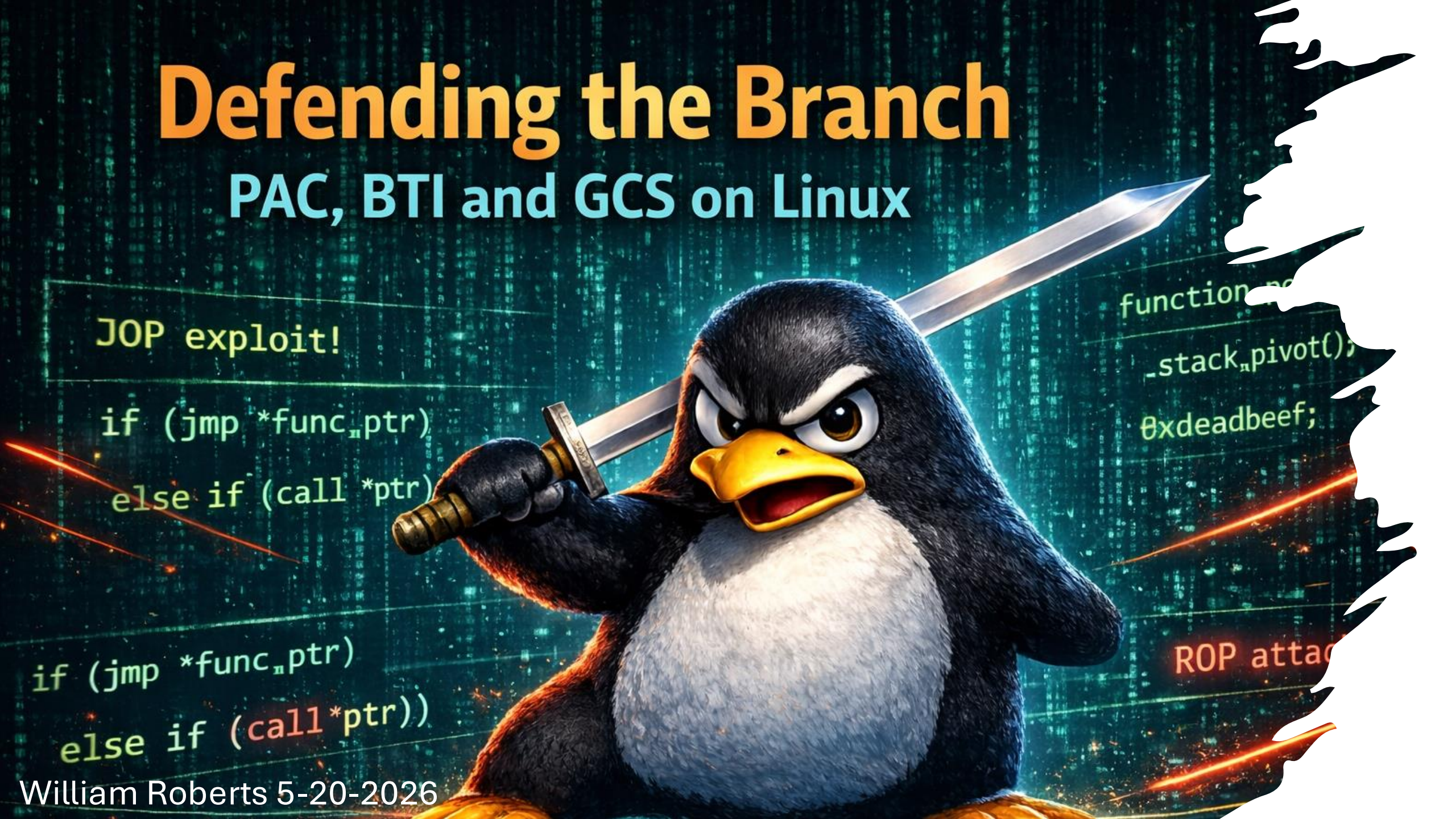
Defending the Branch

PAC, BTI and GCS on Linux



William Roberts 5-20-2026

What is BTI?

