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/2024/lpc/features.pdf

Flashback! 2023 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	needed
`-fstrict-flex-arrays`	yes	yes	yes
FAM `counted_by` attribute	in progress	in progress	???
FAM in unions	needed	needed	N/A
unexpected integer wrapping	broken	broken	exists

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
`-fstrict-flex-arrays`	yes	yes	yes
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

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 GCC & Clang:
  - arm64 Software Shadow Call Stack ready to land

- Areas where Rust hardening needs attention:
  - kCFI is in progress, and very close to ready
  - zero call-used regs needs to happen in Rust code too
  - randstruct needs to work with Rust or structs aren't ordered correctly
  - `counted_by` attribute needs to be investigated
  - arithmetic overflow handling exists, but how to wire up traps consistently vs UBSan?

Parity reached: `counted_by` attribute

- `counted_by` attribute for flexible array members
  - Implemented in GCC 15+.
  - Implemented in Clang 18+.

- Annotation is being added to the Linux kernel; currently have 391 instances, including some cases where we optionally enable it depending on machine endianness. This has already found some real bugs, but most of the reported warnings have been false positives due to incomplete updates required by the annotation (e.g. setting counter before accessing flex array).

Parity reached: flexible array members in unions

- Extended existing language extensions to support Flexible Array Members in unions (and alone in structs)
  - https://gcc.gnu.org/pipermail/gcc/2023-May/241426.html

```
union u {
    int foo;
    char bar[ 0 ];
};
```

  - Implemented in GCC 15+.
  - Implemented in Clang 18+.

Work needed: 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+)	in progress

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

Work needed: forward edge CFI (no new progress)

- 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")))`
    - GCC bug **still open**

- Software (fine-grain: per-function-prototype matching)
  - Clang: inline hash checking: `-fsanitize=kcfi` (arm64 and x86_64)
  - GCC: **inline hash checking still needed** (earlier arm64 effort needs more attention)
- Exploitation of func pointers easier than ever via automated gadget discovery
  - https://www.usenix.org/conference/usenixsecurity19/presentation/wu-wei

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
    - **Kernel support landed finally** (Shadow Stack systems available for 3 years now)!
    - In-kernel Shadow Stack still not explored yet.
  - 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` for general pointers

- Two types of arrays
  - Fixed-sized **bounds in TYPE**
  - Dynamically-sized
    - Variable-length array (VLA) **bounds in TYPE**
    - Flexible array member (FAM) **bounds in attribute**
    - Pointer offset **where are the bounds?**
- Related to the `-fbounds-safety` proposal (mentioned later…)

Work needed: `counted_by` for general pointers (GCC)

- Two major cases
  - Pointers inside structures
    - struct A { char *buf __attribute__((counted_by(size))); size_t size; }
  - Pointers passed as function arguments
    - void fill_array(int *p __attribute__((__counted_by(count))), size_t count)
    - This case is the same as the current attribute “access”:
      - __attribute__ ((access (read_write, 1, 2)))
      - void fill_array(int *p, size_t count);
- GCC will focus on the first case

Work needed: `__builtin_counted_by_ref`

- Provide access to counter via reference to flexible array (`__builtin_counted_by_ref`)
  - GCC & Clang: In progress

Allows for allocation wrappers that are agnostic of `counted_by` annotations:

```
#define allocate_flexible_object(P, FLEX, COUNT)  ({ \
    P = alloc(sizeof(*P) + sizeof(*(P)->FLEX) * (COUNT)); \
    typeof(__builtin_counted_by_ref((P)->FLEX)) count_ptr = \
        __builtin_counted_by_ref((P)->FLEX); \
    if (count_ptr) *count_ptr = COUNT; \
    P; })
```

Work needed: __builtin_counted_by_ref() in GCC

TYPE __builtin_counted_by_ref (PTR→FAM)

returns a pointer to the corresponding counted_by object for PTR→FAM if a counted_by object is associated with it through “counted_by” attribute. Otherwise, returns a void * NULL pointer.

struct A { size_t size; char buf[] __attribute__((counted_by(size))); } *obj1;

