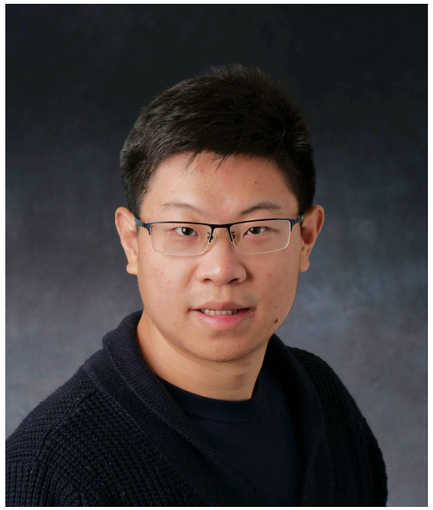
Kill Latest MPU-based Protections in Just One Shot: Targeting All Commodity RTOSes

Speaker: Minghao Lin

Who We Are

Minghao Lin, Professional Research Assistant, , University of Colorado Boulder



Yueqi Chen, Assistant Professor, University of Colorado Boulder

Zicheng Wang, Professional Research Assistant, University of Colorado Boulder

Who We Are

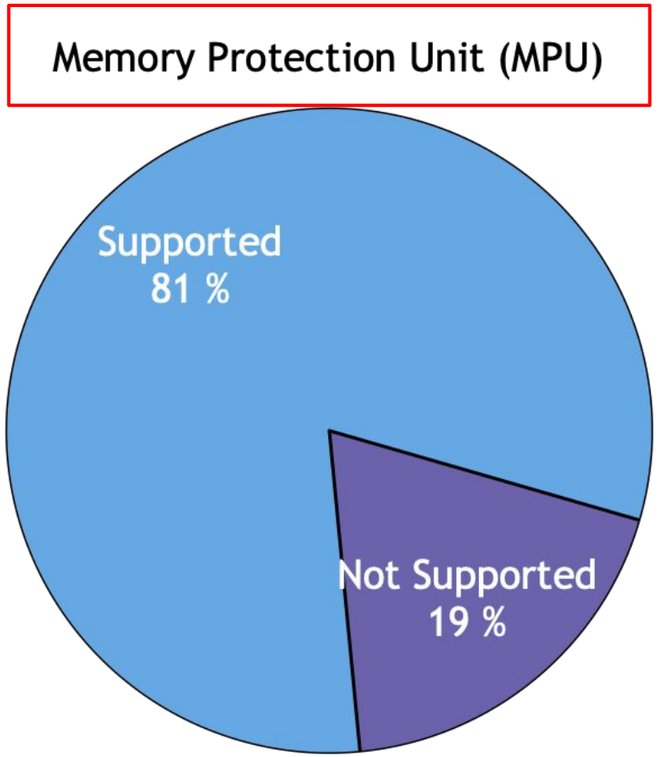
Minghang Shen,
Independent Security
Researcher

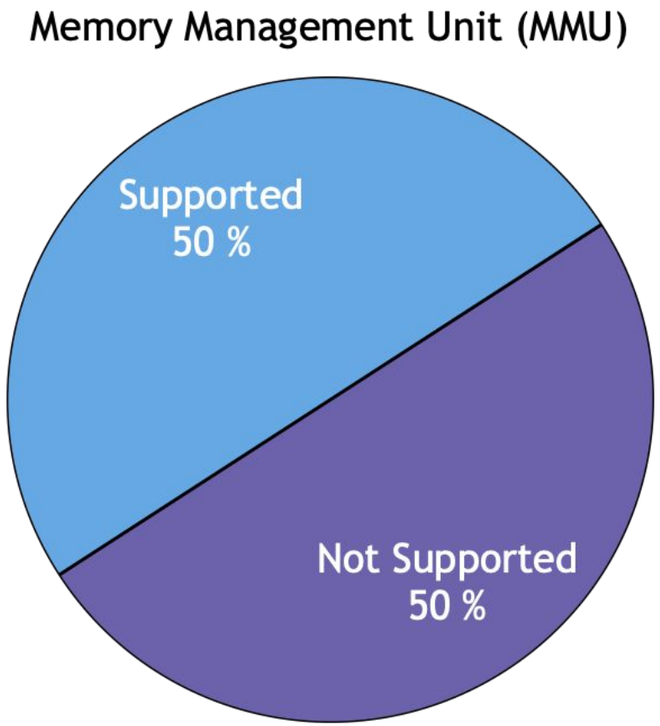
Chaoyang Lin,
Independent Security
Researcher

Jiahe Wang,
Independent Security
Researcher

Real Time Operating Systems Are Everywhere

MPU is Commonly Found in RTOSes





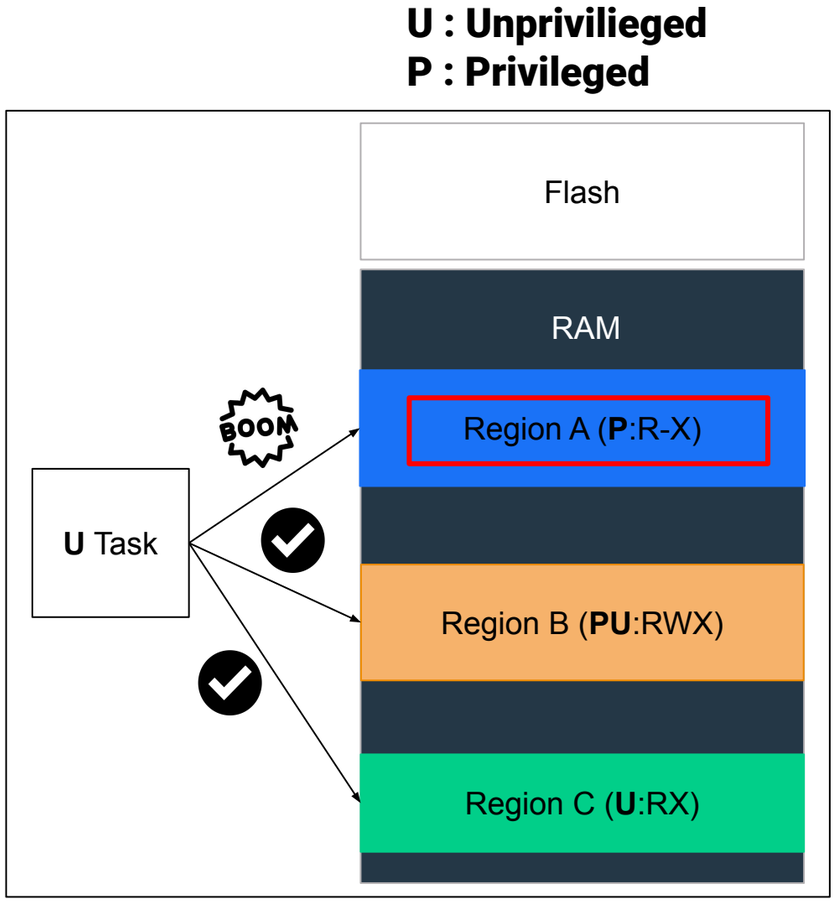
Source: Abbasi, Ali, et al. "Challenges in designing exploit mitigations for deeply embedded systems." *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019.

Memory Protection Unit (MPU)

- Hardware feature commonly found in microcontrollers and processors

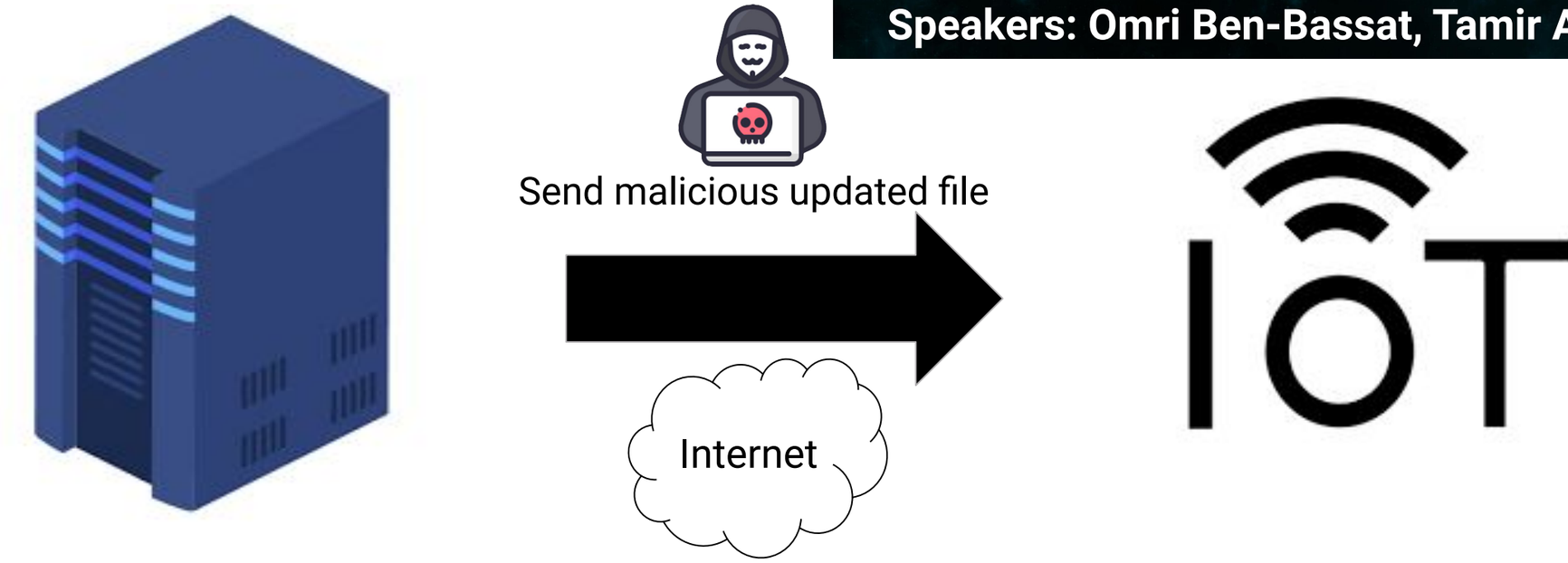
- Functionality
  - Manage the access **permissions** and attributes, e.g., R/W of different regions in memory according execution state, i.e., **Privileged (P) or Unprivileged (U)**

  - **Fault occurs** when access permission is violated



An Exploitation Case

- Over The Air Update



An Exploitation Case

- Vulnerability Details

1. `GetEntireFile()` function is used to parse the file sent through Internet

2. `FileSize` could be very large before malloc, causing integer overflow and thereby a **small allocated memory**

