Linux
Plumbers
Conference 2022

>> Dublin, Ireland / September 12-14, 2022

Where are we on security features?

Kees Cook <keescook@chromium.org>
Qing Zhao <qing.zhao@oracle.com>

https://outflux.net/slides/2022/lpc/features.pdf

Flashback! 2021’s features needing attention

	GCC	Clang
zero call-used registers	yes	no
structure layout randomization	plugin	no
stack protector guard location	arm64 arm32 riscv powerpc	arm64 arm32 riscv powerpc
forward edge CFI	CPU inline hash	CPU call table inline hash
backward edge CFI	CPU	CPU arm64 SCS
`-fstrict-flex-arrays`	no	no
element count attribute	no	no
integer overflow protection	broken	broken
assignment type introspection	no	no

2022: security feature review

	GCC	Clang
zero call-used registers	yes	yes
structure layout randomization	plugin	yes
stack protector guard location	arm64 arm32 riscv powerpc	arm64 arm32 riscv powerpc
forward edge CFI	CPU inline hash	CPU call table inline hash
backward edge CFI	CPU	CPU arm64 SCS
`-fstrict-flex-arrays`	in progress	workable
element count attribute	no	no
integer overflow protection	broken	broken
assignment type introspection	no	no

Parity reached: zero call-used registers

- `-fzero-call-used-regs`
  - Implemented in GCC 11.1+.
  - Implemented in Clang 15+.

- Linux kernel implements `CONFIG_ZERO_CALL_USED_REGS` with `-fzero-call-used-regs=used-gpr`
  - One kernel bug with paravirt outstanding, exposed by Clang

Parity reached: structure layout randomization

- Well, kinda: GCC support is via a plugin in the kernel tree.
- Implemented in Clang 15+:
  - `-frandomize-layout-seed-file=$(objtree)/scripts/basic/randstruct.seed`

- Linux Kernel enables option with:
  - `CONFIG_RANDSTRUCT_FULL`
  - `CONFIG_RANDSTRUCT_PERFORMANCE` (GCC only)

Work needed: stack protector guard location

Arch	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+)	**needed**
powerpc	-mstack-protector-guard=tls -mstack-protector-guard-reg=r13	yes (7.1+)	**needed**

Work needed: forward edge CFI

- 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")))`
    - Very recent GCC bug **under investigation**

- Software (fine-grain: per-function-prototype matching)
  - Clang:
    - Call tables: `-fsanitize=cfi` (currently used in kernel on arm64)
    - Inline hash checking: `-fsanitize=kcfi` (future for arm64 and x86_64)
  - GCC: **inline hash checking needed**
- Fine-grain is *really* needed for security to stop automated gadget exploitation
  - https://www.usenix.org/conference/usenixsecurity19/presentation/wu-wei

Work needed: backward edge CFI

- CPU hardware support at parity
  - x86 CET CPU feature bit and implicit operation: no compiler support needed
    - **Kernel support needed**; Linux hugely behind (CET systems available for 2 years now)!
    - Please, test the userspace series and review it.
    - In-kernel CET not even 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 **needed by both Clang and GCC**
  - arm64 shadow call stack: GCC (12.1+) and Clang (`CONFIG_SHADOW_CALL_STACK`):
    - `-fsanitize=shadow-call-stack`

Background: Proper flexible arrays (C99)

```
struct flexible {
    int foo;
    int bar;
    int array[];
} obj;

sizeof(obj.array) => *compile-time error*
__builtin_object_size(obj.array, 1) => -1
```

Background: 0-element (GNU extension) flexible arrays

```
struct gnu_extension {
    int foo;
    int bar;
    int array[0];
} obj;

sizeof(obj.array) => 0
__builtin_object_size(obj.array, 1) => -1
```

Background: 1-element (ancient) flexible arrays

```
struct ancient {
    int foo;
    int bar;
    int array[1];
} obj;

sizeof(obj.array) => 4
__builtin_object_size(obj.array, 1) => -1
```

Background: Fixed-size arrays

```
struct fixed_size {
    int foo;
    int array[10];
    int bar;
} obj;

sizeof(obj.array) => 40
__builtin_object_size(obj.array, 1) => 40
```

Background: Fixed-size trailing arrays

```
struct fixed_size {
    int foo;
    int bar;
    int array[10];
} obj;

sizeof(obj.array) => 40
__builtin_object_size(obj.array, 1) => -1
```

Background: N-element trailing flexible arrays (whoops)

```
struct sockaddr {
    unsigned char   sa_len;
    sa_family_t     sa_family;
    char            ss_data[14];
} obj;
#define SOCK_MAXADDRLEN 255   /* waaaaat */

sizeof(obj.ss_data) => 14
__builtin_object_size(obj.ss_data, 1) => -1
```

