New and Upcoming Hardening Features in GCC

Alexandre Oliva

oliva@gnu.org, oliva@adacore.com

AdaCore

https://www.fsfla.org/~lxoliva/

GNU Tools Cauldron, September, 2022

Summary

- Context
- Register scrubbing
- Stack scrubbing
- Hardened Conditionals
- Hardened Booleans
- Control Flow Redundancy
- Testing challenges

Context

- AdaCore partnership with big customer
- Hardening against Sw & Hw Glitches
- Focus on Ada and C for specific targets
- Aiming at language and machine independence

Register scrubbing

- Oracle beat us to it (thanks Qing Zhao)
- Suggested a machine-independent approach
- -fzero-call-used-regs in GCC 11
  - Also available as function attribute
- Insert register-zeroing insns before every return
  - Target hook for unusual regs
- used or all; gpr; arg

Stack scrubbing

- (Not) Embecosm's x86*/RISC-V stack_erase
  - -fstrub=strict → -fstrub=relaxed
- Proposed for GCC 12, refreshed for GCC 13
- strub attribute for code and data **types**
- Zero callee's frame after return or exception
- Signature and code changes in late IPA pass

Stack scrubbing: at-calls

```
void __attribute__((strub))
fn (int arg) {



  /* Strub body */
}
...

    fn (0);
```

```
void __attribute__((strub ("at-calls")))
fn (int arg, void **wmp) {
 /* ↑ Strub interface ↑ */
  __strub_update (wmp);
  /* Strub body */
}
... void *wm;
    __strub_enter (&wm);
    try { fn (0, &wm); }
    finally { __strub_leave (&wm); }
```

Stack scrubbing: internal

```
int /* strub ("internal") */
fn (big_t arg, ...) {
```

```
  int __attribute__((strub)) var;
```

```
int __attribute__((strub (wrapper)))
fn (big_t arg, ...) {
  va_list vl; void *wm; va_start (vl, arg);
  __strub_enter (&wm);
  int ret = fn.w (&arg, &wm, &vl);
  finally: __strub_leave (&wm);
  va_end (vl); return ret;
}
int __attribute__((strub (wrapped)))
fn.w (big_t*arg, void**wmp, va_list*vl)
{ __strub_update (wmp);
  int __attribute__((strub)) var;
```

Stack scrubbing: builtin inlining

__strub_enter(&wm);	-O1+sg	wm = &wmp ? *wmp : sp;

__strub_update(wmp);	-O2+s	if (*wmp ⋛ sp ∓ RZS) {
		*wmp = sp ∓ RZS;
		if (&iwmp && *iwmp ⋛ *wmp)
		*iwmp = *wmp; }

__strub_leave(&wm);	-O2	if (wm ⋛ (&wmp ? *wmp : sp)) __strub_leave(&wm);

	-O3	for (void **e↑ = wm,
		**↓p = &wmp ? *wmp : sp;
		p < e; p++) *p = 0;

- **wmp** on to nested(-O3/s)/tail(-O2/must) calls

Stack scrubbing: details

- Early pass assigns modes, checks requirements
  - strict mode: callable, disabled
- Also __strub_update after alloca, not other calls
- Don't split strub nor inline in nonstrub (retry?)
- Don't omit sp restore before __strub_leave
- Optional fixed-size zeroing after __strub_leave?
- Testing: -fstrub={all,at-calls,internal}

Hardened Conditionals

- -fharden-{conditional-branches,compares} (12)
- Trap if reversed compare fails to confirm result
- Catches power-deprived–processor glitches
- Prevent optimization with asm hidden copies

Hardened Conditionals

- Very late gimple passes; scalar types

if (x op y) {	if (x op y) {
	if (x' **!op** y') trap();
/* then */ }	/* then */ }
else {	else {
	if (x' !op y') ; **else** trap();
/* else */ }	/* else */ }

- Vector compares not supported (yet?):

$z = x$ op y;	$z = x$ op y;
	$z' = x'$ **!op** y';
	if ($z == z'$) trap();

Hardened Booleans

- Increase false/true hamming distance
- Trap on neither, catches RAM/CPU glitches
- Ada in for GCC 13, C proposed
  - Ada attribute: validity checking at use points

```
type HBool is new Boolean;
--  Enumeration: (False,      True)
for HBool use (16#5a#, 16#a5#);
for HBool'Size use 8;
pragma Machine_Attribute (HBool, "hardbool");
```

Hardened Booleans for C

- Modeled after Ada, proposed for GCC 13
- hardbool(false = 0, true = ~false) attribute
- Modifier to underlying integral base type
- Decays to _Bool, converts from/through _Bool

```
typedef char __attribute__ ((__hardbool__ (0x5a))) hbool;
hbool first = 0;          /* False, stored as (char)0x5a. */
hbool second = !first; /* True, stored as ~(char)0x5a. */
static hbool zeroinit;  /* False, stored as (char)0x5a. */
auto hbool uninit;      /* Unknown, may trap if read. */
```

Control Flow Redundancy

- -fharden-control-flow-redundancy for GCC 13?
- Catches CFG-incompatible executions
- Blocks set bits in auto basic block bitmap

Control Flow Redundancy

{	{
	word visited/*[1]*/ = 0;
	visited |= 1;
if (x)	if (x) {
	visited |= 2;
g ();	g ();
	}
	visited |= 4;
	__hardcfr_check (3, &visited, &.cfg);
return y;	return y;
}	}

Control Flow Redundancy

- At edges to the exit block (returning stmts):
  - Every set block must have a set predecessor
  - Every set block must have a set successor
- -f**no-**hardcfr-check-exceptions: cleanups
- -f**no-**hardcfr-check-returning-calls: sibcalls
- -fhardcfr-check-noreturn-calls=\
  {always|**not-always**|nothrow|never}
- Exposing expected_throw as an attribute?

Control Flow Redundancy

- Inline testing for single exit, few (16) blocks
- Params: max blocks for inlining and checking
- Runtime CFG for out-of-line testing (libgcc):
  - ..._check(size_t blks, word*vst, word*cfg)
  - Array of 28+-bit (32) words (4G blocks)
  - $\forall$ non-fixed block, pred and succ lists:
    - (bitmask, index into vst)..., (0)
  - Entry and exit as block's own bit

Testing challenges

- Can't trigger hardware glitches
- Enabling features for bootstrap
- Adding options for better coverage
- Detecting scrubbed stack after return
- Hand-crafting tests, matching dump logs

Thank you!

oliva@gnu.org, oliva@adacore.com

https://www.fsfla.org/~lxoliva/

Questions?