3. Followed by **Heap overflow** caused by `memcpy`

```
int16_t GetEntireFile(uint8_t *pRecvBuf,
                      int16_t RecvBufLen,
                      int16_t *ProcessedSize,
                      uint32_t FileSize,
                      char **pFile)
{
    int16_t copyLen = 0;
    static bool firstRun = TRUE;
    static int16_t TotalRecvBufLen = 0;

    if(firstRun)
    {
        TotalRecvBufLen = RecvBufLen;
        firstRun = FALSE;
        if(TotalRecvBufLen < FileSize)
        {
            /* Didn't receive the entire file in the first run. */
            /* Allocate a buffer in the size of the entire file and fill
               it in each round. */
            pTempBuf = (char*)malloc(FileSize + 1);
            if(pTempBuf == NULL)
            {
                /* Allocation failed, return error. */
                return(-1);
            }
            memcpy(pTempBuf, (char *)pRecvBuf, RecvBufLen);
            *ProcessedSize = RecvBufLen;
```

Integer Overflow

Heap Overflow

An Exploitation Case

- Find Function Pointer to Overwrite

- `httpGetHandler` function is used to handle different types of http requests

- `httpRequest` is an array of http handler function pointers

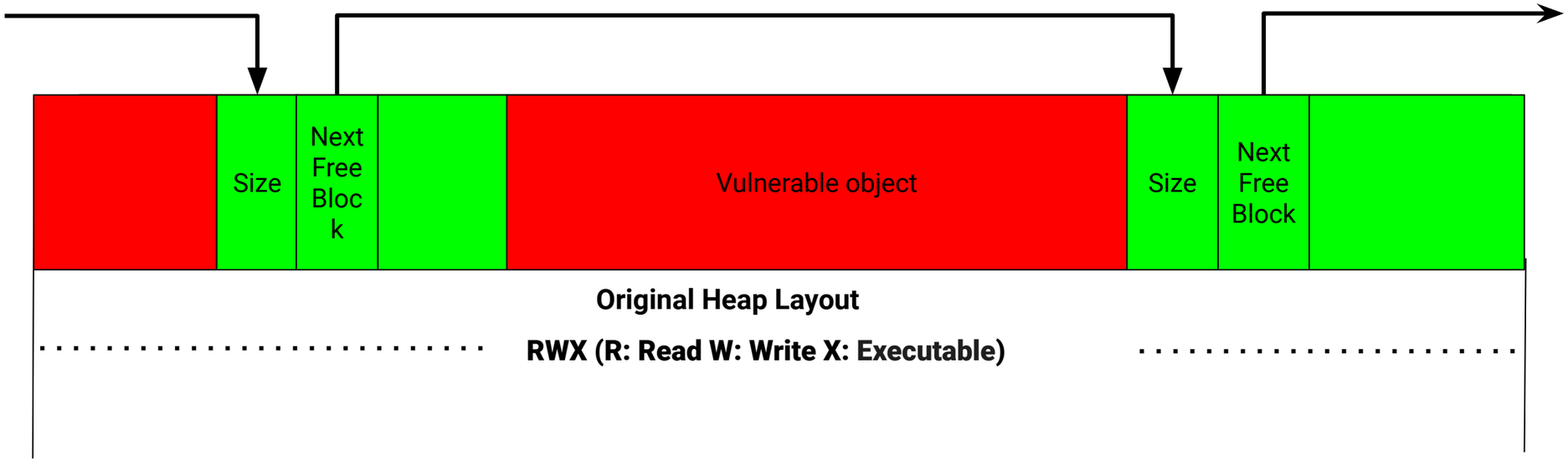
- Overwrite function pointer of the arrary to point shellcode

```
void httpGetHandler(SlNetAppRequest_t *netAppRequest)
{
    uint16_t metadataLen;
    int32_t status;
    uint8_t requestIdx;

    uint8_t argcCallback;
    uint8_t     *argvArray;
    uint8_t     **argvCallback = &argvArray;

    argvArray = gHttpGetBuffer;

    status = httpCheckContentInDB(  netAppRequest,
                                    &requestIdx,
                                    &argcCallback,
                                    argvCallback);

    if(status < 0)
    {
        metadataLen =
            prepareGetMetadata(status, strlen (
                                    (const char *)pageNotFound),
                                HttpContentTypeList_TextHtml);

        sl_NetAppSend (netAppRequest->Handle, metadataLen, gMetadataBuffer,
                       (SL_NETAPP_REQUEST_RESPONSE_FLAGS_CONTINUATION |
                        SL_NETAPP_REQUEST_RESPONSE_FLAGS_METADATA));
        INFO_PRINT("[Link local task] Metadata Sent, len = %d \n\r",
                   metadataLen);

        sl_NetAppSend (netAppRequest->H
                       strlen(
                          (const cha                Found), (uint8_t *)pageNotFound,
                       0); /* mark a                ent */
        INFO_PRINT("[Link local task]                t, len = %d\n\r",
                   strlen ((const char *)pageNotFound));
    }
    else
    {
        httpRequest[requestIdx].serviceCallback(requestIdx, &argcCallback,
                                                argvCallback,
                                                netAppRequest);
    }
```

Http handler function pointers

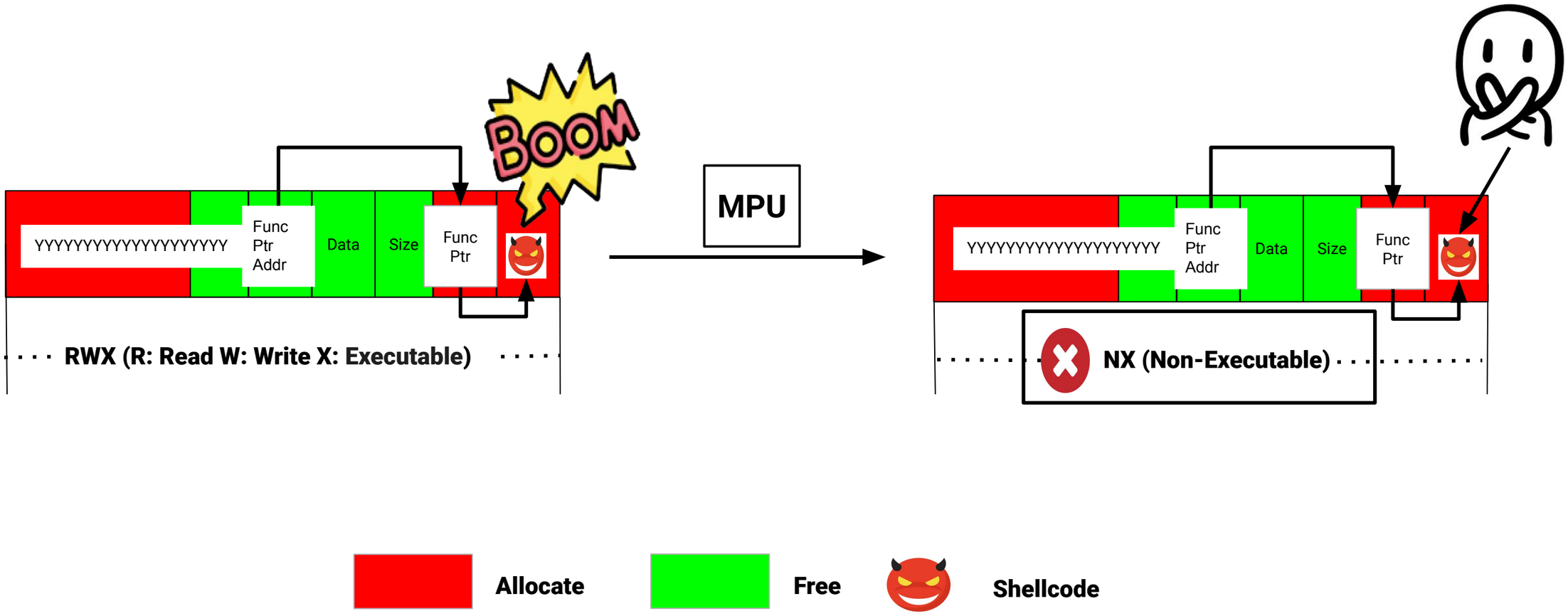
An Exploitation Case

- Heap Layout



An Exploitation Case

- MPU Disables this Exploitation




MPU becomes a terminator for exploitation?