- Branch Target Instructions.
- Protects control flow changes.
  - First instruction after indirect branch must be a BTI instruction.
  - Called the landing pad.
- Reduces gadget count by forcing entry points.
- Instruction mnemonic: bti.
  - Takes a j, c or jc argument to mark a valid landing pad for jump, call or either.
- Throws illegal instruction.
- Arm v8.5 and up.
- Linux kernel 5.8+.
- /proc/cpuinfo flag “bti”.

What is PAC?

- Pointer Authentication Codes.
  - Sign and verify control flow pointers.
- Binaries are instrumented to sign and verify branch targets.
- Pointer modifications detectable.
  - Throws illegal instruction.
  - Stack based LR corruption prevented.
- Can use 2 different keys, A key or the B key.
- Uses QARMA crypto.
- Arm v8.3-A and up.
- Linux Kernel version 5.0+
- /proc/cpuinfo flags “paca” and “pacg”.

What is GCS?

- Guarded Control Stack.
- Hardware controlled Call and Return Stack.
- Corruption of the return addresses on the “normal” stack are detected.
- Arm v9.4-A and up.
- Linux Kernel version 6.13+.
- /proc/cpuinfo flags “gcs”.

Building with PAC/BTI/GCS* Support

Example C Program

```c
#include <stdio.h>

int foo(int x) {
    printf("BILL: %d\n", x);
    return x;
}

int main() {

    return foo(42);
}

```

```
$gcc -o go -mbranch-protection=standard a.c
$readelf -n go 2>/dev/null | head -n5

Displaying notes found in: .note.gnu.property
  Owner                Data size        Description
  GNU                  0x00000010       NT_GNU_PROPERTY_TYPE_0
      Properties: AArch64 feature: BTI, PAC, GCS
```

gcc/clang compiler option: -mbranch-protection=standard

Enables PAC, BTI and GCS. Other fine-grained options available.

PAC and BTI are orthogonal features but can work together. Some PAC instructions can be BTI landing pads.

* GCS requires recent versions of compilers, eg. GCC 15+

Notes on GNU Notes

- The GNU Notes section MUST be instrumented to declare BTI and GCS.
  - Specifically, ELF/Linux, other systems are different.
- The linkers and loaders look at this field in supported runtimes to determine memory protections.
  - BTI - mmap/mprotect loaders: PROT_BTI.
  - GCS – prctl call in libc.
  - Both - If one object file doesn't support it, linker disables it for resulting output.
- Force Linker to error on BTI support missing in object file.
  - `-Wl,-zforce-bti,--fatal-warnings`
  - Example: `gcc -Wl,-zforce-bti,--fatal-warnings …`
- PAC bit is nice for auditing but not required.
  - PAC is callee ABI, so code can be intermingled with non-pac aware stuff for the most part.
  - PAC within exception handling needs to be supported however, more on this later.

Projects with Assembler

- C/C++ projects just work when enabled in compiler options.
- However, handwritten assembly can pose issues.
  - It needs to be augmented to support PAC, GCS and BTI
  - It needs to be augmented to declare support in GNU notes.
- May need to add DWARF support for exception handling.
- Some projects use a non-standard ABI within their assembly.
  - This may make things slightly more difficult.

Assembly BTI Example

C Code

```c
#include <setjmp.h>
#include <stdio.h>

static jmp_buf jbuf;

static int foo(int x) {
    printf("foo: %d\n", x);
    if (x > 1) longjmp(jbuf, 42);
    return x;
}

int main(int argc, char *argv[]) {

    int r = setjmp(jbuf);
    if(r) {
        return r;
    }

    return foo(argc);
}

```

Objdump

```
0000000000000000 <foo>:
   0:   d503245f    bti c
   4:   a9be7bfd    stp x29, x30, [sp, #-32]!
   8:   910003fd    mov x29, sp
   c:   b9001fe0    str w0, [sp, #28]
```

```
0000000000000048 <main>:
  48:   d503245f    bti c
  4c:   a9bd7bfd    stp x29, x30, [sp, #-48]!
```

```
  60:   91000000    add x0, x0, #0x0
  64:   94000000    bl  0 <_setjmp>
  68:   d503249f    bti j
```

Assembly PAC Example

C Code

```c
#include <setjmp.h>
#include <stdio.h>

static jmp_buf jbuf;

static int foo(int x) {
    printf("foo: %d\n", x);
    if (x > 1) longjmp(jbuf, 42);
    return x;
}

int main(int argc, char *argv[]) {

    int r = setjmp(jbuf);
    if(r) {
        return r;
    }

    return foo(argc);
}

