Enabling Architectural Features in Debian: PAC and BTI on arm64

Emanuele Rocca
FOSDEM, February 2025

whoami

- ema
- Debian Developer since 2003
- At Arm since 2023

whoami

- ema
- Debian Developer since 2003
- At Arm since 2023
- Doesn’t Debian work on arm64 already?? (cit.)

What is a Linux Distribution?

A bunch of packages?

- Source and/or Binary

What is a Linux Distribution?

A bunch of packages?

- Source and/or Binary
- With an installer

What is a Linux Distribution?

A bunch of packages?

- Source and/or Binary
- With an installer
- At a given point in time/version (a release)

What is a Linux Distribution?

A bunch of packages?

- Source and/or Binary
- With an installer
- At a given point in time/version (a release)
- Group of people / teams agreeing on rules and procedures

Outline

PAC/BTI?

Enabling the features

Tracking deployment status

The road ahead

PAC/BTI?

arm

PAC/BTI?

- Security features designed to mitigate control flow attacks
- Stack/buffers are not executable, but one can hijack control flow to legit code (eg: glibc routines)
- **PAC**: Pointer Authentication, mitigates Return-Oriented Programming (ROP) attacks
- **BTI**: Branch Target Identification, mitigates Jump-Oriented Programming (JOP) attacks

PAC mitigates ROP attacks

- ROP attack: overwrite return addresses to chain together legit code (gadgets)
- PAC mitigation: at the start of a function the return address in the LR is signed (PACIASP)
- Before returning, the return address is authenticated (AUTIASP)

BTI mitigates JOP attacks

- Hijack control flow by targeting indirect branches (jumps)
- The target can be any executable address, and not just the return address of function calls
- Restrict **where** we can jump to (Branch Target Identifiers)
- Special instructions aka landing pads

Enabling the features

arm

Enabling the features

Compiler Flags

- **-mbranch-protection=standard**
  both PAC and BTI
- **-mbranch-protection=pac-ret**
  PAC only
- **-mbranch-protection=bti**
  BTI only

Enabling PAC

- PAC instructions added to built object
- CPUs supporting PAC will sign (eg: PACIASP) and authenticate addresses (eg: AUTIASP, RETAA)
- CPUs without pauth support execute NOP
- `objdump -d /usr/bin/ls | grep -E 'pac|aut'`

Enabling BTI

- All .o files combined together by the linker need to be built with BTI support
- Only packages built with a BTI-enabled GCC and glibc get BTI support
- The linker adds a note to the notes section of the ELF section if the whole file is BTI aware
- The loader enables the feature at runtime if that's the case
- Most programs are linked using crtbeginS.o and crtendS.o from GCC
- crti.o, Scrt1.o and crtn.o from the GNU C Library (glibc)

Enablement in Debian

Adding -mbranch-protection=standard to key packages

- dpkg 1.22.0, 30 August 2023 adds PAC/BTI to default build flags of all packages
- gcc-13 13.3.0-2, 8 July 2024
- gcc-14 14.1.0-4, 10 July 2024
- glibc 2.39-5, 22 July 2024

All packages built after July 22, 2024 and using the default build flags get BTI support. Possible to opt-out.

Tracking deployment status

arm

Does this ELF file have PAC/BTI?

Check the ELF notes!

```
$ readelf --notes /bin/ls | grep Properties
Properties:  AArch64 feature:  BTI, PAC
```


arm

Which packages have BTI support?

- Analyze the archive using **snapshot.debian.org**: time machine for the Debian archive
- JSON API: list of snapshosts like 20250109T024634Z, 20250109T084424Z ...
  `http://snapshot.debian.org/mr/timestamp/?after=2025-01-09`
- Download all arch-dependent debs found on
  http://snapshot.d.o/archive/debian/$TS/dists/sid/main/binary-arm64/Packages.gz
- `dpkg --extract $deb`, for each ELF file
- `readelf --notes $f | grep -q 'AArch64 feature:  BTI, PAC'`
- Downloading the archive takes about 20 minutes
- Running the analysis with `parallel` on a 8 core system takes 30 minutes

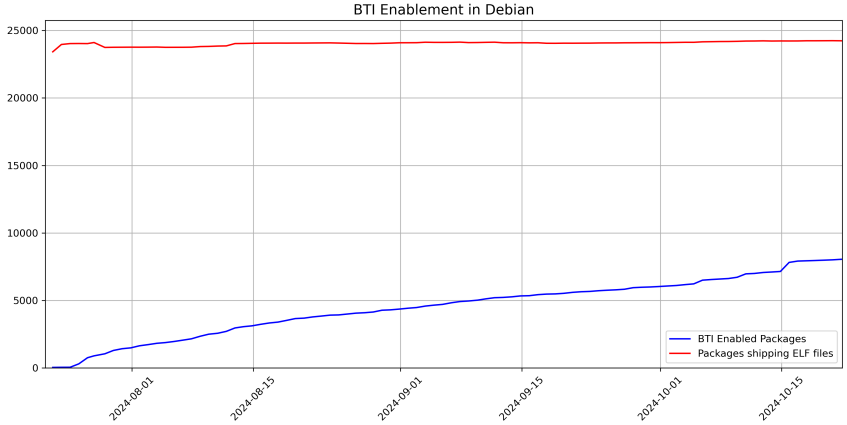

Figure: Packages with BTI before rebuilds

Low-hanging fruits

Which packages would get BTI with a simple rebuild?