FreeRTOS™

TIZEN™

THREADX®

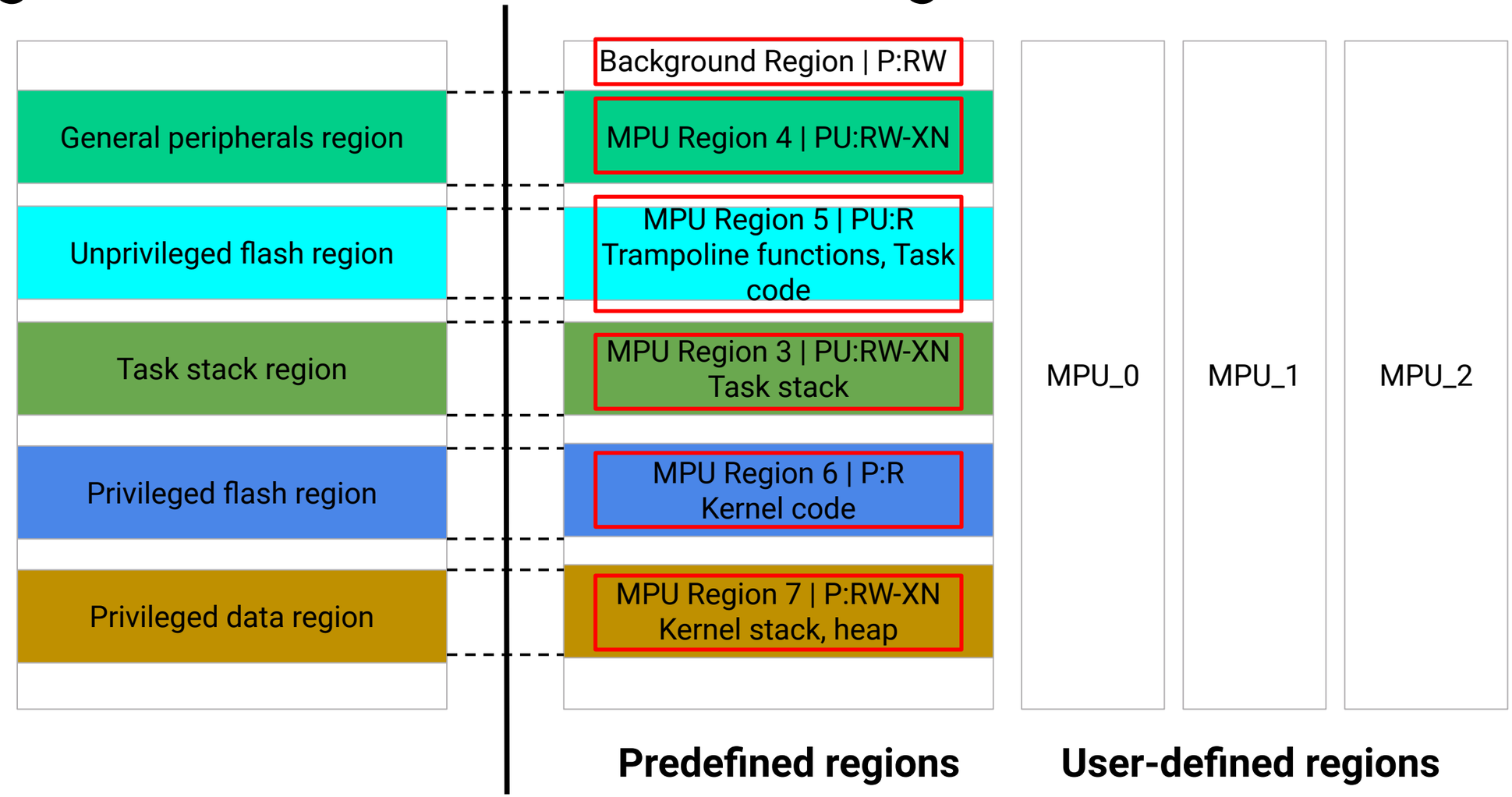
RT-Thread

Zephyr™

arm MBED


You first!
freeRTOS
TM

Privilege Isolation In FreeRTOS Using MPU

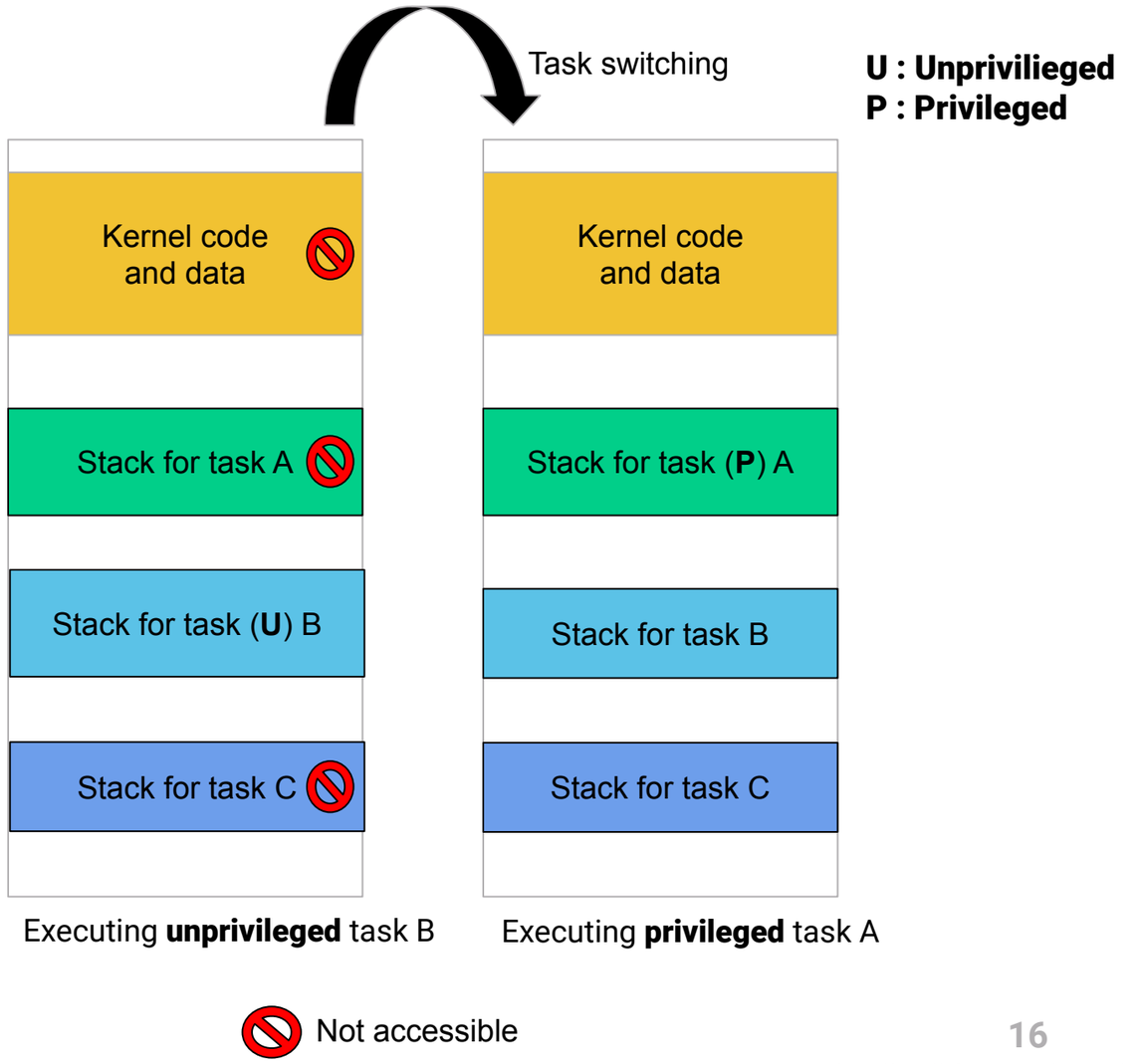


MPU region definitions of ARM-CM3 FreeRTOS-MPU

Memory View Per Task

1. Every Task has their own access permission and execution state

2. When task switching happens, MPU configuration will be changed to the specific task

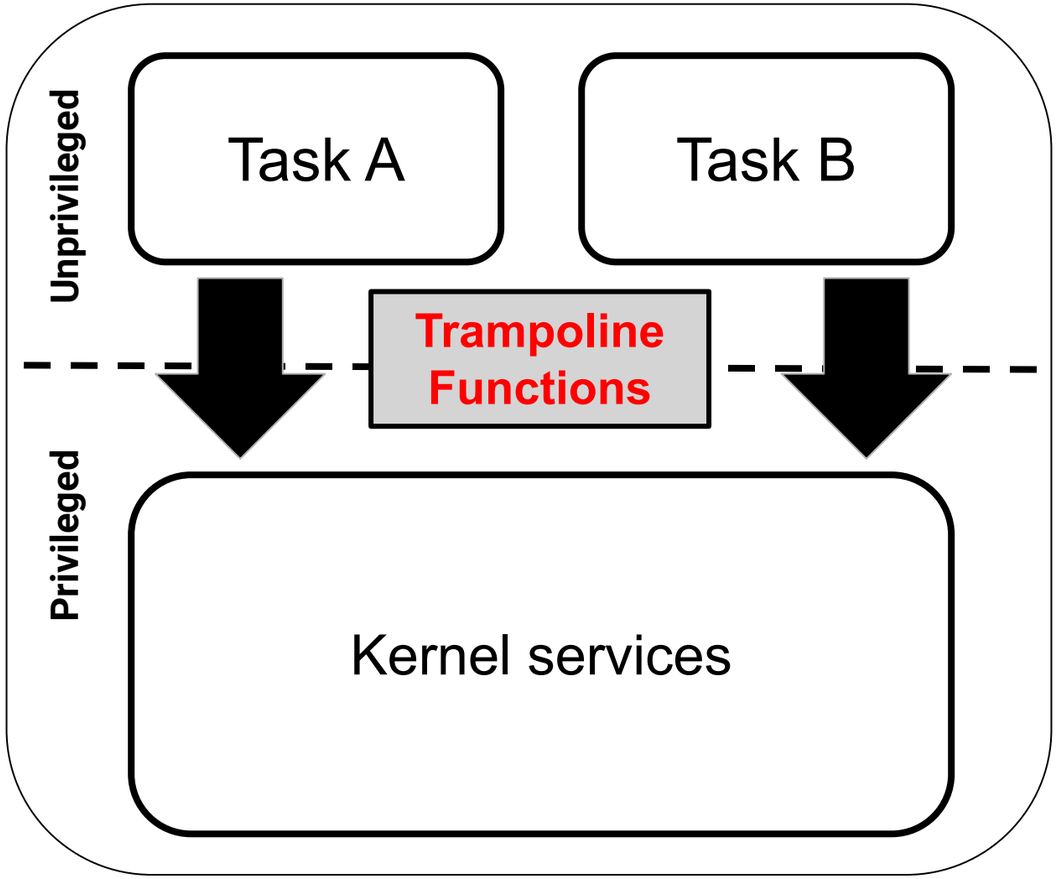


Issue 1 Missing Legitimacy Check During Mode Switch

- Overview of Trampoline Function

- In FreeRTOS, **kernel functions** are wrapped by **trampoline functions** with “MPU_” prefix, which play the role as a trampoline for switching from user mode to kernel mode

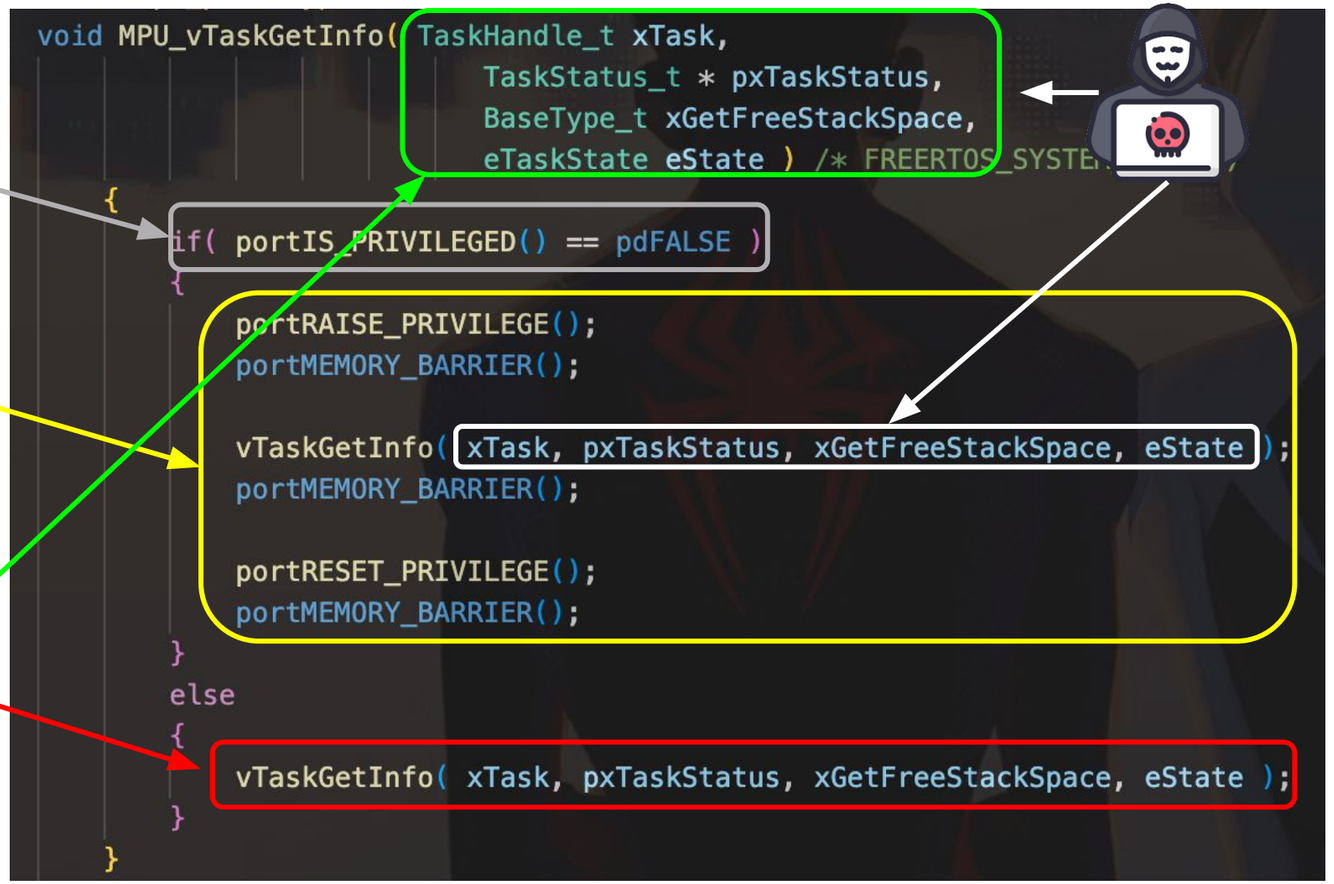
- Non-privileged tasks can call these trampoline functions to request kernel service