```

Objdump

```
0000000000000000 <foo>:
   0:   d503233f    paciasp
   4:   a9be7bfd    stp x29, x30, [sp, #-32]!
   8:   910003fd    mov x29, sp

  40:   a8c27bfd    ldp x29, x30, [sp], #32
  44:   d50323bf    autiasp
  48:   d65f03c0    ret

000000000000004c <main>:
  4c:   d503233f    paciasp
  50:   a9bd7bfd    stp x29, x30, [sp, #-48]!

  90:   a8c37bfd    ldp x29, x30, [sp], #48
  94:   d50323bf    autiasp
  98:   d65f03c0    ret
```

Assembly with PAC and BTI

C Code

```c
#include <setjmp.h>
#include <stdio.h>

static jmp_buf jbuf;

static int foo(int x) {
    printf("foo: %d\n", x);
    if (x > 1) longjmp(jbuf, 42);
    return x;
}

int main(int argc, char *argv[]) {

    int r = setjmp(jbuf);
    if(r) {
        return r;
    }

    return foo(argc);
}
```

Objdump

```
 7 0000000000000000 <foo>:
 8    0:   d503233f    paciasp

25   44:   d50323bf    autiasp
26   48:   d65f03c0    ret


28 000000000000004c <main>:
29   4c:   d503233f    paciasp

36   68:   94000000    bl  0 <_setjmp>
37   6c:   d503249f    bti j

47   94:   d50323bf    autiasp
48   98:   d65f03c0    ret
```

Note: paciasp is
A BTI landing pad

Hint Space

- Arm Supports hint instructions
- Hint instructions are NOP'd on hardware that doesn't support them.
- Let's you run binaries that target new features on older hardware.
  - backwards compatible.
- PAC and BTI have hint instructions
  - Linux uses them for backwards compat.

hint instructions

hint	instruction
hint 25	paciasp
hint 29	autiasp
hint 27	pacibsp
hint 31	autibsp
hint 34	bti c
hint 36	bti j
hint 38	bti jc
hint 12	autia1716
hint 14	autib1716

ARM Manual: https://developer.arm.com/documentation/ddi0487/latest/

DWARF and CFI

- Stands for “Call Frame Information”.
- Let’s one *program* the DWARF information.
  - *Debugging With Arbitrary Record Formats*.
- DWARF defines a Turing complete virtual machine.
  - Yep, you can exploit it :-p.
- You execute some DWARF instructions to resolve the data you need.
- Most of the specs are incomprehensible unless you’re well versed in it.
  - In typical fashion, I was not.

Linux PAC ABI

- Callee function:
  - Signs the Link Register (x30/LR).
  - Stores the LR and Frame Pointer (FP/x29).
  - Does whatever.
  - Loads LR/FP.
  - Verifies the LR.
- Linux uses hint space compatible instructions: autiasp and paciasp.
  - Backwards compatible.
- Can use B key variants.
- Also works with –fomit-frame-pointer.

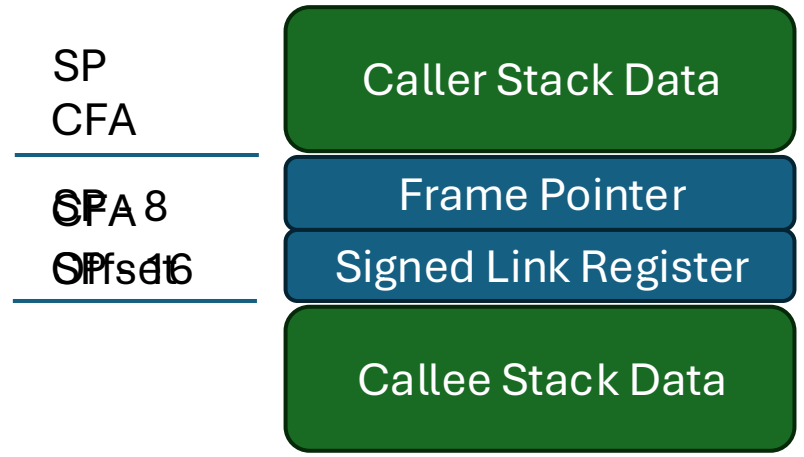
Example C Program Asm

```
12 foo:
13 .LFB0:
14     .cfi_startproc
15     hint    25 // paciasp
16     .cfi_window_save
17     stp x29, x30, [sp, -16]!
18     .cfi_def_cfa_offset 16
19     .cfi_offset 29, -16
20     .cfi_offset 30, -8
21     mov x29, sp
22     adrp    x0, .LC0
23     add x0, x0, :lo12:.LC0
24     bl  puts
25     mov w0, 42
26     ldp x29, x30, [sp], 16
27     .cfi_restore 30
28     .cfi_restore 29
29     .cfi_def_cfa_offset 0
30     hint    29 // autiasp
31     .cfi_window_save
32     ret
33     .cfi_endproc
```

