**

Work needed: -fbounds-safety

- Clang features:
 - External bounds annotations: `__counted_by(N)`, `__sized_by(N)`, and `__ended_by(P)`
 - **Done:** for struct members
 - *Pending:* [for variables](#) and [for function parameters](#)
 - Pointers to a single object (i.e. pointer arithmetic is a compile time error):
 - Planned: `__single`
 - Internal bounds annotations (vectors/bi-directional pointers):
 - Planned: `__bidi_indexable` and `__indexable`
 - Sentinel-delimited arrays:
 - Planned: `__null_terminated` and `__terminated_by(T)`
 - Annotation for interoperating with bounds-unsafe code:
 - Planned: `__unsafe_indexable`
- GCC: **Needed**

See [Apple's LLVM Enforcing Bounds Safety in C RFC](#)

Questions / Comments ?

Thank you for your attention!

Kees Cook <kees@kernel.org>

Justin Stitt <justinstitt@google.com>

Bonus Slides...

Work needed: Link Time Optimization

- Toolchain support is at parity
 - GCC: `-flto`
 - Clang: `-flto` or `-flto=thin`
- Linux kernel support is only present with Clang, **GCC needed**
- No recent patches sent to LKML
- Latest development branch (against v6.1) appears to be Jiri Slaby's, continuing Andi Kleen's work:
 - <https://git.kernel.org/pub/scm/linux/kernel/git/jirislaby/linux.git/log/?h=lto>

Work needed: Spectre v1 mitigation

- GCC: wanted? no open bug...
- Clang:
 - `-mspeculative-load-hardening`
 - `__attribute__((speculative_load_hardening))`
 - <https://lvm.org/docs/SpeculativeLoadHardening.html>
- Performance impact is relatively high, but lower than using lfence everywhere.
- Really needs some kind of “reachability” logic to reduce overhead.

- Does anyone care about this?