Issue 1 Missing Legitimacy Check During Mode Switch

- Implementation of Trampoline Function

1. Check if current execution state is **privileged or not**
2. If not, it will **raise privilege**, then call the kernel function. Finally, it will **drop privilege**
3. If current execution state is privileged, it will directly call kernel function
4. **No check** for parameters of `MPU_vTaskGetInfo`

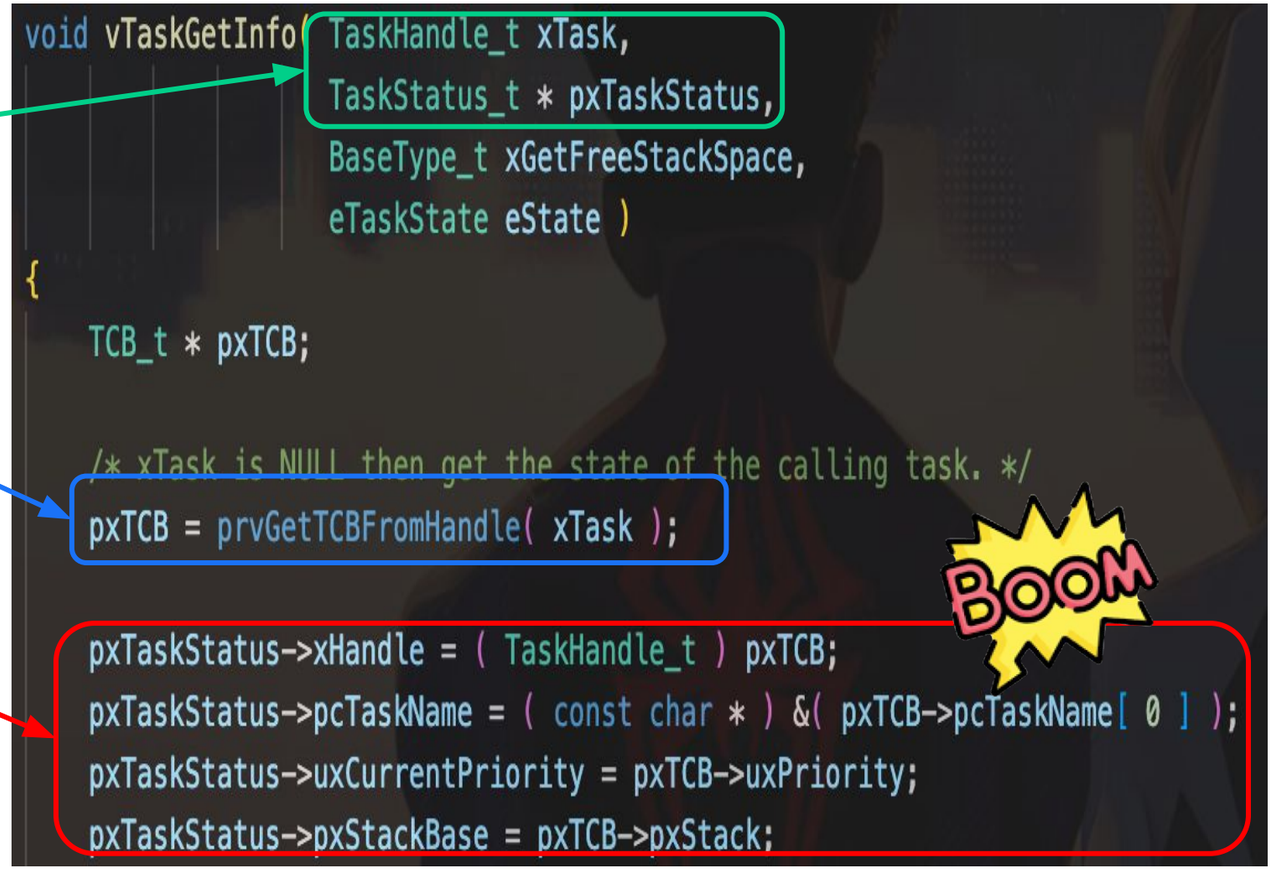


All trampoline functions are in FreeRTOS-Kernel/include/mpu_prototype.h

Issue 1 Missing Legitimacy Check During Mode Switch

- Arbitrary Read or Write in vTaskGetInfo

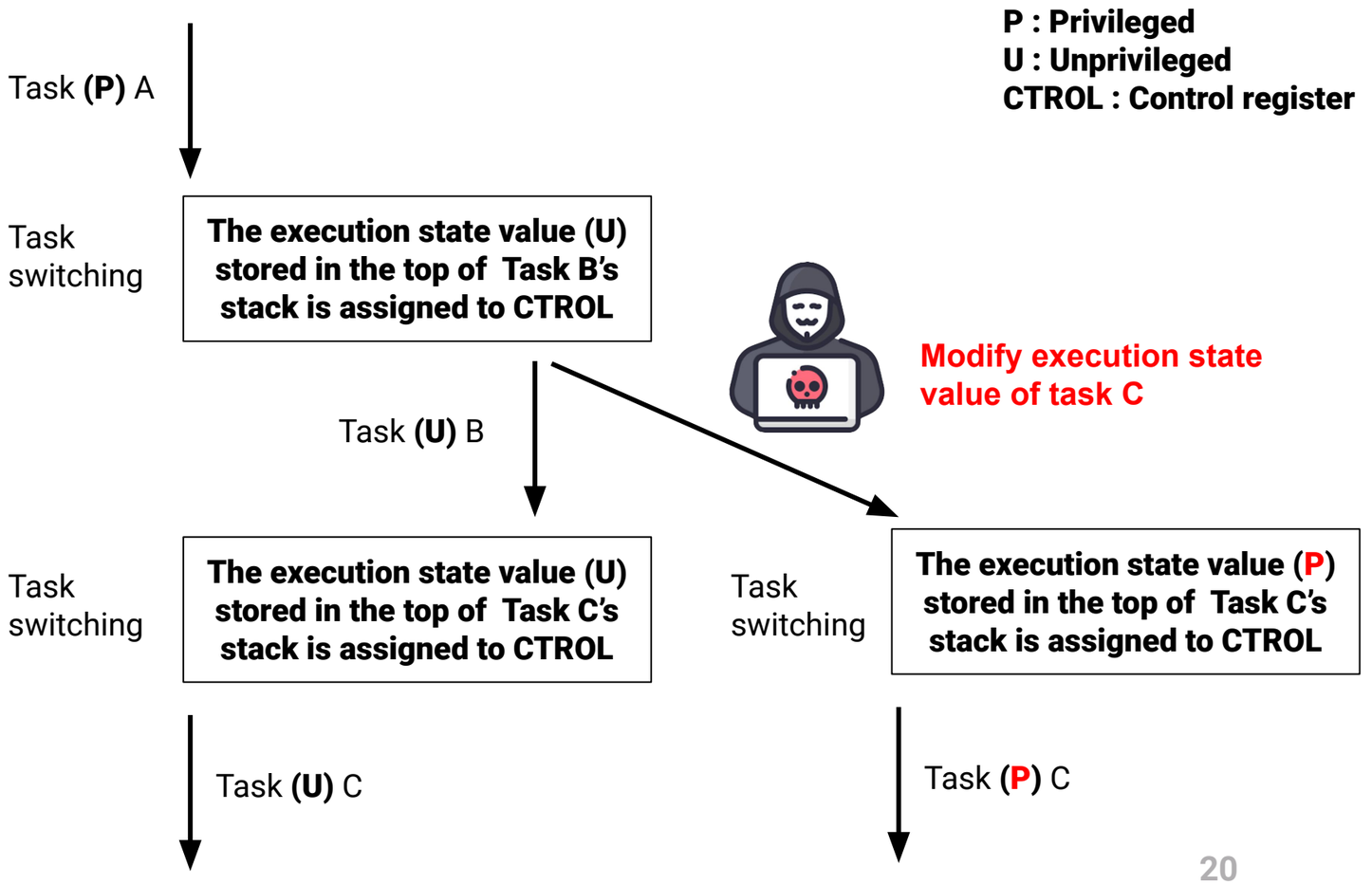
1. Unprivileged task can pass **two arbitrary pointers** to parameters `xTask` and `pxTaskStatus`
2. Then, `pxTCB` is later assigned as xTask
3. `pxTaskStatus` and `pxTCB` is dereferenced → **arbitrary read from or write to any pointers**



Issue 1 Missing Legitimacy Check During Mode Switch

- Privilege Escalation

- A task is privileged or not depends on the value stored in top of its stack
- When task switching happens, CTROL will be set to the execution state of the next task
- Leverage arbitrary write to modify the execution state value to be privileged