CFI Explained

- Starts the frame record
- Indicates LR is signed
- Offsets the CFA to sp + 16
- Updates where LR/FP are relative to CFA
- Restores original rules
- Restores CFA to SP base
- Negates LR signed state
- Ends record

```
12 foo:
13 .LFB0:
14     .cfi_startproc
15     hint    25 // paciasp
16     .cfi_window_save
17     stp x29, x30, [sp, -16]!
18     .cfi_def_cfa_offset 16
19     .cfi_offset 29, -16
20     .cfi_offset 30, -8
21     mov x29, sp
22     adrp    x0, .LC0
23     add x0, x0, :lo12:.LC0
24     bl  puts
25     mov w0, 42
26     ldp x29, x30, [sp], 16
27     .cfi_restore 30
28     .cfi_restore 29
29     .cfi_def_cfa_offset 0
30     hint    29 // autiasp
31     .cfi_window_save
32     ret
33     .cfi_endproc
```



CFA – Canonical Frame Address

DWARF and Common Information Entry

- Use readelf to see DWARF
  - read .eh_frame section for raw bytecode.
  - Use debug-dump for disassembled bytecode.
- CIE is common for all frames
  - Think of it like default arguments to CFI data unless explicitly changed.
  - The CFA IS THE STACK POINTER.
  - CIEs have FDEs (next slide).
- DWARF defines .debug_<name> however this is .eh_layout.
  - No spec for it :-p.

```
$readelf --debug-dump=frames go
Contents of the .eh_frame section:


00000000 0000000000000010 00000000 CIE
  Version:               1
  Augmentation:          "zR"
  Code alignment factor: 4
  Data alignment factor: -8
  Return address column: 30
  Augmentation data:     1b
  DW_CFA_def_cfa: r31 (sp) ofs 0
```

DWARF and Frame Definition Entries

```
000000b8 0000000000000020 000000bc FDE cie=00000000 pc=00000000004101e8..0000000000410210
  DW_CFA_advance_loc: 4 to 00000000004101ec
  DW_CFA_AARCH64_negate_ra_state
  DW_CFA_advance_loc: 4 to 00000000004101f0
  DW_CFA_def_cfa_offset: 16
  DW_CFA_offset: r29 (x29) at cfa-16
  DW_CFA_offset: r30 (x30) at cfa-8
  DW_CFA_advance_loc: 24 to 0000000000410208
  DW_CFA_restore: r30 (x30)
  DW_CFA_restore: r29 (x29)
  DW_CFA_def_cfa_offset: 0
  DW_CFA_advance_loc: 4 to 000000000041020c
  DW_CFA_AARCH64_negate_ra_state
  DW_CFA_nop
```

- Each function gets a FDE associated with a range for the PC.
- FDEs are programs combined with CIE to generate proper state for unwinding.
- Use addr2line to correlate FDEs to functions.

