PAC and BTI in Debian, what are they and why should I care?

Steve Capper

Debian MiniConf 2023-11-25

Intro to Pointer Authentication and Branch Target Identifiers

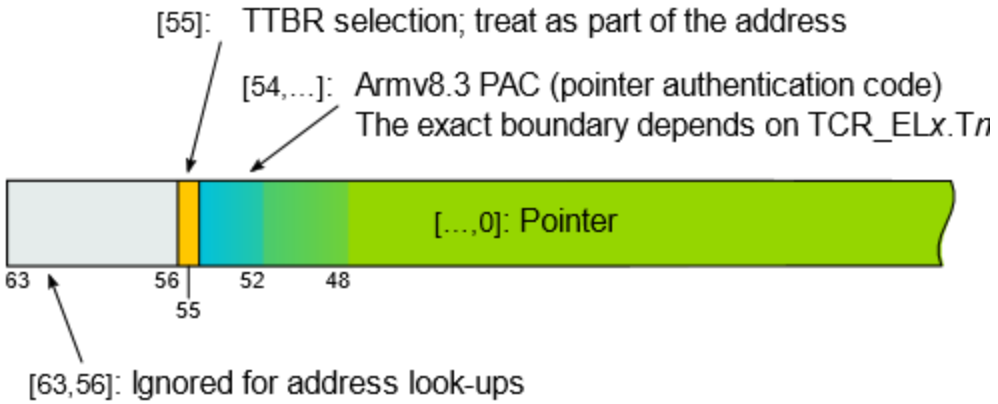
The problem we're trying to solve

```
stp    x29, x30, [sp, #-32]!
mov    x29, sp
<rest of function>
ldp    x29, x30, [sp], #32
ret
```

- It may be possible to overflow the stack/buffers with a payload that is controlled,
- On AArch64 neither the stack nor most data buffers are executable,
- Looking above, however, one can control the value of x30≡lr, thus they could **dictate control flow** to already resident code (f.e. glibc routines),
- Suitable routines (gadgets) can then lead to a Turing complete exploit in a data buffer,
- This is termed a return-oriented programming (ROP) attack.

Pointer Authentication (FEAT_PAC)

Cryptographically signed pointer authentication codes (PACs) were introduced in Armv8.3-A. They provide a means to protect pointers from outside manipulation.



1. Each process has its own secret keys, and it uses them to cryptographically *sign* pointers and produce a Pointer Authentication Code (PAC) in the upper bits of a virtual address,
2. Then before being used, a pointer is *authenticated*. If the PAC bits were valid, then the upper bits are cleared leaving a valid pointer. Otherwise, an invalid pointer is returned that will provoke a **translation fault**.

An example prologue/epilogue

Without pointer authentication

```
stp     x29, x30, [sp, #-48]!
mov     x29, sp

<rest of function>

ldp     x29, x30, [sp], #48
ret
```

With pointer authentication

```
paciasp ≡ pacia x30,sp
stp     x29, x30, [sp, #-48]!
mov     x29, sp

<rest of function>

ldp     x29, x30, [sp], #48
autiasp ≡ autia x30,sp
ret
```

- This would protect our example program from ROP style attacks,
- To aid with the deployment of Pointer Authentication, **some** of the instructions are in the “NOP space” (meaning that on systems without the support, the instructions are interpreted as a NOP),
- In gcc, passing `-mbranch-protection=standard` will produce code that uses pointer authentication (and another protection mechanism we’ll discuss later).

Building on top of pointer authentication (FEAT_BTI)

Pointer authentication can mitigate against return-oriented programming style attacks by authenticating pointers that are persisted within function calls (particularly link register), however there is another kind of exploit that it doesn't protect against: **jump-oriented programming** (JOP).

It is possible to also affect control flow by targeting indirect branches, f.e. "`br x0`". Then a chain of JOP gadgets can be formulated. Note that these branch targets may not necessarily persist across function calls so cannot be authenticated.

We can, however, mitigate against JOP-style attacks by restricting **where** indirect branches can land. Armv8.5-A introduces branch target identifiers (BTIs).

