
AWS
re:Invent
NOV. 28 – DEC. 2, 2022 | LAS VEGAS, NV

CMP409-R

Enhancing OS and application security on AWS Graviton–based instances

Will Koplitz

Senior Graviton Specialist SA
AWS

Arthur Petitpierre

Principal Graviton Specialist SA
AWS

Arthur Petitpierre – Who am I?

2016-2019
Delivering Snowballs – Paris

2019-today
Paddling for AWS Graviton – Seattle

But also:
$JOB-1 : HPC Services CTO
2016 - 2019 : AWS EMEA HPC Specialist SA
2019 - today : AWS Graviton Specialist SA

AWS Graviton2 vs. AWS Graviton3

Processor	Graviton2	Graviton3
Instances	M6g/M6gd, C6g/C6gd/C6gn, R6g/R6gd, T4g, X2gd, G5g, and Im4gn/Is4gen	C7g
Core	Neoverse-N1	Neoverse-V1
Frequency	2500 MHz	2600 MHz
Turbo supported	No	No
Instruction latencies	Instruction latencies	Instruction latencies
Interconnect	CMN-600	CMN-700
Architecture revision	ARMv8.2-a	ARMv8.4-a
Additional features	fp16, rcpc, dotprod, crypto	sve, rng, bf16, int8, crypto
Recommended -mcpu flag	Neoverse-n1	Neoverse-512tvb
RNG instructions	No	Yes
SIMD instructions	2x Neon 128-bit vectors	4x Neon 128-bit vectors / 2x SVE 256-bit
LSE (atomic mem operations)	Yes	Yes
Pointer authentication	No	Yes
Cores	64	64
L1 cache (per core)	64 KB inst / 64 KB data	64 KB inst / 64 KB data
L2 cache (per core)	1 MB	1 MB
LLC (shared)	32 MB	32 MB
DRAM	8x DDR4	8x DDR5
DDR encryption	Yes	Yes

Pointer authentication – Why?

Many exploits come down to convincing code (kernel or otherwise) to access a pointer that was crafted by an unauthorized user

Buffer-overflow exploits and return-oriented programming, for example, both rely on placing a pointer where a return address is expected; when the processor “returns” to that address, the unauthorized user takes control

ARMv8.3 pointer authentication

Arm64 memory model (see Documentation/arm64/memory.txt)

3-level page table

64-bit pointers, but only the bottom 40 bits are used

Remaining 20 bits are equal to highest significant bit

So they could be used for something else . . .

Pointer authentication code (1/2)

Using the uppermost bits to store an authentication code

Code is calculated from 3 values (`PAC instruction`):

- Pointer itself
- A secret key hidden in the process context
- The current stack pointer

Pointer authentication code (2/2)

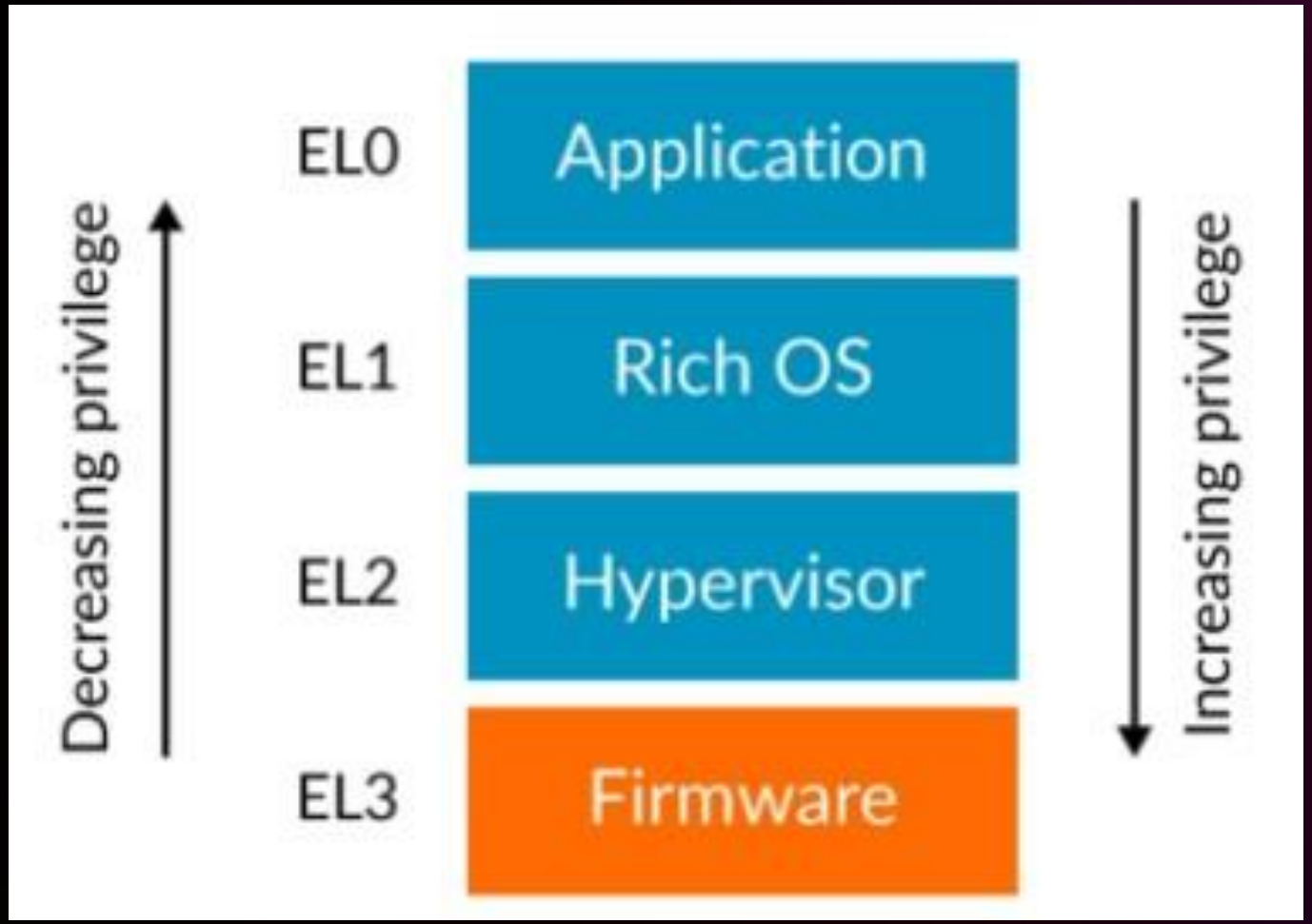
A pointer with the authentication code can’t be dereferenced directly (not a valid address), so would generate an exception

