AArch64 TAGGED ADDRESS ABI

Authors: Vincenzo Frascino <vincenzo.frascino@arm.com>
Catalin Marinas <catalin.marinas@arm.com>

Date: 21 August 2019

This document describes the usage and semantics of the Tagged Address ABI on AArch64 Linux.

1. Introduction

On AArch64 the `TCR_EL1.TBI0` bit is set by default, allowing userspace (EL0) to perform memory accesses through 64-bit pointers with a non-zero top byte. This document describes the relaxation of the syscall ABI that allows userspace to pass certain tagged pointers to kernel syscalls.

2. AArch64 Tagged Address ABI

From the kernel syscall interface perspective and for the purposes of this document, a "valid tagged pointer" is a pointer with a potentially non-zero top-byte that references an address in the user process address space obtained in one of the following ways:

- `mmap()` syscall where either:
  - flags have the `MAP_ANONYMOUS` bit set or
  - the file descriptor refers to a regular file (including those returned by `memfd_create()`) or `/dev/zero`
- `brk()` syscall (i.e. the heap area between the initial location of the program break at process creation and its current location).
- any memory mapped by the kernel in the address space of the process during creation and with the same restrictions as for `mmap()` above (e.g. data, bss, stack).

The AArch64 Tagged Address ABI has two stages of relaxation depending on how the user addresses are used by the kernel:

1. User addresses not accessed by the kernel but used for address space management (e.g. `mprotect()`, `madvise()`). The use of valid tagged pointers in this context is allowed with these exceptions:

   - `brk()`, `mmap()` and the `new_address` argument to `mremap()` as these have the potential to alias with existing user addresses.

     NOTE: This behaviour changed in v5.6 and so some earlier kernels may incorrectly accept valid tagged pointers for the `brk()`, `mmap()` and `mremap()` system calls.

   - The `range.start`, `start` and `dst` arguments to the `UFFDIO_*` `ioctl()``s used on a file descriptor obtained from ``userfaultfd()`, as fault addresses subsequently obtained by reading the file descriptor will be untagged, which may otherwise confuse tag-unaware programs.

     NOTE: This behaviour changed in v5.14 and so some earlier kernels may incorrectly accept valid tagged pointers for this system call.

2. User addresses accessed by the kernel (e.g. `write()`). This ABI relaxation is disabled by default and the application thread needs to explicitly enable it via `prctl()` as follows:

   - `PR_SET_TAGGED_ADDR_CTRL`: enable or disable the AArch64 Tagged Address ABI for the calling thread.

     The `(unsigned int) arg2` argument is a bit mask describing the control mode used:

     - `PR_TAGGED_ADDR_ENABLE`: enable AArch64 Tagged Address ABI. Default status is disabled.

     Arguments `arg3`, `arg4`, and `arg5` must be 0.

   - `PR_GET_TAGGED_ADDR_CTRL`: get the status of the AArch64 Tagged Address ABI for the calling thread.

     Arguments `arg2`, `arg3`, `arg4`, and `arg5` must be 0.

   The ABI properties described above are thread-scoped, inherited on clone() and fork() and cleared on exec().

   Calling `prctl(PR_SET_TAGGED_ADDR_CTRL, PR_TAGGED_ADDR_ENABLE, 0, 0, 0)` returns `-EINVAL` if the AArch64 Tagged Address ABI is globally disabled by `sysctl abi.tagged_addr_disabled=1`. The default `sysctl abi.tagged_addr_disabled` configuration is 0.

When the AArch64 Tagged Address ABI is enabled for a thread, the following behaviours are guaranteed:

- All syscalls except the cases mentioned in section 3 can accept any valid tagged pointer.
- The syscall behaviour is undefined for invalid tagged pointers: it may result in an error code being returned, a (fatal) signal being raised, or other modes of failure.
- The syscall behaviour for a valid tagged pointer is the same as for the corresponding untagged pointer.

A definition of the meaning of tagged pointers on AArch64 can be found in Documentation/arm64/tagged-pointers.rst.

3. AArch64 Tagged Address ABI Exceptions

3. AArch64 Tagged Address ABI Exceptions

The following system call parameters must be untagged regardless of the ABI relaxation:

- `prctl()` other than pointers to user data either passed directly or indirectly as arguments to be accessed by the kernel.
- `ioctl()` other than pointers to user data either passed directly or indirectly as arguments to be accessed by the kernel.
- `shmat()` and `shmdt()`.
- `brk()` (since kernel v5.6).
- `mmap()` (since kernel v5.6).
- `mremap()`, the `new_address` argument (since kernel v5.6).

Any attempt to use non-zero tagged pointers may result in an error code being returned, a (fatal) signal being raised, or other modes of failure.

4. Example of correct usage

```c
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/mman.h>
#include <sys/prctl.h>

#define PR_SET_TAGGED_ADDR_CTRL      55
#define PR_TAGGED_ADDR_ENABLE        (1UL << 0)

#define TAG_SHIFT            56

int main(void)
{
     int tbi_enabled = 0;
     unsigned long tag = 0;
     char *ptr;

     /* check/enable the tagged address ABI */
     if (!prctl(PR_SET_TAGGED_ADDR_CTRL, PR_TAGGED_ADDR_ENABLE, 0, 0, 0))
             tbi_enabled = 1;

     /* memory allocation */
     ptr = mmap(NULL, sysconf(_SC_PAGE_SIZE), PROT_READ | PROT_WRITE,
                MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
     if (ptr == MAP_FAILED)
             return 1;

     /* set a non-zero tag if the ABI is available */
     if (tbi_enabled)
             tag = rand() & 0xff;
     ptr = (char *)((unsigned long)ptr | (tag << TAG_SHIFT));

     /* memory access to a tagged address */
     strcpy(ptr, "tagged pointer\n");

     /* syscall with a tagged pointer */
     write(1, ptr, strlen(ptr));

     return 0;
}
```