Branch Target Identifiers

Software can restrict the possible targets for indirect branches, in order to do this, both userspace and kernel space need to be involved.

From the userspace side: BTI instructions act as landing pads for indirect branches. These are also in the NOP space.

One employs the following gcc flag: `-mbranch-protection=standard` (which includes both PAC and BTI)

From the kernel side memory pages need to be marked as *guarded*. This is achieved by setting a bit in the page table entry. The libc loader will mmap pages as PROT_BTI if it determines that *all the execution units* are BTI aware.

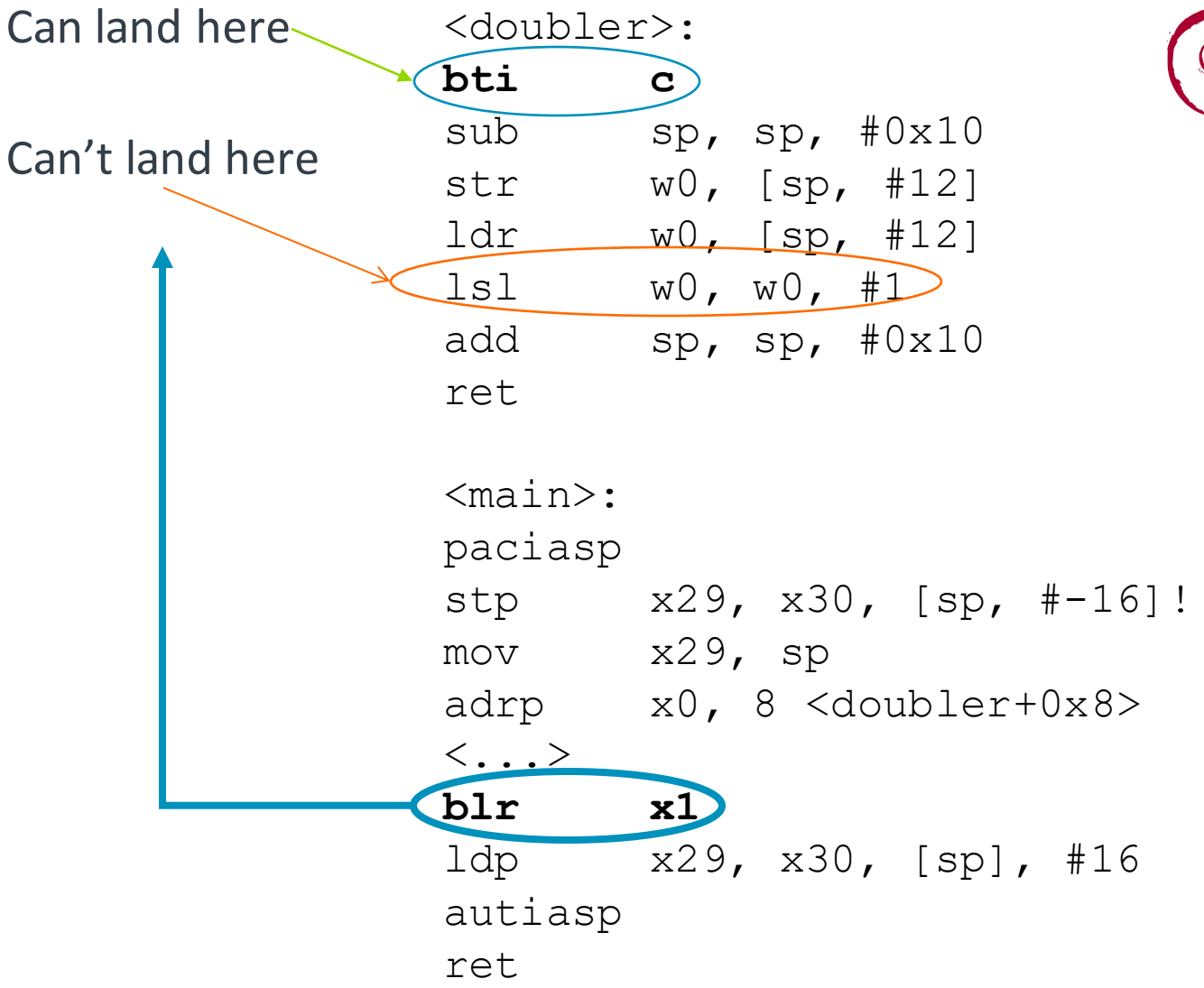
An example of BTI

```
int (*operation)(int argument);

int doubler(int argument)
{
  return 2*argument;
}

int main(void)
{
  operation = doubler;
  return operation(0);
}
```