FDE to Function

- Probably better ways.
- I use addr2line with base address for FDE window.

```
$addr2line -f -e go 0x4101e8
foo
/home/bill/workspace/a.c:3
```

Unwinding

```
55    static inline void *
56    aarch64_demangle_return_addr (struct _Unwind_Context *context,
57                                  _Unwind_FrameState *fs,
58                                  _Unwind_Word addr_word)
59    {
60      void *addr = (void *)addr_word;
61      const int reg = DWARF_REGNUM_AARCH64_RA_STATE;
62
63      if (fs->regs.how[reg] == REG_UNSAVED)
64        return addr;
65
66      /* Return-address signing state is toggled by DW_CFA_GNU_window_save (where
67         REG_UNSAVED/REG_UNSAVED_ARCHEXT means RA signing is disabled/enabled),
68         or set by a DW_CFA_expression.  */
69      if (fs->regs.how[reg] == REG_UNSAVED_ARCHEXT
70          || (_Unwind_GetGR (context, reg) & 0x1) != 0)
71        {
72          _Unwind_Word salt = (_Unwind_Word) context->cfa;
73          if (aarch64_cie_signed_with_b_key (context) != 0)
74            return __builtin_aarch64_autib1716 (addr, salt);
75          return __builtin_aarch64_autia1716 (addr, salt);
76        }
77
78      return addr;
79    }
80
```

This is libgcc, but other unwinders will be similar, like llvm-libunwinder and libunwinder

auti1716 – Verifying PACs

- Is within the hint space, so NOPs on older machines.
- For invalid signatures, depending on version will produce one of:
  - Invalid address that will page fault.
  - FEAT_PAUTH2: Raise exception “Illegal Instruction”.
- Why default to x17 and x16 registers?
  - Intra-Procedural corruptible registers.
  - Simple terms: registers useable by unwinders.
- A or B signing key is in CIE Augmentation, can be intermixed.

Arm CFI Directives

CFI Directive	Purpose
.cfi_b_key_frame	Mark CIE for B signing key.
.cfi_negate_ra_state .cfi_window_save (deprecated)	Mark FDE as signed LR pointer. Opcode: DW_CFA_AARCH64_negate_ra_state.

-mbranch-protection options

option	effect
none	disabled
bti	Enables bti
pac-ret	Enables PAC for return addresses using A key
leaf	Extend PAC to include leaf functions
b-key	Use the B key for PAC
gcs	Enable GCS
standard	pac-ret+bti+gcs

Note: options can be joined with the + operator. For example: –mbranch-protection=“pac-ret+b-key+leaf+bti”.

Setting the GNU Notes Bits

```
#   define GNU_PROPERTY_AARCH64_BTI (1 << 0)   /* Has Branch Target Identification */
#   define GNU_PROPERTY_AARCH64_POINTER_AUTH \
    (1 << 1)                                   /* Has Pointer Authentication */

#  if GNU_PROPERTY_AARCH64_POINTER_AUTH != 0 || GNU_PROPERTY_AARCH64_BTI != 0
    .pushsection .note.gnu.property, "a";
    .balign 8;
    .long 4;
    .long 0x10;
    .long 0x5;
    .asciz "GNU";
    .long 0xc0000000; /* GNU_PROPERTY_AARCH64_FEATURE_1_AND */
    .long 4;
    .long (GNU_PROPERTY_AARCH64_POINTER_AUTH | GNU_PROPERTY_AARCH64_BTI);
    .long 0;
    .popsection;
#  endif
```

OpenSSL Code: arm_arch.h

Marking GNU Notes section

- Clang can output for BTI **-mmark-bti-property**.
- https://clang.llvm.org/docs/ClangCommandLineReference.html#cmdoption-clang-mmark-bti-property.
- It would be nice to have one for PAC too, but it does not.
- clang -mmark-bti-property -c empty.s -o empty.o.

```
bill@fedora:~/workspace/test$ readelf -n empty.o

Displaying notes found in: .note.gnu.property
  Owner                Data size        Description
  GNU                  0x00000010       NT_GNU_PROPERTY_TYPE_0
      Properties: AArch64 feature: BTI
```