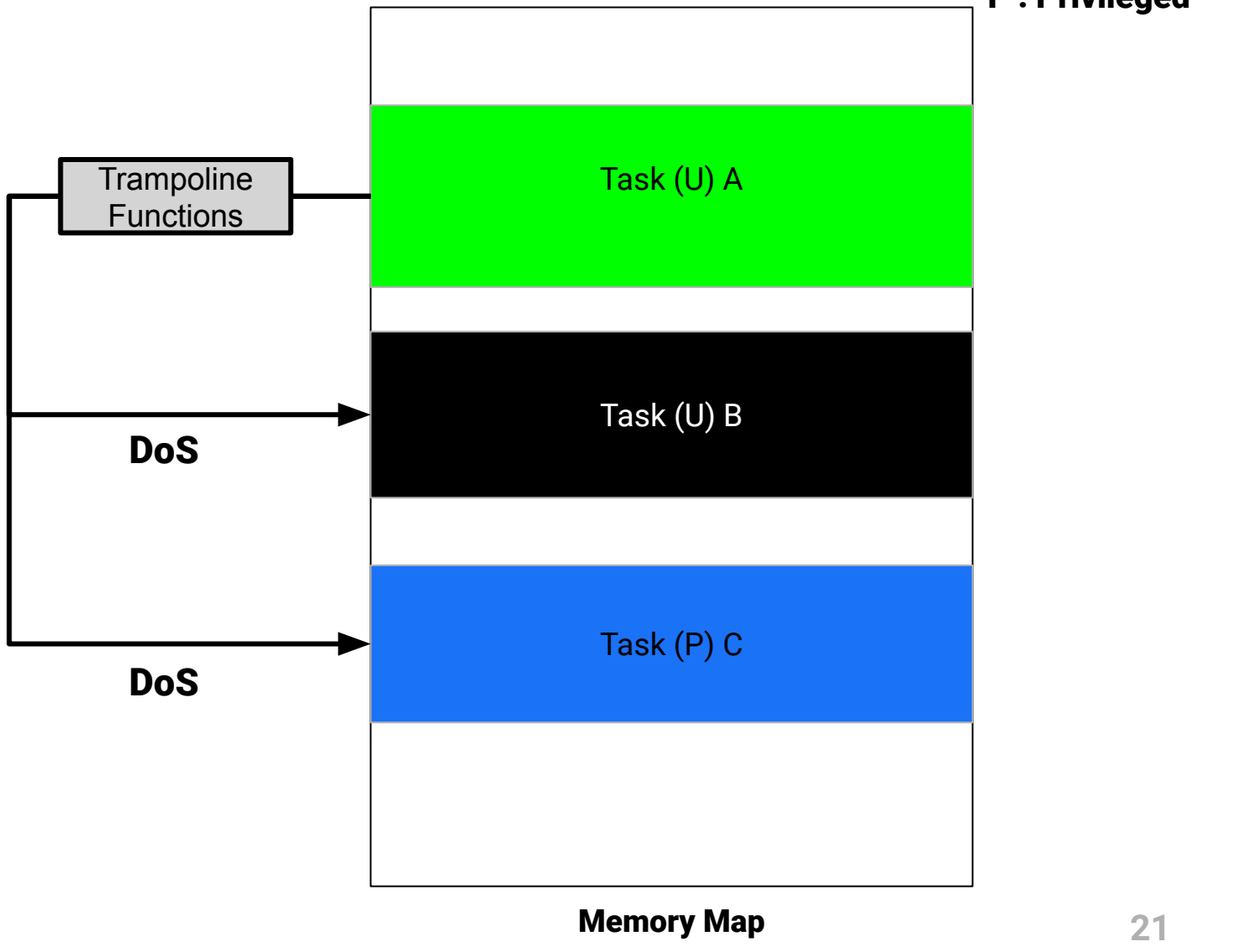
Issue 1 Missing Legitimacy Check During Mode Switch

- Trampoline Functions DoS Other Tasks

```
void MPU_vTaskSuspend( TaskHandle_t pxTaskToSuspend ) /* FREERTOS_SYST
    {
        if( portIS_PRIVILEGED() == pdFALSE )
        {
            portRAISE_PRIVILEGE();
            portMEMORY_BARRIER();

            vTaskSuspend( pxTaskToSuspend );
            portMEMORY_BARRIER();

            portRESET_PRIVILEGE();
            portMEMORY_BARRIER();
        }
        else
        {
            vTaskSuspend( pxTaskToSuspend );
        }
    }
```

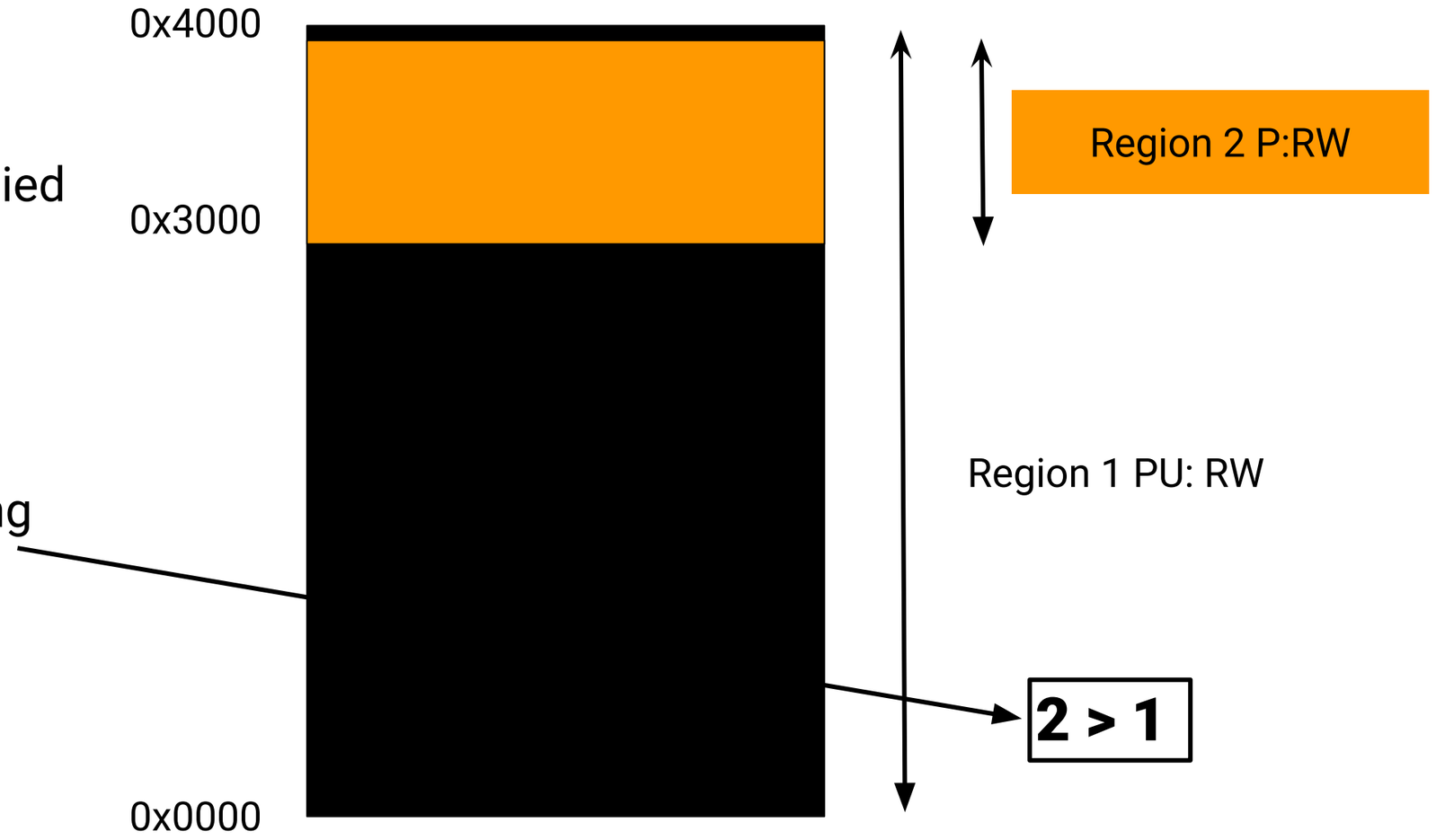
TaskControlBlock pointer

Suspend other tasks



MPU Region Overlapping

- The two regions have different permissions, the permissions associated with region 2 are applied
- For overlapping regions, a fixed priority scheme determines attributes and permissions for memory access to the overlapping region



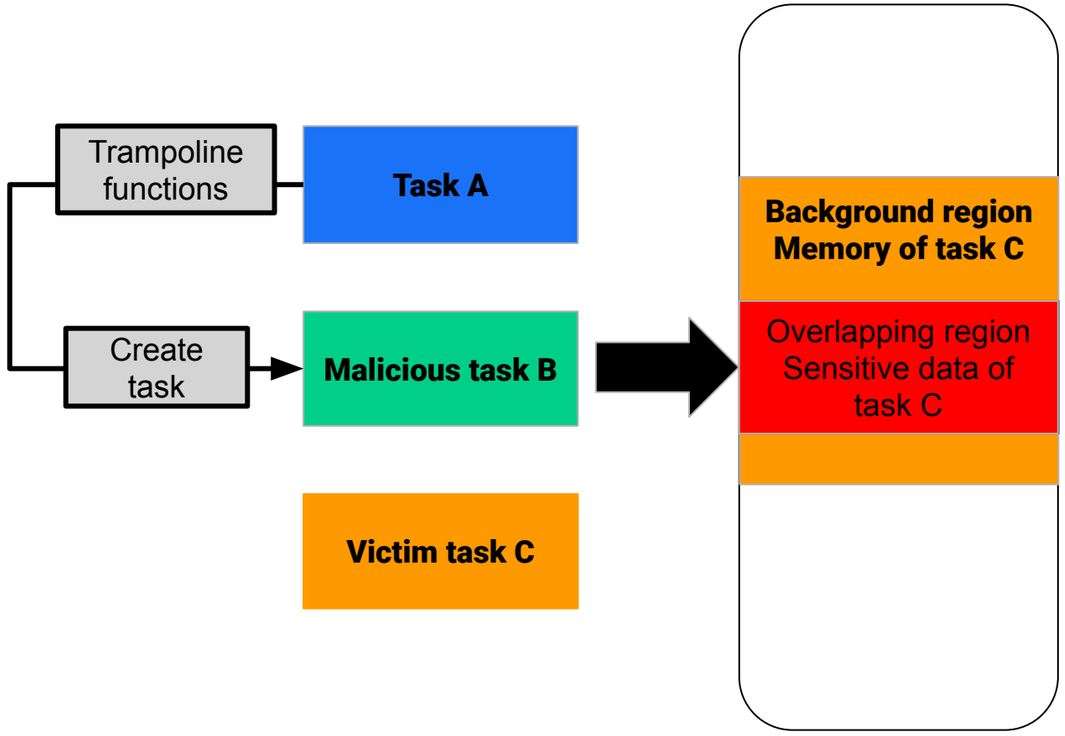
Issue 2 Mistaken MPU Configuration

MPU Region 0~2 are user-defined MPU regions

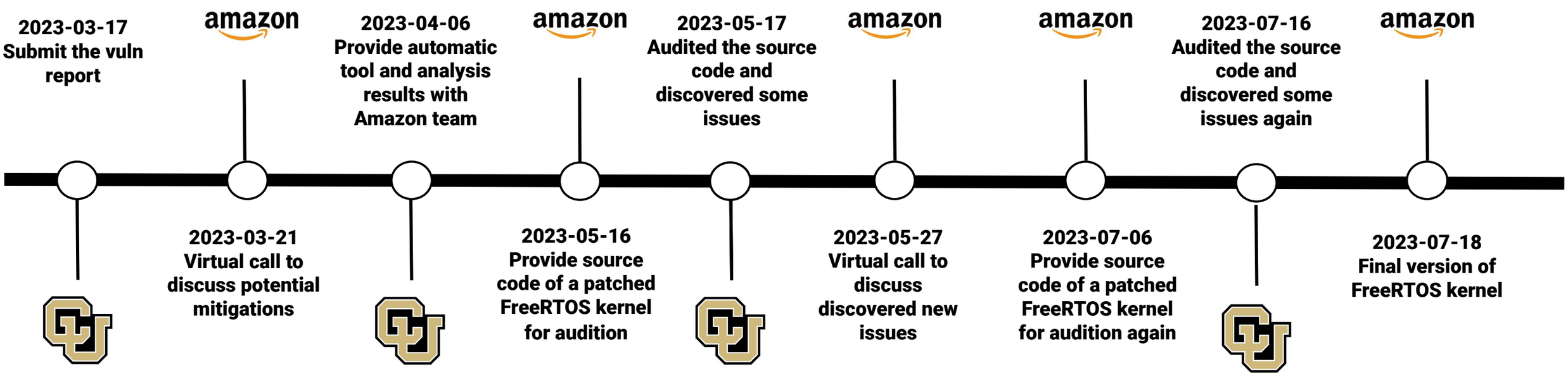
1. When creating a child task, the parent task can configure MPU 0-2 regions of child task