- Chat with Paul Gevers at the Cambridge miniDebconf
- Make a list of all binary packages (_arm64.deb, not _all.deb) **shipping at least one ELF file**
- Find out all those without BTI (definition: zero ELF files with BTI on)
- Rebuild the corresponding source packages
- Do they get BTI now? Most do! 7367 source packages to be precise.

Mass rebuilds

- Double-check that all now-BTI-enabled packages were last uploaded before July 22
- Send list of 7K source packages to the Release Team asking for rebuilds (Debianese: binNMUs)
- Sebastian Ramacher took care of scheduling the rebuilds. Thanks!

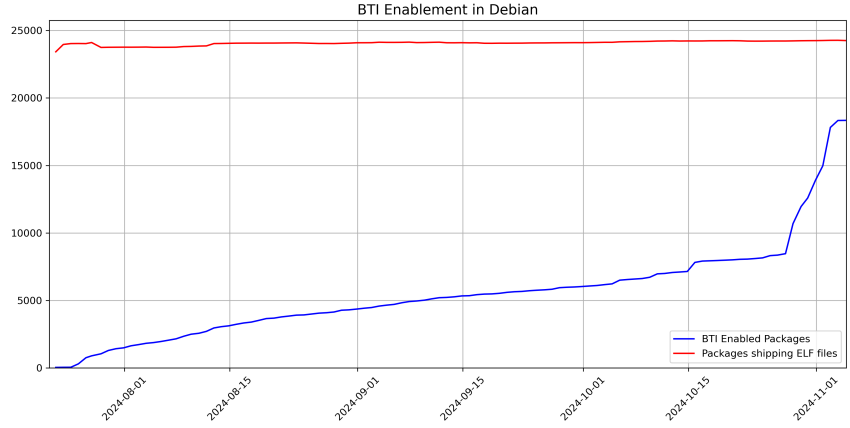

Figure: Packages with BTI after rebuilds

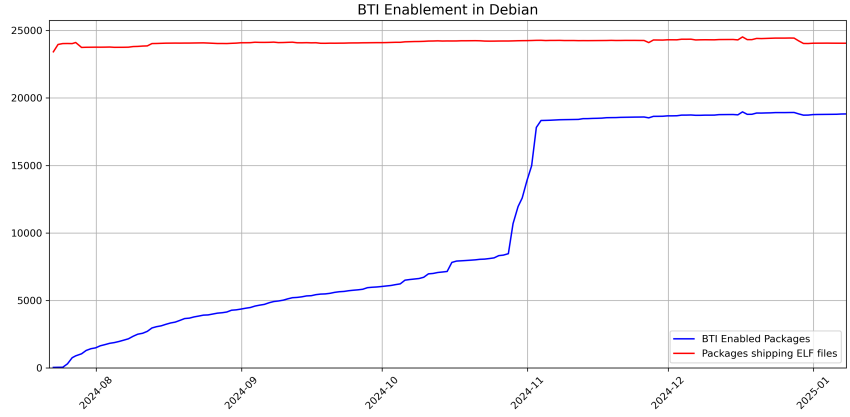

Figure: Packages with BTI in early January 2025

The road ahead

arm

Status in January 2025

- 18834 binary packages have BTI
- 5239 don’t. They come from 3415 source packages

Source packages without BTI

By language

- GCC and LLVM are not the only compilers producing ELF files in Debian!

Source packages without BTI

By language

- GCC and LLVM are not the only compilers producing ELF files in Debian!
- 1179 haskell

arm

Source packages without BTI

By language

- GCC and LLVM are not the only compilers producing ELF files in Debian!
- 1179 haskell
- 423 go

Source packages without BTI

By language

- GCC and LLVM are not the only compilers producing ELF files in Debian!
- 1179 haskell
- 423 go
- 273 ocaml

Source packages without BTI

By language

- GCC and LLVM are not the only compilers producing ELF files in Debian!
- 1179 haskell
- 423 go
- 273 ocaml
- 213 rust

arm

Source packages without BTI

Next steps

- haskell: work ongoing at Arm
- go: upstream task `https://github.com/golang/go/issues/66054`
- rust: upstream task `https://github.com/rust-lang/rust/issues/113369`

arm

Source packages without BTI

The long tail

- Another 1327 packages need to be looked into
- Ignoring CFLAGS
- Long tail of strange cases

arm

Conclusions

- PAC and BTI are useful arm64 features available in Trixie: `https://wiki.debian.org/ToolChain/PACBTI`
- Distro work is not only hacking. Communication, coordination, tracking status…
- No integration work, no feature

…but I thought rust was safe!

- Language safety checks in safe languages and Control-Flow Integrity break different stages of the exploit
- An attacker can carefully maneuver between the languages to mount a successful attack
- Mergendahl, Samuel, Nathan Burow, and Hamed Okhravi. ”Cross-Language Attacks.” NDSS. 2022.