PSTATE BTYPE

Processor Handling of Control Flow State

Instructions and PSTATE.BTYPE value

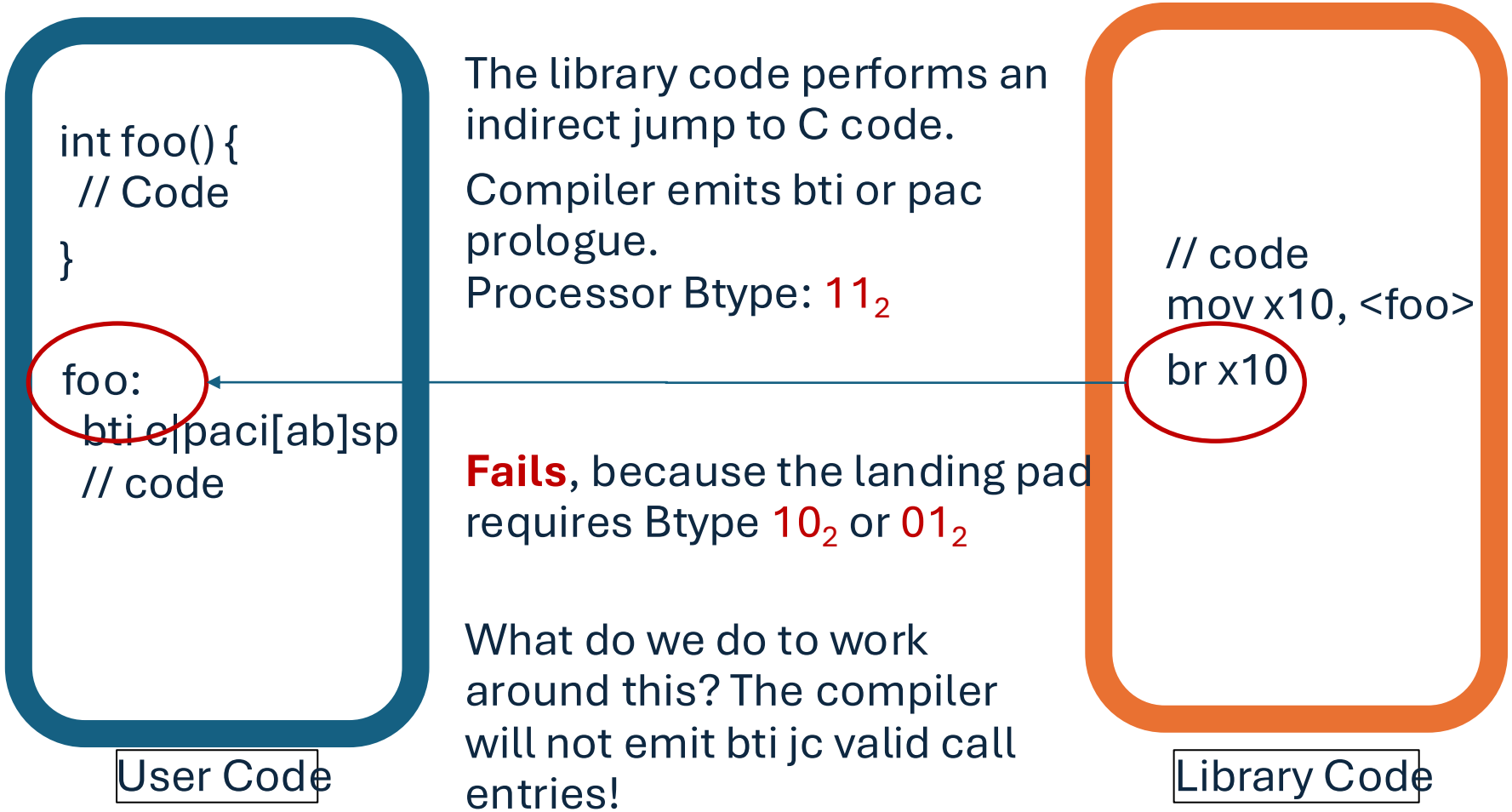
Branch Instructions	BTYPE
BR, BRAA, BRAB, BRAAZ, BRABZ	11_2
BLR, BLRAA, BLRAB, BLRAAZ, BLRABZ	10_2
BR, BRAA, BRAB, BRAAZ, BRABZ **(x16, x17)**	01_2

The x16, x17 register trick was useful in shim assembly between user function and library that would return to the caller and branched to internal routines or user function directly. User functions use bti c landing pads.