2. Unfortunately, the FreeRTOS kernel doesn't examine if this configuration has conflict with other tasks, resulting in memory overlapping between tasks

3. Adversaries can exploit this mistake to access the memory of victim tasks, stealing or tampering critical data

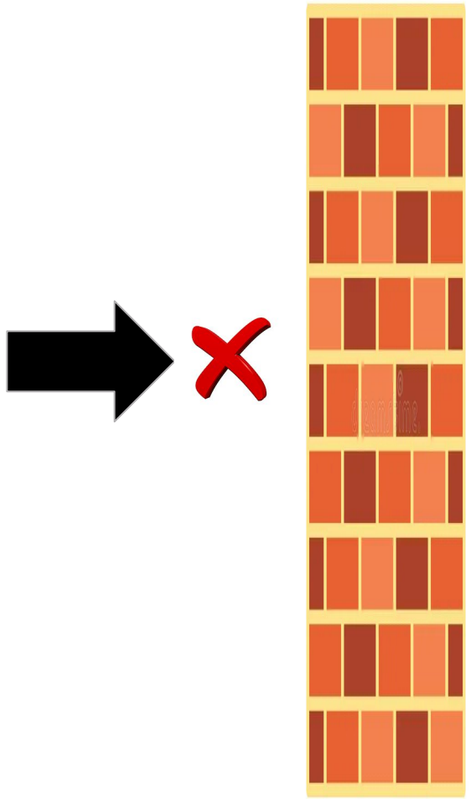


Report to Amazon Team And Got Response



Amazon Team Mitigations for Fixing These Issues

- Limited Trampoline Functions



- MPU_xQueueCreateMutex
- MPU_xQueueCreateMutexStatic
- MPU_xQueueCreateCountingSemaphore
- MPU_xQueueCreateCountingSemaphoreStatic
- MPU_xQueueGenericCreate
- MPU_xQueueGenericCreateStatic
- MPU_xQueueCreateSet
- MPU_xQueueRemoveFromSet
- MPU_xQueueGenericReset
- MPU_xTaskCreate
- MPU_xTaskCreateStatic
- MPU_vTaskDelete
- MPU_vTaskPrioritySet
- MPU_vTaskSuspendAll
- MPU_xTaskResumeAll
- MPU_xTaskGetHandle
- MPU_xTaskCallApplicationTaskHook
- MPU_vTaskList
- MPU_vTaskGetRunTimeStats
- MPU_xTaskCatchUpTicks
- MPU_xEventGroupCreate
- MPU_xEventGroupCreateStatic
- MPU_vEventGroupDelete
- MPU_xStreamBufferGenericCreate

Amazon Team Mitigations for Fixing These Issues

- Added Function For Checking Access Permissions And Buffer Ranges

Check if the memory is in MPU region and access permission of the memory is violated by looking up MPU settings

```
BaseType_t xPortIsAuthorizedToAccessBuffer( const void * pvBuffer,
                                            uint32_t ulBufferLength,
                                            uint32_t ulAccessRequested ) /* PRIVILEGED_FUNCTION */
{
    uint32_t i, ulBufferStartAddress, ulBufferEndAddress;
    BaseType_t xAccessGranted = pdFALSE;
    const xMPU_SETTINGS * xTaskMpuSettings = xTaskGetMPUSettings( NULL ); /* Calling task's MPU settings. */

    if( ( xTaskMpuSettings->ulTaskFlags & portTASK_IS_PRIVILEGED_FLAG ) == portTASK_IS_PRIVILEGED_FLAG )
    {
        xAccessGranted = pdTRUE;
    }
    else
    {
        if( portADD_UINT32_WILL_OVERFLOW( ( ( uint32_t ) pvBuffer ), ( ulBufferLength - 1UL ) ) == pdFALSE )
        {
            ulBufferStartAddress = ( uint32_t ) pvBuffer;
            ulBufferEndAddress = ( ( ( uint32_t ) pvBuffer ) + ulBufferLength - 1UL );

            for( i = 0; i < portTOTAL_NUM_REGIONS; i++ )
            {
                /* Is the MPU region enabled? */
                if( ( xTaskMpuSettings->xRegionsSettings[ i ].ulRLAR & portMPU_RLAR_REGION_ENABLE ) == portMPU_RLAR_REGION_ENABLE )
                {
                    if( portIS_ADDRESS_WITHIN_RANGE( ulBufferStartAddress,
                                                     portEXTRACT_FIRST_ADDRESS_FROM_RBAR( xTaskMpuSettings->xRegionsSettings[ i ].ulRBAR ),
                                                     portEXTRACT_LAST_ADDRESS_FROM_RLAR( xTaskMpuSettings->xRegionsSettings[ i ].ulRLAR ) ) &&
                        portIS_ADDRESS_WITHIN_RANGE( ulBufferEndAddress,
                                                     portEXTRACT_FIRST_ADDRESS_FROM_RBAR( xTaskMpuSettings->xRegionsSettings[ i ].ulRBAR ),
                                                     portEXTRACT_LAST_ADDRESS_FROM_RLAR( xTaskMpuSettings->xRegionsSettings[ i ].ulRLAR ) ) &&
                        portIS_AUTHORIZED( ulAccessRequested,
                                           prvGetRegionAccessPermissions( xTaskMpuSettings->xRegionsSettings[ i ].ulRBAR ) ) )
                    {
                        xAccessGranted = pdTRUE;
                        break;
```

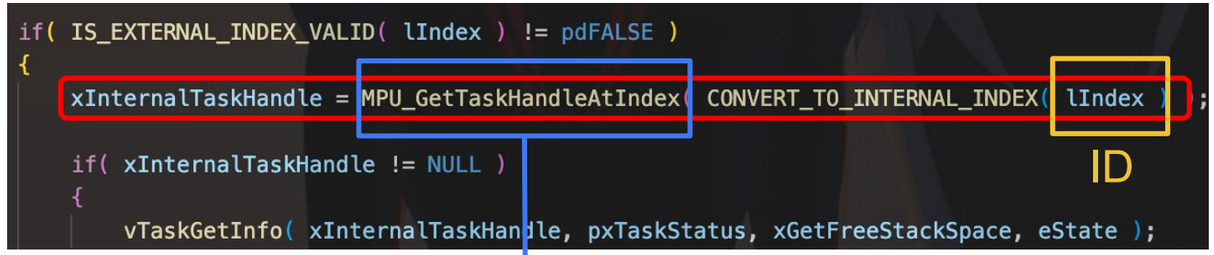
Accessed memory, memory size and access operation read/write

Added Function

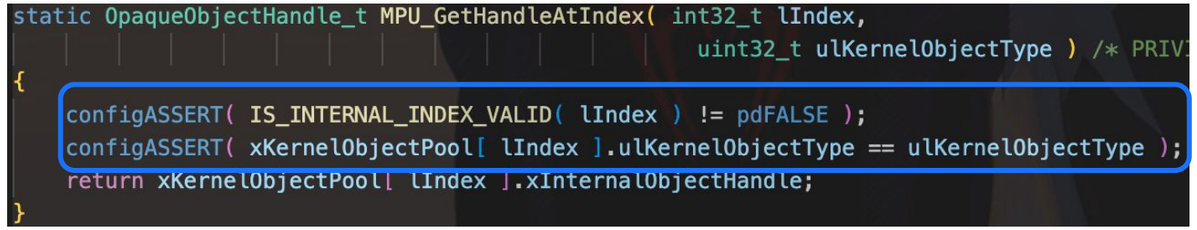
Amazon Team Mitigations for Fixing These Issues

- Replace Object Pointer with Object ID

1. Trampoline functions retrieve objects via ID rather than a raw pointer value



2. Check if the type of object to be retrieved and object ID is valid, if pass check, return an object from the object pool

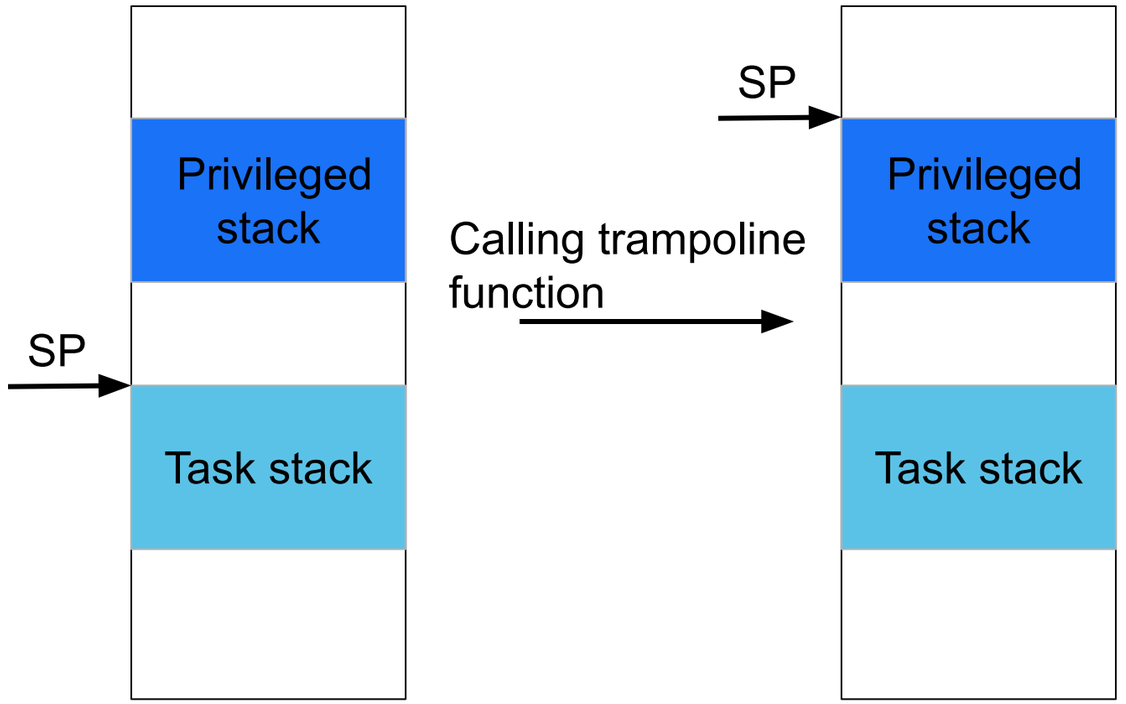


Amazon Team Mitigations for Fixing These Issues

- Adjust The Location of Context && Privileged Stack for Trampoline Functions

1. The task context including execution state value is now stored in TCB which is accessible to privileged code only

2. The trampoline function are now executed on a separate privileged only stack. When a task calls trampoline function, the stack pointer register will change from task stack to privileged only stack.



