Toolchain security features status update

Kees Cook <keescook@chromium.org>

Qing Zhao <qing.zhao@oracle.com>

Bill Wendling <morbo@google.com>

Justin Stitt <justinstitt@google.com>

https://outflux.net/slides/2025/lpc/features.pdf

Flashback! 2024 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 needed	CPU inline hash	in progress
backward edge CFI	CPU SCS:arm64 inline	CPU SCS:arm64 inline	SCS:arm64
FAM `counted_by` attribute	yes	yes	???
FAM in unions	yes	yes	N/A
unexpected integer wrapping	broken	broken	exists
`-fbounds-safety`	needed	in progress	N/A

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

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 (some tweaks still happening)

- 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
  - `counted_by` attribute needs to be investigated, can run enforce it too?
  - arithmetic overflow handling exists, but how to wire up traps consistently vs UBSan?

Parity reached: per-thread stack protector guard location

Arch	Linux Kernel Options	GCC	Clang
x86_64 & ia32	`-mstack-protector-guard-reg=fs` `-mstack-protector-guard-symbol=__stack_chk_guard`	yes (8.1+)	yes (16+)

arm64	`-mstack-protector-guard=sysreg`
	`-mstack-protector-guard-reg=sp_el0`
	`-mstack-protector-guard-offset=...TSK_STACK_CANARY...`
		yes (9.1+)	yes (14+)

arm32	`-mstack-protector-guard=tls` `-mstack-protector-guard-offset=...TSK_STACK_CANARY...`	yes (13.1+)	yes (15+)

riscv	`-mstack-protector-guard=tls`
	`-mstack-protector-guard-reg=tp`
	`-mstack-protector-guard-offset=...TSK_STACK_CANARY...`
		yes (12.1+)	yes (20+)

powerpc	`-mstack-protector-guard=tls` `-mstack-protector-guard-reg=r13`	yes (7.1+)	yes (20+)

Work needed: forward edge CFI (I got tired of waiting...)

- 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 (fine-grain: per-function-prototype matching)
  - Clang: inline hash checking: `-fsanitize=kcfi` (arm64 and x86_64)
  - GCC: **inline hash checking under review (v9)**

Work needed: backward edge CFI (no new 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`

Parity reached: `counted_by` attribute for pointer members

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

- Not just for Flexible Array Members any more!
  - Implemented in GCC 16+ (also with `void *`, 16+)
  - Implemented in Clang 21+ (also with `void *`, 22+)

- Annotation not yet being added to the Linux kernel, as there is some ongoing discussion since `counted_by` for FAM and `counted_by` for pointers happened in different compiler versions:
  - raise compiler version requirement?
  - different macro? `counted_by_ptr()`

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**

- Implement "Overflow Behavior Type" annotations in kernel for explicitly declared overflow behavior: `wrap` or `trap` so far, starting with `size_t`:
  - `typedef unsigned long __ob_trap size_t;`
  - Clang: 22?; 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
- `__builtin_counted_by_ref` works with `counted_by` with pointers
  - Implemented in Clang (22+)
  - *In progress* in GCC
- Clarify `-Warray-bounds` warnings (GCC only, Clang **lacks value tracking**)
  - `-fdiagnostics-show-context`
    - Implemented https://gcc.gnu.org/bugzilla/show_bug.cgi?id=109071
    - Used by Linux (v6.19+)
- 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+)

Work needed: `-fbounds-safety`

- Coordinate between Clang and GCC to implement Apple's bounds safety features:
  - Pointers to a single object: **`__single`**
    - Pointer arithmetic is a compile time error
  - External bounds annotations: **`__counted_by(N)`**, **`__sized_by(N)`**, and **`__ended_by(P)`**
  - Internal bounds annotations (vectors/bi-directional pointers): **`__bidi_indexable`** and **`__indexable`**
  - Sentinel-delimited arrays: **`__null_terminated`** and **`__terminated_by(T)`**
  - Annotation for interoperating with bounds-unsafe code: **`__unsafe_indexable`**

See Apple's LLVM Enforcing Bounds Safety in C RFC

Questions / Comments ?

Thank you for your attention!

Kees Cook <keescook@chromium.org>

Qing Zhao <qing.zhao@oracle.com>

Bill Wendling <morbo@google.com>

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
- No recent patches sent to LKML
- Latest development branch (against v5.19) 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://llvm.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?