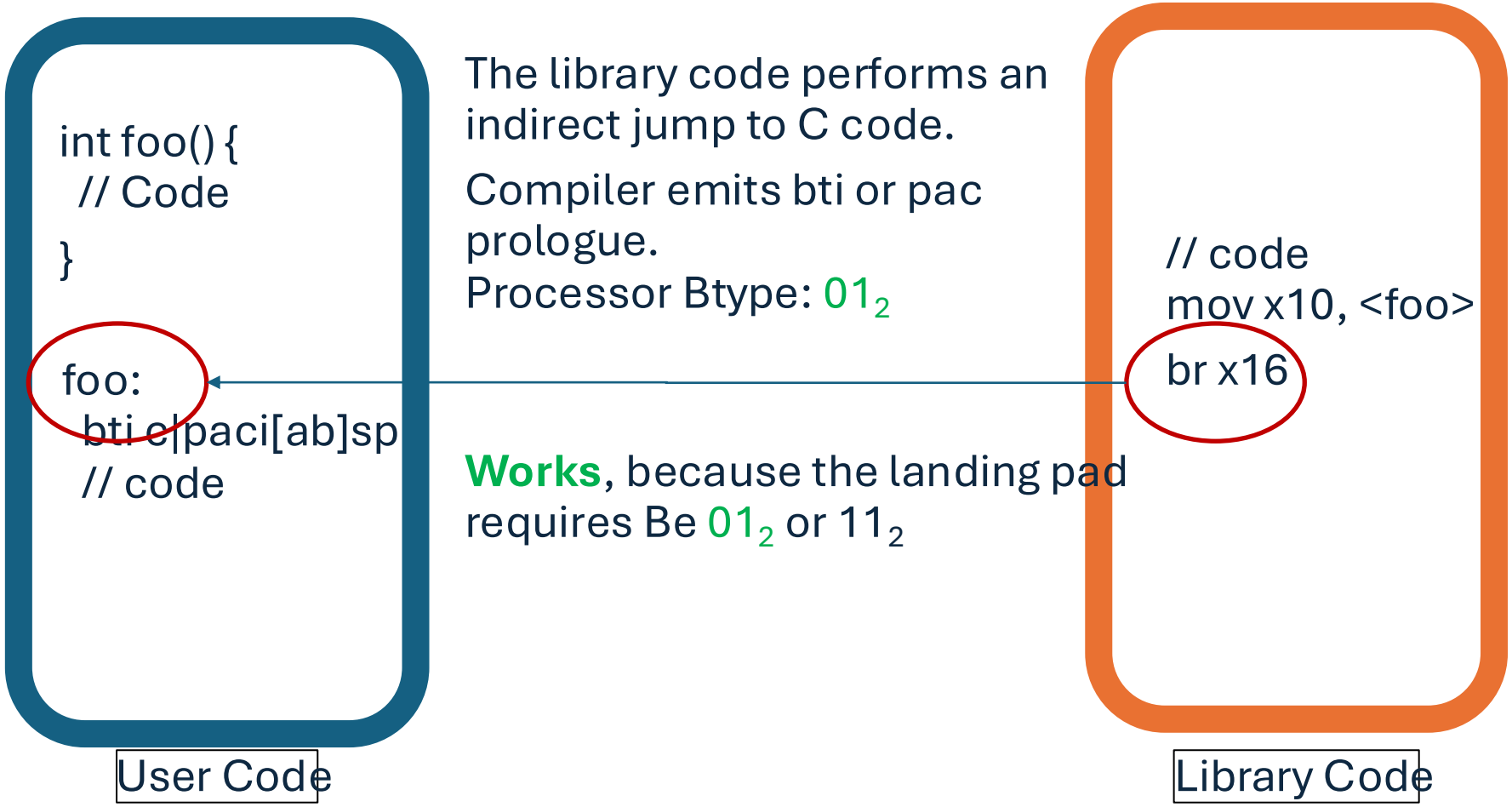
Accepted PSTATE.BTYPE State

Argument	Accepted State	Use Case
BTI c	10_2 and $\mathbf{01_2}$	Function Calls
BTI j	11_2 and $\mathbf{01_2}$	Jumps
BTI jc	All	All

Indirect Jump Failure bad PSTATE.BTYPE



Indirect Jump Using Intraprocedural Registers



Guarded Control Stack (GCS)

- The processor maintains a separate, protected, stack.
- Call and Return Instructions push and pop to that stack.
  - Example instructions would be brl and ret.
- Overwrites on the traditional stack cannot modify call and return information.
- ELF Binaries marked with GCS in GNU Notes.
  - Just like PAC and BTI examples, except OR in another bit: FEATURE_1_GCS.
- Runtime, typically libc, will go through all the binaries and check for GCS support.
  - Statically linked binaries just check that binary contains the marking in the GNU Notes section.
  - Dynamically linked binaries ALL dependencies must contain FEATURE_1_GCS in GNU Notes.
- Runtime enables it via prctl interface.
  - Note that once enabled, the function enabling it cannot return, underflow the stack.
  - Typically this is not a problem, it’s done before main() is called.