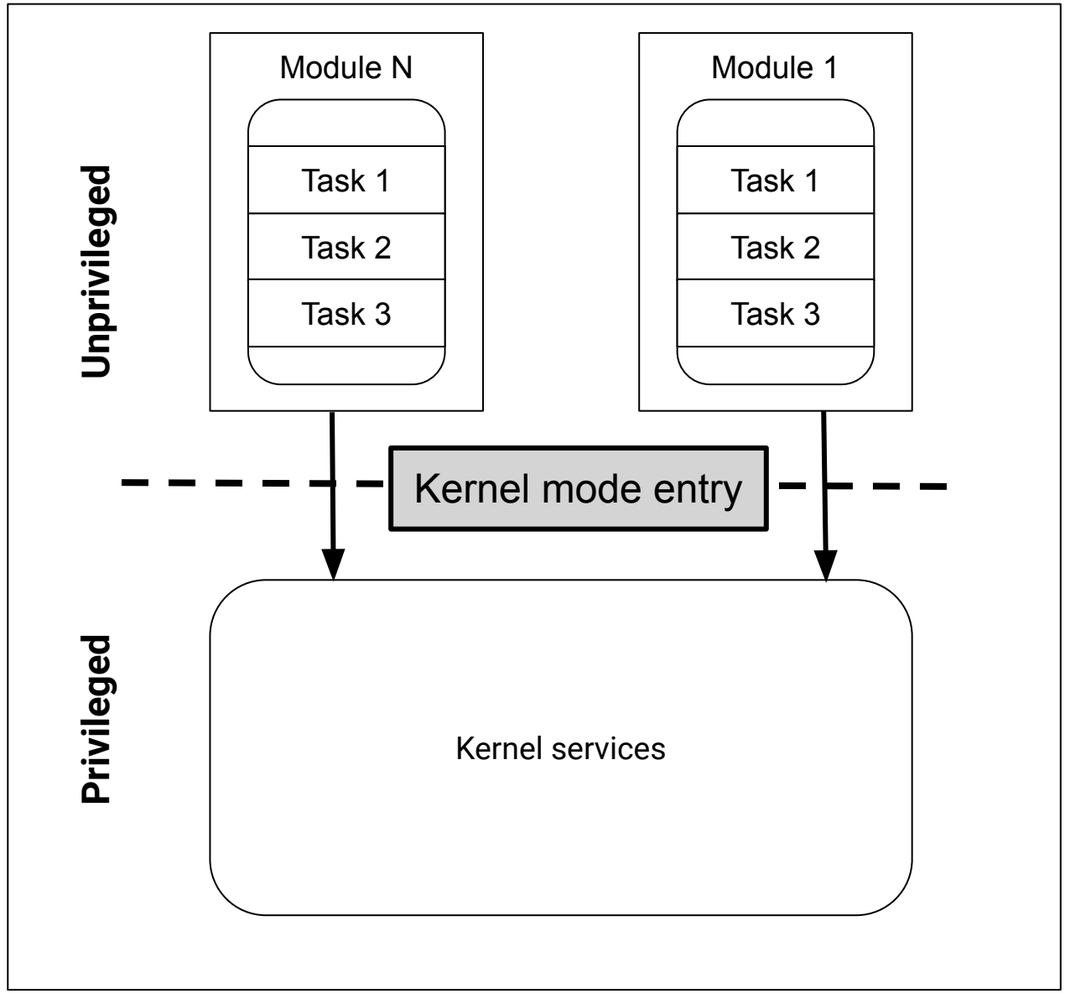
How about Other RTOSes?


You next!
THREADX

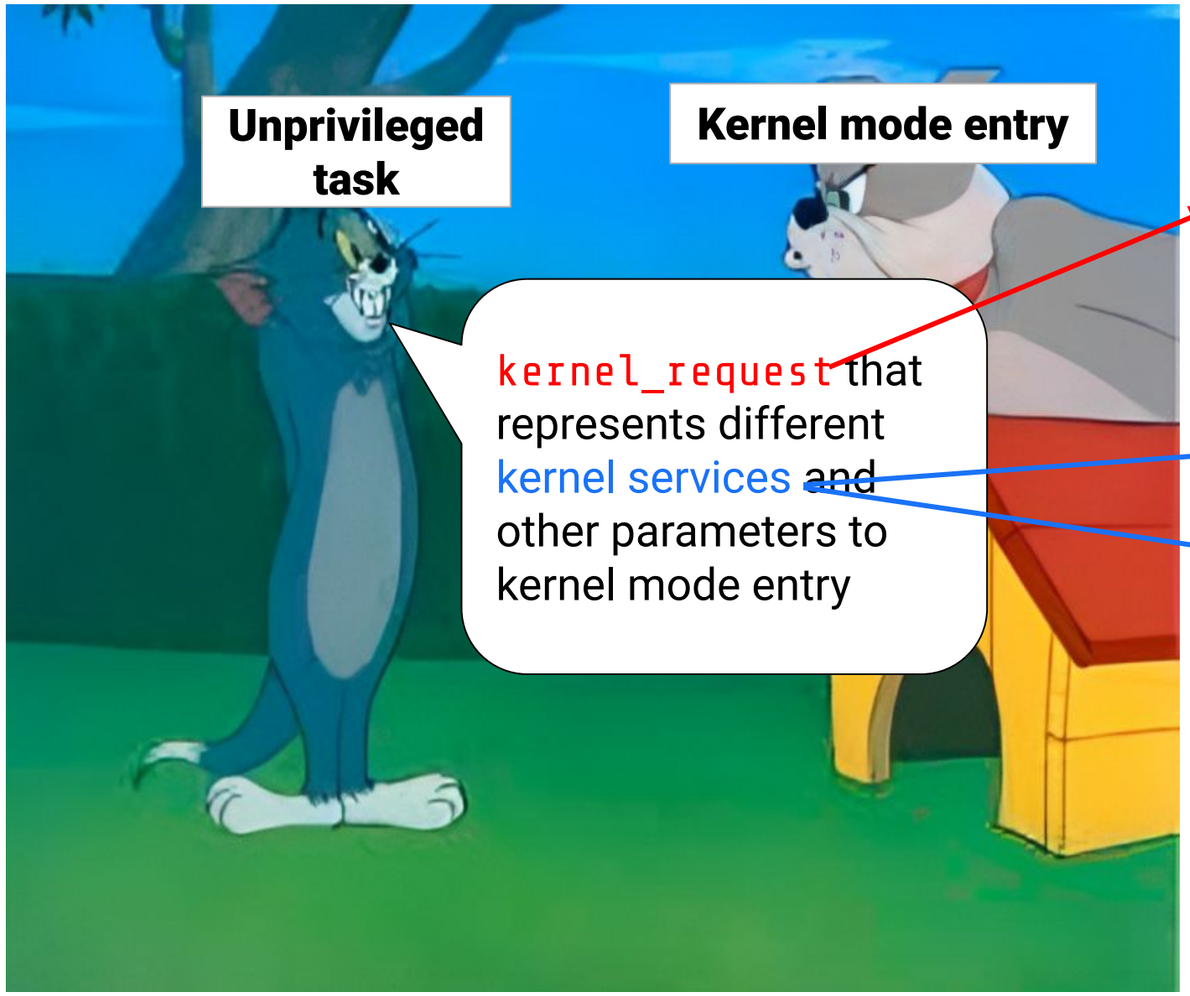
Module Concept in ThreadX

- The smallest unit of memory management is a module which comprises a set of tasks
  - MPU 5, 6, 7 for module data
  - MPU 1, 2, 3, 4 for module code
  - MPU 0 for kernel mode entry

- Similar to FreeRTOS, unprivileged tasks call kernel mode entry to request kernel services