(using gcc with `-mbranch-protection=standard`)



Deploying PAC + BTI

PAC deployment considerations

- PAC instructions are implemented in the prologues/epilogues of functions and are mostly self-contained,
- One exception is call-stack unwinders; they need to strip off the pointer authentication codes in order to correctly unwind,
- libunwind and friends already have PAC support,
- The Linux kernel, gcc and clang all have PAC support too.

BTI deployment considerations

- BTI is trickier to deploy because we need to worry about traversing execution units (f.e. libraries and executables),
- ELF files are marked with notes to advertise their BTI compatibility,
- The run time loader then mmap's with PROT_BTI as appropriate,
- When building software we basically have:

$$BTI_{executable} = \bigcap BTI_{execution\ units}$$

- In other words, an executable is marked as BTI compatible if and only if all the execution units of the program are also marked as BTI compatible,
- For deployment in Debian this means ensuring that BTI is enabled in all an executable's dependencies,
- Assembler is one area where special consideration is needed.

Enabling BTI in assembler

The first rule of assembler is… not to use assembler!

- When adding BTI instructions to an execution unit, one needs to also advertise their presence with a notes section, for example:

```
.pushsection .note.gnu.property, "a"
.balign 8
.long 4                 /* size of the name - "GNU\0" */
.long 0x10              /* size of descriptor */
.long 0x5               /* NT_GNU_PROPERTY_TYPE_0 */
.asciz "GNU"
.long 0xc0000000        /* pr_type - GNU_PROPERTY_AARCH64_FEATURE_1_AND */
.long 4                 /* pr_datasz - 4 bytes */
.long 3                 /* pr_data - GNU_PROPERTY_AARCH64_FEATURE_1_BTI | GNU_PROPERTY_AARCH64_FEATURE_1_PAC */
.long 0                 /* pr_padding - bring everything to 8 byte alignment */
.popsection
```

- The runes for the notes can be found documented here:
  - https://github.com/ARM-software/abi-aa/blob/2023Q3/aaelf64/aaelf64.rst
  - https://github.com/hjl-tools/linux-abi/wiki/linux-abi-draft.pdf
- And the BTI instruction can be found documented here:
  - https://developer.arm.com/documentation/102433/0100/Jump-oriented-programming

Debugging PAC and BTI

What happens if PAC or BTI get “tripped”?

This behaviour is for a Linux kernel – what to look for in a userspace crash dump…

PAC

1. Attack mode: return address corrupted,
2. The *authenticate* instruction will generate an invalid LR register, (because it will fail the cryptographic check)
3. A *translation fault* will occur on return,
4. Which will manifest as a **SIGSEGV**. (si_code: Address not mapped to object)

BTI

1. Attack mode: function pointer corrupted,
2. Branch occurs to an instruction that is not a BTI or PAC landing pad,
3. A *branch target exception* will occur, (because the page table entry will be marked as a *Guarded Page* by the kernel),
4. The kernel will inject this back into the userspace process as a **SIGILL**. (si_code: Illegal opcode)

Kernel kill switches

- If there is a suspected issue with PAC or BTI, it is possible to **completely** disable them (in both the kernel and in userspace) at boot time via the following kernel command line parameters:
  - **arm64.nopauth** – disable PAC
  - **arm64.nobti** – disable BTI
- The above will prevent the kernel from employing PAC or BTI itself as well as removing PAC or BTI from HWCAPS (and `/proc/cpuinfo`).
- Additionally; the NOP-space instructions will, again, behave as NOPs.