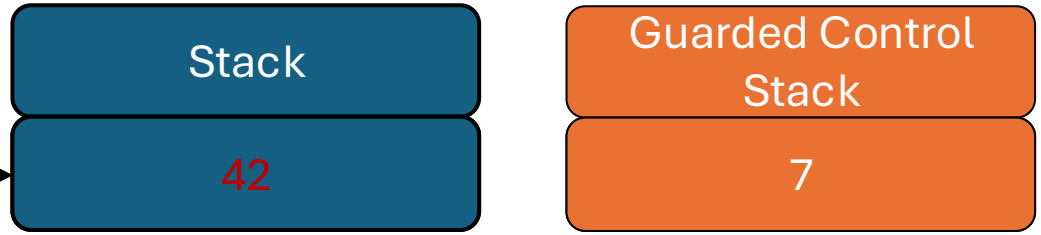
GCS Enabled

- Once GCS is enabled, the processor uses the protected stack for call and returns
- Branch and Link instructions work *exactly* as before, with an addition of pushing the return address to the protected stack
- The regular stack is still utilized *exactly* as before
- Return instructions work *exactly* as before, with the addition of when ret is invoked, the link register must match what is in the protected stack
- Only privileged code can modify the protected stack directly
  - prctl interface on Linux for doing this from user space
    - Push and pop the guarded stack or full write access
    - Typically not needed for most software, but language runtimes and debuggers may need it
  - These interfaces can be disabled as well using prctl
- Threads/Co-routines/Fibers etc, GCSSS1 and GCSS2 stack swap instructions

GCS Example

```
foo:
0: stp    x29, x30, [sp, -16]!
1: // code
2: ldp    x29, x30, [sp], 16
3: ret

main:
4: stp    x29, x30, [sp, -16]!
5: mov    x29, sp
6: bl     foo
7: ldp    x29, x30, [sp], 16
8: ret
```



The return value is popped from the stack into the LR

$42 \neq 7$

Code is vulnerable, attacker overwrites return address

Since the values do not match, an exception occurs

Other Factoids

- Some PAC instructions are also BTI landing pads, ie paci(a|b)sp is a valid BTI C target.
- cfi_window_save: is a SPARC directive for saving register window.
- __cxa_throw() does the heavy lifting in C++.
- LD_LIBRARY_PATH and LD_PRELOAD friends, you can load up a libgcc with –g and –O0.
- Using x16 or x17 as the indirect branch, eg br instruction, you can jump to bti ‘c’ landing pads instead of bti ‘j’.

Resources

- This was a journey, this is what I found helpful
- YouTube
  - Reliable Stack Traces, the Reality of Myth
  - Javier Honduvilla Coto - "How stack unwinding works?" - C++ London
  - Exploiting the hard-working DWARF
  - Linux Debuginfo Formats
- Documentation
  - DWARF: https://dwarfstd.org/
  - ARM DWARF: https://github.com/ARM-software/abi-aa/blob/main/aadwarf64/aadwarf64.rst
  - https://sourceware.org/binutils/docs/as/CFI-directives.html
  - https://developer.arm.com/documentation/102374/0102/Procedure-Call-Standard
  - https://developer.arm.com/documentation/101028/0012/5--Feature-test-macros
  - https://gpages.juszkiewicz.com.pl/arm-socs-table/arm-socs.html
  - https://developer.arm.com/documentation/ddi0602/2024-03/Shared-Pseudocode/aarch64-functions-pac
  - https://developer.arm.com/documentation/ddi0602/2024-03/Base-Instructions/AUTIA--AUTIA1716--AUTIASP--AUTIAZ--AUTIZA--Authenticate-Instruction-address--using-key-A-
  - https://developer.arm.com/documentation/102433/0200/Jump-oriented-programming

Q and A

https://xkcd.com/153/