__builtin_counted_by_ref (obj1→buf) returns &obj1→size

struct B {size_t size; char buf[];} *obj2;

__builtin_counted_by_ref (obj2→buf) returns (void *)NULL;

Work needed: __builtin_counted_by_ref() in GCC (2)

- The first version was submitted on 8/13/2024:
  - https://gcc.gnu.org/pipermail/gcc-patches/2024-August/660267.html
  - A lot of discussion since then;
- The 2nd/3rd/4th versions were submitted on 9/10/2024:
  - https://gcc.gnu.org/pipermail/gcc-patches/2024-September/662723.html
  - Waiting for approval.

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 (i.e. "Rubenesque" 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

Work needed: other aspects of bounds checking

- Handling nested structures ending in a Flexible Array Member (Clang)
  - https://github.com/llvm/llvm-project/issues/72032
- Clarifying `-Warray-bounds` warnings (GCC)
  - https://gcc.gnu.org/bugzilla/show_bug.cgi?id=109071

Work needed: Clarifying -Warray-bounds warnings (GCC)

```
1 extern void warn(void);
2 #define MAX_ENTRIES     4
3 static inline void assign(int val, int *regs, int index)
4 {
5         if (index >= MAX_ENTRIES)
6                 warn();
7         *regs = val;
8 }
9 struct nums {int vals[MAX_ENTRIES];};
10
11 void sparx5_set(int *ptr, struct nums *sg, int index)
12 {
13         int *val = &sg→vals[index];   // where is the warning come from???
14
15         assign(0,    ptr, index);
16         assign(*val, ptr, index);
17 }
```

latest-gcc -O2 -Wall -c -o t.o t.c

```
t.c: In function ‘sparx5_set’:
t.c:13:29: warning: array subscript 4 is above array bounds of ‘int[4]’ [-Warray-bounds=]
   13 |         int *val = &sg->vals[index];
t.c:9:18: note: while referencing ‘vals’
    9 | struct nums {int vals[MAX_ENTRIES];};
```

Work needed: Clarifying -Warray-bounds warnings (GCC)

A new option -fdiagnostic-explain-harder: add more information in the warning message to report where such index value come from.

```
$ gcc -Wall -O2 -fdiagnostics-explain-harder -c -o t.o t.c
t.c: In function ‘sparx5_set’:
t.c:12:23: warning: array subscript 4 is above array bounds of ‘int[4]’ [-Warray-bounds=]
   12 |   int *val = &sg->vals[index];
      |              ~~~~~~~~^~~~~~~
  ‘sparx5_set’: events 1-2
    4 |   if (*index >= 4)
      |      ^
      |      |
      |      (1) when the condition is evaluated to true
......
   12 |   int *val = &sg->vals[index];
      |              ~~~~~~~~~~~~~~~~
      |                      |
      |                      (2) out of array bounds here
t.c:8:18: note: while referencing ‘vals’
    8 | struct nums {int vals[4];};
      |                  ^~~~
```

Work needed: Clarifying -Warray-bounds warnings (GCC)

The current status:

The 2nd version of the patch has been submitted to GCC upstream on 7/12/2024:

https://gcc.gnu.org/pipermail/gcc-patches/2024-July/657150.html

Waiting for approval.

Work needed: unexpected arithmetic wrap-around

- Technically working …
  - GCC & Clang: `-fsanitize={signed-integer-overflow,pointer-overflow}`
  - Clang: has; GCC: **Needed**: `-fsanitize=unsigned-integer-overflow`
- … but there are some significant behavioral caveats related to `-fwrapv` and `-fwrapv-pointer` (enabled via kernel's use of `-fno-strict-overflow`)
  - "It's not an undefined behavior to wrap around."
  - Clang: 19+; GCC: **Needed**
- 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**
- Type filtering support allows instrumentation to be toggled for specific types
  - Clang: 20?; GCC: **Needed**
- Add annotations in kernel for un*expected* wrap-around types (`size_t` first)
  - Clang: 20?; GCC: **Needed**

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?