# Pointer authentication in AArch64 Linux

Author: Mark Rutland <mark.rutland@arm.com>

Date: 2017-07-19

This document briefly describes the provision of pointer authentication functionality in AArch64 Linux.

## Architecture overview

The ARMv8.3 Pointer Authentication extension adds primitives that can be used to mitigate certain classes of attack where an attacker can corrupt the contents of some memory (e.g. the stack).

The extension uses a Pointer Authentication Code (PAC) to determine whether pointers have been modified unexpectedly. A PAC is derived from a pointer, another value (such as the stack pointer), and a secret key held in system registers.

The extension adds instructions to insert a valid PAC into a pointer, and to verify/remove the PAC from a pointer. The PAC occupies a number of high-order bits of the pointer, which varies dependent on the configured virtual address size and whether pointer tagging is in use.

A subset of these instructions have been allocated from the HINT encoding space. In the absence of the extension (or when disabled), these instructions behave as NOPs. Applications and libraries using these instructions operate correctly regardless of the presence of the extension.

The extension provides five separate keys to generate PACs - two for instruction addresses (APIAKey, APIBKey), two for data addresses (APDAKey, APDBKey), and one for generic authentication (APGAKey).

## Basic support

When CONFIG_ARM64_PTR_AUTH is selected, and relevant HW support is present, the kernel will assign random key values to each process at exec*() time. The keys are shared by all threads within the process, and are preserved across fork().

Presence of address authentication functionality is advertised via HWCAP_PACA, and generic authentication functionality via HWCAP_PACG.

The number of bits that the PAC occupies in a pointer is 55 minus the virtual address size configured by the kernel. For example, with a virtual address size of 48, the PAC is 7 bits wide.

When ARM64_PTR_AUTH_KERNEL is selected, the kernel will be compiled with HINT space pointer authentication instructions protecting function returns. Kernels built with this option will work on hardware with or without pointer authentication support.

In addition to exec(), keys can also be reinitialized to random values using the PR_PAC_RESET_KEYS prctl. A bitmask of PR_PAC_APIAKEY, PR_PAC_APIBKEY, PR_PAC_APDAKEY, PR_PAC_APDBKEY and PR_PAC_APGAKEY specifies which keys are to be reinitialized; specifying 0 means "all keys".

## Debugging

When CONFIG_ARM64_PTR_AUTH is selected, and HW support for address authentication is present, the kernel will expose the position of TTBR0 PAC bits in the NT_ARM_PAC_MASK regset (struct user_pac_mask), which userspace can acquire via PTRACE_GETREGSET.

The regset is exposed only when HWCAP_PACA is set. Separate masks are exposed for data pointers and instruction pointers, as the set of PAC bits can vary between the two. Note that the masks apply to TTBR0 addresses, and are not valid to apply to TTBR1 addresses (e.g. kernel pointers).

Additionally, when CONFIG_CHECKPOINT_RESTORE is also set, the kernel will expose the NT_ARM_PACA_KEYS and NT_ARM_PACG_KEYS regsets (struct user_pac_address_keys and struct user_pac_generic_keys). These can be used to get and set the keys for a thread.

## Virtualization

Pointer authentication is enabled in KVM guest when each virtual cpu is initialised by passing flags KVM_ARM_VCPU_PTRAUTH_[ADDRESS/GENERIC] and requesting these two separate cpu features to be enabled. The current KVM guest implementation works by enabling both features together, so both these userspace flags are checked before enabling pointer authentication. The separate userspace flag will allow to have no userspace ABI changes if support is added in the future to allow these two features to be enabled independently of one another.

As Arm Architecture specifies that Pointer Authentication feature is implemented along with the VHE feature so KVM arm64 ptrauth code relies on VHE mode to be present.

Additionally, when these vcpu feature flags are not set then KVM will filter out the Pointer Authentication system key registers from

KVM_GET/SET_REG_* ioctls and mask those features from cpufeature ID register. Any attempt to use the Pointer Authentication instructions will result in an UNDEFINED exception being injected into the guest.

## Enabling and disabling keys

The prctl PR_PAC_SET_ENABLED_KEYS allows the user program to control which PAC keys are enabled in a particular task. It takes two arguments, the first being a bitmask of PR_PAC_APIAKEY, PR_PAC_APIBKEY, PR_PAC_APDAKEY and PR_PAC_APDBKEY specifying which keys shall be affected by this prctl, and the second being a bitmask of the same bits specifying whether the key should be enabled or disabled. For example:

```
prctl(PR_PAC_SET_ENABLED_KEYS,
      PR_PAC_APIAKEY | PR_PAC_APIBKEY | PR_PAC_APDAKEY | PR_PAC_APDBKEY,
      PR_PAC_APIBKEY, 0, 0);
```

disables all keys except the IB key.

The main reason why this is useful is to enable a userspace ABI that uses PAC instructions to sign and authenticate function pointers and other pointers exposed outside of the function, while still allowing binaries conforming to the ABI to interoperate with legacy binaries that do not sign or authenticate pointers.

The idea is that a dynamic loader or early startup code would issue this prctl very early after establishing that a process may load legacy binaries, but before executing any PAC instructions.

For compatibility with previous kernel versions, processes start up with IA, IB, DA and DB enabled, and are reset to this state on exec(). Processes created via fork() and clone() inherit the key enabled state from the calling process.

It is recommended to avoid disabling the IA key, as this has higher performance overhead than disabling any of the other keys.