Am I actually compiling with BTI support?

- One can see if an ELF binary has BTI support by checking the notes, f.e.

```
readelf -n $(which ls)

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

- If one doesn't see BTI listed then the runtime loader will not activate BTI,
- We have just realized that BTI is not yet fully enabled for Debian Sid ☹

Okay… so why am I not getting BTI enabled executables?

- BTI will only be enabled if every single execution unit advertises BTI as being enabled,
- `-mbranch-protection=standard` won't complain if BTI doesn't get enabled,
- However, we can ask the linker to tell us off with `-z force-bti`,

```
gcc -mbranch-protection=standard -z force-bti  -o hello ./hello.c
/usr/bin/ld: /usr/lib/gcc/aarch64-linux-gnu/12/../../../aarch64-linux-gnu/Scrt1.o: warning: BTI turned on by -z force-
bti when all inputs do not have BTI in NOTE section.
/usr/bin/ld: /usr/lib/gcc/aarch64-linux-gnu/12/../../../aarch64-linux-gnu/crti.o: warning: BTI turned on by -z force-
bti when all inputs do not have BTI in NOTE section.
/usr/bin/ld: /usr/lib/gcc/aarch64-linux-gnu/12/crtbeginS.o: warning: BTI turned on by -z force-bti when all inputs do
not have BTI in NOTE section.
/usr/bin/ld: /usr/lib/gcc/aarch64-linux-gnu/12/crtendS.o: warning: BTI turned on by -z force-bti when all inputs do
not have BTI in NOTE section.
/usr/bin/ld: /usr/lib/gcc/aarch64-linux-gnu/12/../../../aarch64-linux-gnu/crtn.o: warning: BTI turned on by -z force-
bti when all inputs do not have BTI in NOTE section.
```

- It is then a matter to chase through the .o files and figure out why they aren't BTI enabled,
- Please do not upload any binaries with these warnings in, as they may crash on BTI enabled systems!

PAC + BTI status within Debian

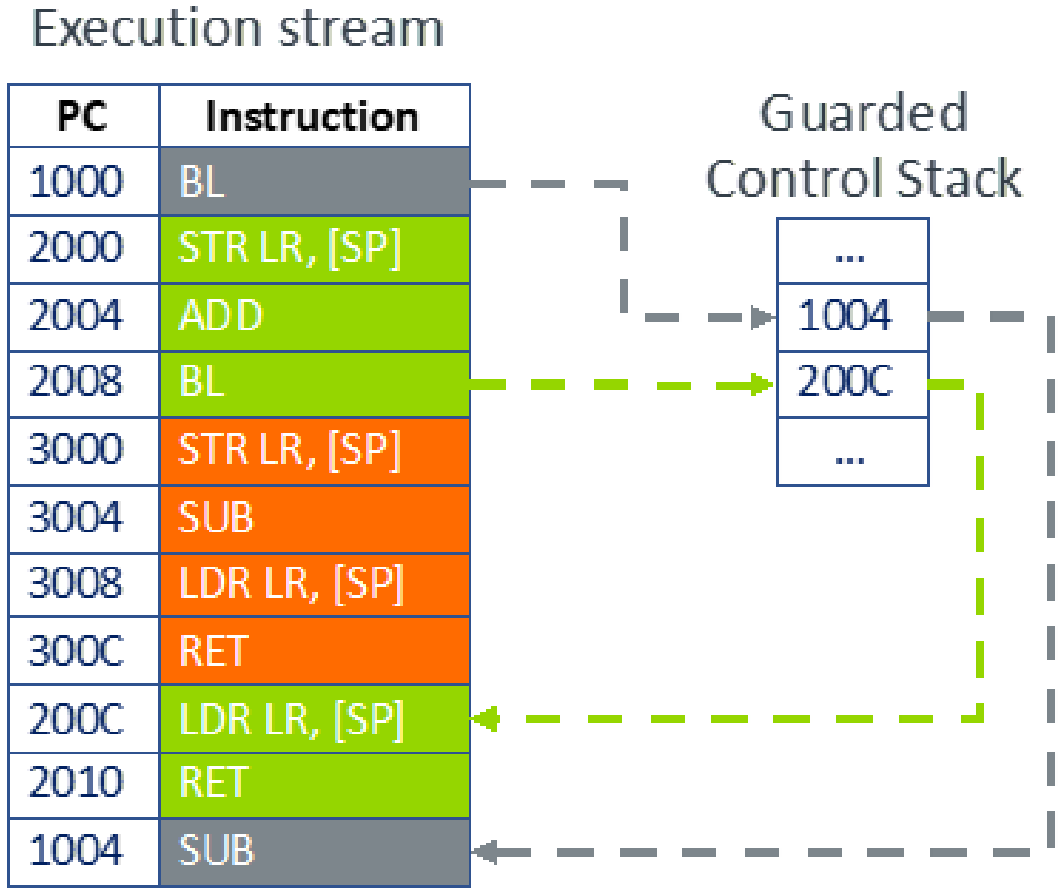
- The `-mbranch-protection=standard` flag is enabled for the “hardening flags” in dpkg-dev (thus are used by the buildd’s for most packages),
- However, we’ve just realized that the gcc package needs to have this enabled too!
- Once https://bugs.debian.org/cgi-bin/bugreport.cgi?bug=1055711 is resolved, we would expect BTI to make its way into most Debian “Trixie” packages,

Bonus architecture! Guarded Control Stack (GCS)

Guarded Control Stack

https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/arm-a-profile-architecture-2022

- This is an architectural extension in Armv9-A that explicitly protects the call-stack itself,
- The call-stack is duplicated in a protected memory region and with restricted read/write from userspace.
