Security Feature Parity: GCC and Clang

Linux Plumbers Conference 2019

Kees Cook <keescook@chromium.org>

https://outflux.net/slides/2019/lpc/gcc-clang.pdf

old school security feature examples

- stack canaries: `-fstack-protector-strong`
- uninitialized variable analysis: `-Wmaybe-uninitialized`
- format string safety analysis: `-Wformat-security`
- read-only relocations: `-Wl,-z,relro`
- immediate bindings: `-Wl,-z,bindnow`
- Position Independent Executable to use ASLR: `-Wl,-z,pie -fPIE`
- Variable Length Array analysis: `-Wvla`

overview of newer features

	gcc	clang
function sections	yes	yes
implicit fallthrough	yes	yes
Link Time Optimization	yes	yes
stack probing	yes	no
Spectre v1 mitigation	no	yes
caller-saved register wiping	patch	no
stack variable auto-initialization	plugin	yes
structure layout randomization	plugin	no
signed overflow protection	yes, usability issues	yes, usability issues
unsigned overflow protection	no	yes, usability issues
backward edge CFI	hardware only	hardware w/ arm64 soft
forward edge CFI	hardware only	yes

per-function sections

- `-ffunction-sections`
  - gcc: working!
  - clang: working!

- Supports fine-grain ASLR (randomize sections at kernel boot)

switch case fallthrough markings

- `-Wimplicit-fallthrough`
  - gcc: `__attribute__((fallthrough))` and parses comments too!
  - clang: `__attribute__((fallthrough))`

- Kernel now free of implicit fallthroughs
  - Looking through the roughly 500 patches just in the last year, about 10% of warnings were real bugs

Link Time Optimization

- gcc: `-flto`
- clang: `-flto` or `-flto=thin`

- Required for software CFI
- Lots of pain to update build tooling
- Questions about C memory model vs Kernel memory model

stack probing

- gcc: `-fstack-clash-protection`
- clang: needed

- Defense against giant VLAs/alloca()s
- Kernel removed all VLA usage, so this is mainly a concern for userspace.

Spectre v1 mitigation

- gcc: needed
- clang: `-mspeculative-load-hardening`
  https://llvm.org/docs/SpeculativeLoadHardening.html

- Performance impact is relatively high, but lower than using `lfence` everywhere.

zero caller-saved registers on func return

- gcc: patch only

  `-mzero-caller-saved-regs=used`

  https://github.com/clearlinux-pkgs/gcc/blob/master/zero-regs-gcc8.patch
- clang: needed

- Virtually no performance impact (xor is highly pipelined), and makes sure no leftover register contents can be used for speculation-style attacks.

stack variable auto-initialization

- gcc: plugin only
  https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/scripts/gcc-plugins/structleak_plugin.c
- clang: `-ftrivial-auto-var-init=pattern` (needs `...=zero`)

- Linus wants to be able to depend on zeroing in the kernel

structure layout randomization

- `__attribute__((randomize_layout))`
  - gcc: plugin only https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/scripts/gcc-plugins/randomize_layout_plugin.c
  - clang: stalled https://reviews.llvm.org/D59254

- Fun for really paranoid builds

signed overflow protection

- `-fsanitize=signed-integer-overflow`
  - gcc: working!
  - clang: working!

- Available handling modes need improvement (e.g. 6% object size increase just from the warning text additions). Better to have a user-defined handler.
- Would be nice to have a “warn and continue with saturated value” mode instead of either “die” or “warn and continue with wrapped value”.

unsigned overflow detection

- `-fsanitize=unsigned-integer-overflow`
  - gcc: needed
  - clang: working!

- This one isn’t technically “undefined behavior”, but it certainly leads to exploitable conditions.
- Same thoughts as signed overflow:
  - Available handling modes need improvement (e.g. 6% object size increase just from the warning text additions). Better to have a user-defined handler.
  - Would be nice to have a “warn and continue with saturated value” mode instead of either “die” or “warn and continue with wrapped value”.

CFI (backward edge: returns)

- hardware
  - x86: CET feature bit
    - no compiler support needed!
  - arm64: PAC instructions
    - gcc: `-mbranch-protection=pac-ret`
    - clang: `-mbranch-protection=pac-ret`
      - needs function attribute to disable branch-protection
- software shadow stack
  - clang: `-fsanitize=shadow-call-stack` on arm64 only (x86: wait for CET?)
  - gcc: needed

CFI (forward edge: indirect calls)

- hardware (coarse-grain: entry points)
  - x86: ENDBR instruction
    - gcc: `-fcf-protection=branch`
    - clang: `-fcf-protection=branch`
  - arm64: BTI instruction
    - gcc: `-mbranch-protection=bti`
    - clang: `-mbranch-protection=bti`
      - needs function attribute to disable branch-protection
- software (fine-grain: per-function-prototype)
  - clang: `-fsanitize=cfi`
- We *really* need fine-grain forward edge CFI: stops automated gadget exploitation
  - https://www.usenix.org/conference/usenixsecurity19/presentation/wu-wei