Regaining the pointer value (`AUT instruction`) will recalculate the authentication code, and compare

If the two match, signature is removed

Otherwise, pointer is modified to ensure a fault

ARM64 exception levels



Linux kernel implementation

The processor provides 5 separate keys: two for executable pointers, two for data, and one “general” key (128-bit values)

Can be used but not read at EL0

Key is assigned at exec(), inherited by threads, childs

Stored in process context, in system registers

Support for userland added in 4.21 (ARM64_PTR_AUTH)

5.7 adds support for in-kernel pointer authentication

Compiler-level support

GCC:

- GCC7 adds -msign-return-address=non-leaf|all
- GCC9 adds -mbranch-protection=none|standard|pac-ret[+leaf+b-key]|bti

LLVM: Support starting from version 8

Userland and Linux distributions

Glibc: PAC support starting from 2.32

AMZL2: Supported at the kernel level (4.14, 5.10), Glibc and compiler, but nothing compiled with PtrAuth

Real support targeted with AL2022

Ubuntu: Kernel, glibc, and compiler support in 22.04; runtimes should come with next version

Performance impact

PACIASP: Generally a single cycle

AUTIASP: Up to 16 cycles, but well predicted by branch predictor

ARMv8.4-a introduces RETAA, which combines AUTIASP+RET

PACIASP and AUTIASP are NOP on arch < ARMv8.4-a

In practice: Tested on Node.js application, saw no measurable impact (your mileage may vary)

Why sometimes assembly matters

This statement seems to indicate that the PAC instructions, being in the reserved NOP space, should be ignored by older ARMv8 processors. However, when my build of the Node.js runtime is executed on a Graviton2 CPU, this is the result:

```
(gdb) r
Starting program: /home/jgowdy/node-v16.14.2-linux-arm64/bin/node
Missing separate debuginfos, use: debuginfo-install glibc-2.26-58.amzn2.aarch64
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".

Program received signal SIGILL, Illegal instruction.
0x000000000155da5c in uv_mutex_init (mutex=<optimized out>) at ../deps/uv/src/unix/thread.c:284
284     ../deps/uv/src/unix/thread.c: No such file or directory.
Missing separate debuginfos, use: debuginfo-install libgcc-7.3.1-13.amzn2.aarch64 libstdc++-7.3.1-
13.amzn2.aarch64
(gdb) disassemble
Dump of assembler code for function uv_mutex_init:
   0x000000000155da40 <+0>:     paciasp
   0x000000000155da44 <+4>:     stp     x29, x30, [sp, #-16]!
   0x000000000155da48 <+8>:     mov     x1, #0x0                        // #0
   0x000000000155da4c <+12>:    mov     x29, sp
   0x000000000155da50 <+16>:    bl      0xa02510 <pthread_mutex_init@plt>
   0x000000000155da54 <+20>:    ldp     x29, x30, [sp], #16
   0x000000000155da58 <+24>:    neg     w0, w0
=> 0x000000000155da5c <+28>:    retaa
End of assembler dump.
```

Let’s check that on a live system

AWS Graviton getting started guide on GitHub

https://github.com/aws/aws-graviton-getting-started

- This guide has been assembled by our Graviton team and is designed to help users transition and optimize their applications
- It covers various languages and libraries, and includes tips and tricks for each
- In general, using latest versions of operating systems, compilers, and language runtimes will provide access to latest Arm64 improvements and optimizations

Thank you!

Will Koplitz
kopl@amazon.com

Arthur Petitpierre
arthurpt@amazon.com
@ArthurPtP

Please complete the session survey in the **mobile app**