- Having the call-stack in its own area also facilitates profiling tools as walking the call-stack becomes significantly simpler,
- On the RHS diagram a BL pushes the return address on the control stack, and a RET pops from the control stack.



Status of GCS software support

- Support still needs to be sent upstream for gcc and the dynamic linker,
- Kernel patches can be found discussed on list:
  - https://lore.kernel.org/linux-arm-kernel/20231122-arm64-gcs-v7-0-201c483bd775@kernel.org/
- The approach above attempts to closely match the x86 shadow stack kernel mechanism with a view to maximising portability,
- Questions from our side:
  - Are folks interested in shadow stacks in general?
  - If so, would having GCS follow the existing shadow stack mechanisms be the preferred approach?

References

- The kernel documentation for PAC:
  - https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/arch/arm64/pointer-authentication.rst?h=v6.6
- ELF ABI documentation:
  - https://github.com/ARM-software/abi-aa/blob/2023Q3/aaelf64/aaelf64.rst
  - https://github.com/hjl-tools/linux-abi/wiki/linux-abi-draft.pdf
- Documentation on the BTI instruction:
  - https://developer.arm.com/documentation/102433/0100/Jump-oriented-programming
- Arm article on PAC + BTI:
  - https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/Learn%20the%20Architecture/Providing%20protection%20for%20complex%20software.pdf
- A nice summary on ROP + JOP style attacks:
  - https://llsoftsec.github.io/llsoftsecbook/#code-reuse-attacks
- A brief introduction to GCS:
  - https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/arm-a-profile-architecture-2022

arm

Thank you for your attention!
Any questions/comments?

Thank You
Danke
Gracias
Grazie
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה

Gratuitous Recruitment Spam

Apologies, nothing for Debian directly this time.

- We are recruiting software engineers in Sunny Manchester and Cambridge at Arm, for the following open source software roles:
  - Software defined networking,
  - Automotive and Industrial solutions,
  - Software defined storage and transactional databases,
  - Toolchains,
  - Linux Kernel & Android.
- Should anyone here be interested (or know anyone who may be interested); please do get in touch with me: steve.capper@arm.com
- There is a careers page which I can help folk navigate too:
  - https://careers.arm.com
- **I would be more than happy to answer any recruitment queries.**