Work needed: treating Flexible Array Members strictly

- New option for **C/C++**: `-fstrict-flex-arrays[=N]`
- New attribute for **field_decl**: `strict_flex_arrays(N)`
  - Attribute can be used with or without option `-fstrict-flex-arrays`
  - Attribute has higher priority when both are present

N	Treated as FAM	Strictness	
0	[ ], [0], [1], [n]	Least strict	Default when option not present
1	[ ], [0], [1]		
2	[ ], [0]		
3	[ ]	Most strict	Default when option present without value

Strict Flexible Array Members (warnings)

Update `-Warray-bounds` to issue warnings for different levels of `-fstrict-flex-arrays[=N]`

N	`-Warray-bounds` issues warning for ?
0	none
1	[n]
2	[n], [1]
3	[n], [1], [0]

A new `-Wstrict-flex-arrays` is not needed

Strict Flexible Array Members (GCC plan)

Update all phases that handle FAMs with multiple levels;

Update warnings of `-Warray-bounds`, `-Wstringop-overflow`, `-Wstringop-overread`, etc., with multiple levels;

A new warning `-Wzero-length-array`; (Is this really needed?)

> to warn when zero-length arrays are used to discourage non-standard GCC extension. PR94428

Strict Flexible Array Members (current state)

GCC patches:

- Set 1: the new option and new attribute, control the analysis phase with multiple levels
- Set 2: update `-Warray-bounds`, `-Wstringop-overflow`, `-Wstringop-overread` with multiple levels
- Set 3: Add a new `-Wzero-length-array` warning (PR94428)

Set 1 patch is under review and revision: v1, v2, v3, and latest v4:

https://gcc.gnu.org/pipermail/gcc-patches/2022-September/601174.html

Clang 16+ has `-fstrict-flex-arrays=[0|1|2]` (and `-Wzero-length-array`) but **needs `-fstrict-flex-arrays=3`**

Work needed: bounds-checked Flexible Array Members

After finishing all work of `-fstrict-flex-arrays` to make the fixed array bounds more accurate;

Add a new attribute to annotate bounds of FAMs to enable flexible array bounds checking at runtime;

```
struct object {
    int items;
    int flex[] __attribte__((__element_count__(items)));
};
```

Use new attribute for array bounds check of flexible arrays (via `-fsanitize=bounds`) and so that `__builtin_dynamic_object_size()` can use it too (for FORTIFY_SOURCE).

Maybe also add a builtin for answering "does this object end with a flexible array?”

Work needed: arithmetic overflow protection

- Technically working …
    - GCC & Clang: `-fsanitize=signed-integer-overflow`
    - Clang: `-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."
- More than avoiding "undefined behavior", we want no "unexpected behavior".
    - Like run-time bounds checking, **need arithmetic overflow to be handled** as a trap or "warn and continue with wrapped value" *and* a way to optionally allow wrap-around.
    - It would be nice to have a "warn and continue with saturated value" mode instead, to reduce the chance of denial of service and reach normal error checking.
- Dare we invent a C exception handling mechanism?

Work needed: assignment type introspection

- `__builtin_lvalue_type()` for use in function-like assignments, type validation, type size checking, etc.

For example, given:

```
        struct something *p;

        p = kmalloc(sizeof(*p), gfp);
```

The definition of `kmalloc()` has no way to introspect the type its result is being assigned to. The following form would, but requires refactoring 33,000 callers:

```
        kmalloc(&p, gfp);
```

e.g.: assignment type and size introspection

```
#define kmalloc(size, gfp)    ({                                        \
    __builtin_lvalue_type() __ptr;                                      \
    if (size > sizeof(*__ptr))                                          \
        __ptr = _alloc_by_bucket_size(typeof(*__ptr), size, gfp);       \
    else                                                                \
        __ptr = _alloc_fixed_size(typeof(*__ptr), sizeof(*__ptr), gfp); \
    ptr;                                                                \
})
```

e.g.: assignment type verification (fancy `__must_check`)

```
#define __must_check_type(type, expr...)   ({           \
    BUILD_BUG_ON(                                       \
        !__same_type(__builtin_lvalue_type(), type));   \
    expr;                                               \
})

#define something(args...) __must_check_type(size_t, __something(args))

size_t okay = something(foo, bar);     /* ok: type of “okay” matches */
int truncated = something(foo, bar);   /* build failure */
```

Questions / Comments ?

Thank you for your attention!

Kees Cook <keescook@chromium.org>

Qing Zhao <qing.zhao@oracle.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?