White paper

Pointer/Data Verification with the PX5 RTOS

By Bill Lamie

Published January 25, 2023

Pointer/Data Verification with the PX5 RTOS

By Bill Lamie, PX5

Today, safety and security for embedded devices are paramount. While distinct requirements, they have a high degree of overlap in the embedded PX5 RTOS context. For one, the PT5 RTOS can help in avoiding memory corruption, the most common source of safety and security issues in embedded devices. Moreover, leveraging the patent-pending Pointer/Data Verification technology unique to the PX5 RTOS can greatly enhance the safety and security profile of your embedded application.

What is Pointer/Data Verification (PDV)?

Pointer/Data Verification is a software-only technique to help detect and mitigate both intentional and accidental memory corruption. The operating system creates and stores a verification code for important information. Then, before the important information is used, a new verification code is generated and compared with what was stored previously. If the codes don't match, memory corruption has occurred and the PX5 RTOS immediately alerts the application by calling the central error handling function. Applications can define what happens in the central error handling.

Verification code calculation

The application can define the formula for generating the verification code. However, by default the verification code is a combination of a run-time identification (secret) passed to the PX5 RTOS in the px5_pthread_start API along with the value of the important information and the address to store the generated code. The default formula looks something like this:

Verification Code = ((Data Value) + (Address to Store Code) + (Secret)) ^ (Secret)

For verification codes that are run-time unique, we recommend using a True Random Number Generator (TRNG) if available in hardware. With use of a TRNG, the verification code for each important data element has a temporal property, i.e., it will be unique for each execution of the application running on top of the PX5 RTOS. This makes it much harder for hackers to insert malicious information such as function pointers for remote execution attacks. The verification code address provides a special property to the verification code. It's unlikely that any two images will have the same memory layout, which again makes it more difficult for hackers to successfully change important information without detection.

Verified Information

The PX5 RTOS provides optional PDV protection over a series of important internal information, including:

- All function pointers used in the PX5 RTOS
- Global data of the PX5 RTOS
- Internal system structures with the PX5 RTOS (threads, queues, etc.)
- Return addresses on internal PX5 RTOS functions
- Metadata pointers used for memory management
- APIs for application-specific use of PDV

Enabling PDV

As mentioned, using PDV is optional and is not enabled by default. In addition, it can be individually enabled for specific PX5 RTOS areas. To enable PDV, the PX5 RTOS source should be built with the following defines (depending on the exact verification requested):

PX5_FUNCTION_POINTER_VERIFY_ENABLE	When defined, all function pointers used internally in the PX5 RTOS are evaluated against the verification code that established when they were set up. With this enabled, it's much harder for hackers to insert rogue function pointers in remote execution attacks.
PX5_OBJECT_VERIFY_ENABLE	When defined, all internal PX5 RTOS objects (including the global PX5 RTOS data) are evaluated against the verification code that was established when they were created. This mechanism facilitates early detection of intentional and accidental memory corruption.

PX5_MEMORY_MANAGER_VERIFY_ENABLE	When defined, all internal PX5 RTOS memory manager linked-lists are evaluated against the verification code that was established when the memory pool was created and memory is allocated or released. This mechanism helps early detection of memory corruption, most usually associated with the application writing past the allocated memory.
PX5_STACK_VERIFY_ENABLE	When defined, the PX5 RTOS performs stack integrity checking, including verification of the function call return address when possible (when supported by the compiler).

PDV Overhead

The amount of overhead associated with using PDV depends on the compiler, but is generally minimal. Assuming the default verification code generation as described previously, the assembly code to generate the verification code is only a couple of instructions on a typical Arm Cortex-M architecture, as follows:

```
ADDS R3, R2, R0
EORS R3, R3, R4
```

This code assumes that R0 contains the important data value, R2 contains the address to store the verification code, and R3 contains the run-time secret. It's reasonable to assume that each register might require a load instruction (LDR) and there will be one store instruction (STR) to store the code. Given all of that, building and storing the default verification code takes roughly six assembly instructions.

To verify the code, another six assembly instructions are required, along with another three instructions to load the previously stored code, compare it, and branch to either the "okay" path or to the central error handling.

Example PDV Stack Verification

The following is an IAR/Cortex-M example of how PDV can verify the stack (return address) in the pthreads API call *pthread_mutex_lock*. Here is a snippet of the internal PX5 RTOS C implementation of the *pthread_mutex_lock* API:

```
/*
 *
 *  Function:
 *
 *      px5_pthread_mutex_lock
 *
 *
 *  Author:
 *
 *      Bill Lamie, PX5
 *
 *
 *  Description:
 *
 *      This function attempts to get the mutex lock. If the lock is available, this function returns immediately.
 *      Otherwise, if the lock is unavailable, this function blocks the calling thread until it becomes available.
 *
 *
 *  Parameters:
 *
 *      mutex_handle                            Handle of mutex to lock
 *
 *
 *  Return values:
 *
 *      PX5_SUCCESS (0)                         Successful mutex lock
 *      EINVAL                                  Specified mutex handle is invalid
 *
 */

int             px5_pthread_mutex_lock(pthread_mutex_t *  mutex_handle)
{

PX5_STACK_VERIFY_INFO

PX5_PROTECTION_SAVE

px5_mutex_control *     mutex_pointer;
px5_thread_control *    executing_thread;
int                     status;


    /* Check the stack size, if enabled by the application.  */
    PX5_STACK_SIZE_CHECK(PX5_PTHREAD_MUTEX_LOCK_ID)

    /* Setup the stack verification information, if enabled by application.  */
    PX5_STACK_VERIFY_SETUP(PX5_PTHREAD_MUTEX_LOCK_ID)

    /* Pickup the pointer of the mutex.  */
    mutex_pointer =  (px5_mutex_control *) mutex_handle -> internal_mutex_control;
```

At this point in the function, the thread stack looks like the following:

Current stack pointer ->	0x00000f0d
	0x9d913aa9
	0x2002823c
	0x20000f7c
	0x00000000
	0x20029198
	0x2000000c
Beginning of function's stack ->	0x00000f0d

The return address of this function call is in blue, and has the value of 0x00000f0d. The verification code for this thread stack – based on the return address – is in red (0x9d913aa9).

```
            /* Return an error indicating the thread already owns the mutex.  */
            status =  EDEADLK;
        }
    }
    else
    {

        /* The mutex is not available, block until it becomes available.  */
        px5_internal_thread_suspend(executing_thread, PX5_THREAD_MUTEX_SUSPENDED, &(mutex_pointer -> suspended_list), &(mutex
    }

    /* End and restore prior protection posture.  */
    PX5_PROTECTION_END

    /* Verify stack integrity if enabled by application.  */
    PX5_STACK_VERIFY(PX5_PTHREAD_MUTEX_LOCK_ID)

    /* Return completion status to caller.  */
    return(status);
}
```

Before the function returns, the verification code is recalculated and compared against the stored value in red (0x9d913aa9). If they match, the processing returns normally to the caller via the saved return address. If the verification code does not match, the central error processing of the PX5 RTOS is called to alert the application of a stack corruption.

Summing Up

The PDV technology of the PX5 RTOS is a useful tool to help improve device safety and security. The temporal and spatial properties make it quite effective. Its low overhead makes using it practical. However, it is but a powerful part of the device's overall defense in depth strategy.

Enhance • Simplify • Unite

11440 West Bernardo Court • Suite 300
San Diego, CA 92127, USA

Phone: +1 (858) 753-1715
Email: info@px5rtos.com
Website: px5rtos.com