New and Upcoming Hardening Features in GCC

Alexandre Oliva

oliva@gnu.org, oliva@adacore.com

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

GNU Tools Cauldron, September, 2022

Copyright 2022 AdaCore (last changed in September 2022)
This work is licensed under the Creative Commons BY-SA 4.0 International License.
https://www.fsfla.org/svn/fsfla/ikiwiki/blogs/lxolive/pres/hardening/
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

<table>
<thead>
<tr>
<th>void <strong>attribute</strong>((strub)) fn (int arg) {</th>
<th>void <strong>attribute</strong>((strub (&quot;at-calls&quot;))) fn (int arg, void **wmp) {</th>
</tr>
</thead>
<tbody>
<tr>
<td>/* Strub body */</td>
<td>/* ↑ Strub <strong>interface</strong> ↑ */</td>
</tr>
<tr>
<td>}</td>
<td>__strub_update (wmp);</td>
</tr>
<tr>
<td>...</td>
<td>/* Strub <strong>body</strong> */</td>
</tr>
</tbody>
</table>
| fn (0); | }

... void *wm;

__strub_enter (&wm);

try { fn (0, &wm); } finally { __strub_leave (&wm); }
## Stack scrubbing: internal

```c
int /* strub ("internal") */
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)) var;
```

```c
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

<table>
<thead>
<tr>
<th>Function</th>
<th>Optimization Level</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__strub_enter(&amp;wm);</code></td>
<td>-O1+sg</td>
<td><code>wm = &amp;wmp ? *wmp : sp;</code></td>
</tr>
</tbody>
</table>
| `__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

```c
if (x op y) {
    /* then */
} else {
    /* else */
}
```

- Vector compares not supported (yet?):

```c
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

```ada
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`

```c
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

- `f-harden-control-flow-redundancy` for GCC 13?
- Catches CFG-incompatible executions
- Blocks set bits in auto basic block bitmap
Control Flow Redundancy

```ada
{ word visited/*[1]*/ = 0;
    visited |:= 1;
    if (x) {
        visited |:= 2;
        g ();
    }
    visited |:= 4;
    _hardcfr_check (3, &visited, &.cfg);
    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
• -fnno-hardcfr-check-exceptions: cleanups
• -fnno-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)
  - ∀ 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

AdaCore

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

Questions?