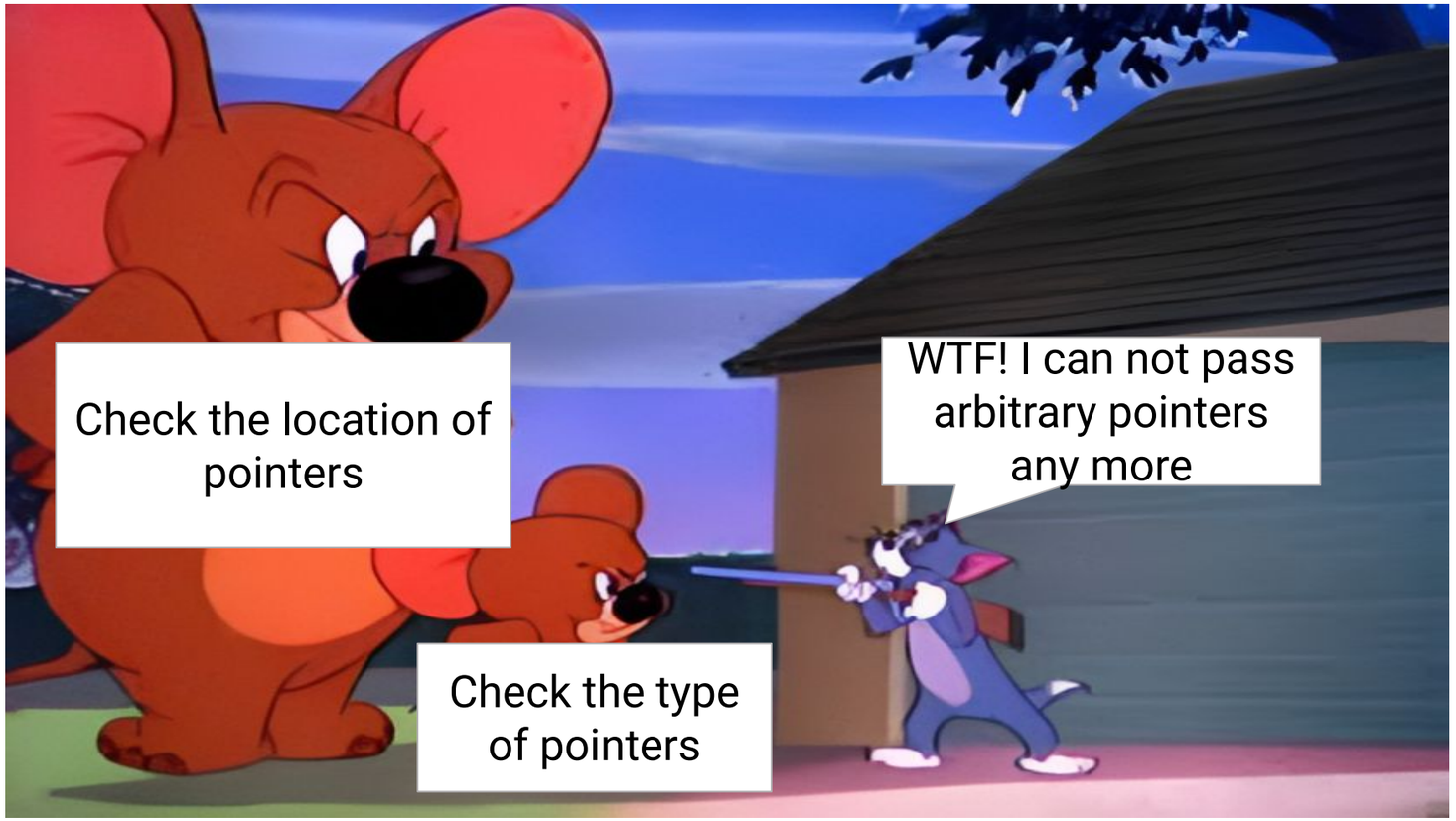


Trampoline Functions in ThreadX




```
ALIGN_TYPE _txm_module_manager_kernel_dispatch(ULONG
kernel_request, ALIGN_TYPE param_0, ALIGN_TYPE param_1,
ALIGN_TYPE param_2)
{
    switch (kernel_request)
    {
    case TXM_BLOCK_ALLOCATE_CALL:
        _txm_module_manager_tx_block_allocate_dispatc
     h(...);
    case TXM_BLOCK_POOL_CREATE_CALL:
        _txm_module_manager_tx_block_pool_create_disp
     atch(...);
     . . .
}
```

Trampoline Functions' Checks in ThreadX



Trampoline Functions' Checks in ThreadX (cont.)

```
static ALIGN_TYPE _txm_module_manager_tx_thread_delete_dispatch(TXM_MODULE_INSTANCE *module_instance, ALIGN_TYPE param_0)
{
                                                    /* UINT _txe_thread_delete(
                                                        TX_THREAD *thread_ptr -> param_0
                                                    ); */
ALIGN_TYPE return_value;

    if (module_instance -> txm_module_instance_property_flags & TXM_MODULE_MEMORY_PROTECTION)
    {
        if (!TXM_MODULE_MANAGER_PARAM_CHECK_OBJECT_FOR_USE(module_instance, param_0, sizeof(TX_THREAD)))
            return(TXM_MODULE_INVALID_MEMORY);
    }

    return_value = (ALIGN_TYPE) _txe_thread_delete(
        (TX_THREAD *) param_0
    );

    /* Deallocate object memory.  */
    if (return_value == TX_SUCCESS)
    {
        return_value = _txm_module_manager_object_deallocate((VOID *) param_0);
    }
    return(return_value);
}
```

Function definition

Check if the thread_ptr is in kernel space

Check if the thread_ptr is valid

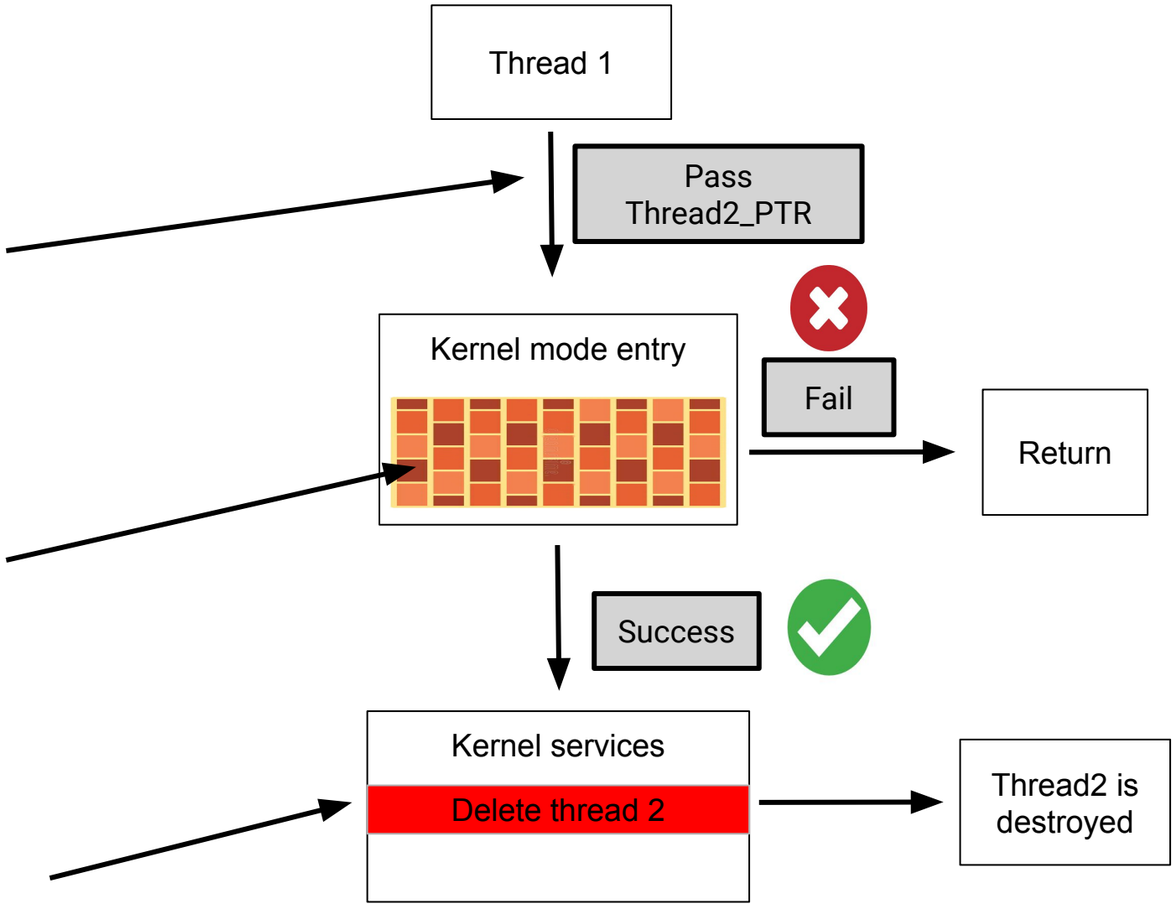
```
/* Now check for invalid thread ID.  */
else if (thread_ptr -> tx_thread_id != TX_THREAD_ID)
{

    /* Thread pointer is invalid, return appropriate error code.  */
    status =  TX_THREAD_ERROR;
}
```

```
/* Define thread control
specific data definitions.  */

#define TX_THREAD_ID    ((ULONG) 0x54485244)
```

Is Trampoline Function in ThreadX Really Secure?

An Illustrative Example

1. Malicious thread 1 pass the pointer of thread2 handler to kernel mode entry.
2. Check if the PTR is in kernel space and PTR is the TX_THREAD* class based on tx_thread_id
3. Matches the type and location Thread2 is destroyed



Automatic Approach to Identify Similar Issues

- Use CodeQL to Do Code Audition

1. The source is the parameters of trampoline function

2. The sink is assign expression including arithmetic and bitwise operation

3. Add additionalTaint. If A object is taint, field B is also taint after accessing the field B like A.B

```
// find arbitrary write
class MPUTaskArbWCfg extends TaintTracking::Configuration {
    Quick Evaluation: MPUTaskArbWCfg
    MPUTaskArbWCfg () { this = "MPUTaskArbWCfg" }
    Quick Evaluation: isSource
    override predicate isSource(DataFlow::Node source) {
        source.asParameter() instanceof MPUTaskTaintedParament
    }
    Quick Evaluation: isSink
    override predicate isSink(DataFlow::Node sink) {
        exists(AssignArithmeticOperation ao,AssignBitwiseOperation ag,AssignExpr ae|
            ae.getLValue().getAChild*() = sink.asExpr()
            or ag.getLValue().getAChild*() = sink.asExpr()
            or ao.getLValue().getAChild*() = sink.asExpr()
        )
    }
    //this ignore the number
    Quick Evaluation: isSanitizer
    override predicate isSanitizer(DataFlow::Node node) {
        not node.getType().getUnderlyingType() instanceof PointerType
      }
    Quick Evaluation: isAdditionalTaintStep
    override predicate isAdditionalTaintStep(DataFlow::Node node1, DataFlow::Node node2) {
        exists(FieldAccess fa |
            fa = node2.asExpr()
            and node1.asExpr() = fa.getQualifier()
```

Results From Automation

- We found 43 trampoline functions causing arbitrary write, 29 trampoline function causing arbitrary read, 23 trampoline function causing other security issues

- We have released our CodeQL script and result of automation in GitHub

- Git link: https://github.com/MinghaoLin2000/TrampolineFuncAnalyzer4FreeRTOS

#	source	[1] ▲	sink
1	xTask	MPU_ulTaskGenericNotifyValueClear	pxTCB
2	xTask	MPU_uxTaskGetStackHighWaterMark	pucStackByte
3	xTask	MPU_uxTaskGetStackHighWaterMark2	pucStackByte
4	xEventGroup	MPU_vEventGroupDelete	pxBlockToInsert
5	xEventGroup	MPU_vEventGroupDelete	pxNextFreeBlock
6	xEventGroup	MPU_vEventGroupDelete	pxBlockToInsert
7	xEventGroup	MPU_vEventGroupDelete	puc
8	xEventGroup	MPU_vEventGroupDelete	pxLink
9	xEventGroup	MPU_vEventGroupDelete	pxNextFreeBlock
10	xEventGroup	MPU_vEventGroupDelete	pxBlockToInsert
11	xEventGroup	MPU_vEventGroupDelete	pxBlockToInsert
12	xEventGroup	MPU_vEventGroupDelete	pxNextFreeBlock
13	xQueue	MPU_vQueueDelete	pxBlockToInsert
14	xQueue	MPU_vQueueDelete	puc
15	xQueue	MPU_vQueueDelete	pxLink
16	xQueue	MPU_vQueueDelete	pxNextFreeBlock
17	xQueue	MPU_vQueueDelete	pxBlockToInsert
18	xQueue	MPU_vQueueDelete	pxNextFreeBlock
19	xQueue	MPU_vQueueDelete	pxBlockToInsert
20	xQueue	MPU_vQueueDelete	pxNextFreeBlock
21	xQueue	MPU_vQueueDelete	pxBlockToInsert
22	xStreamBuffer	MPU_vStreamBufferDelete	puc
23	xStreamBuffer	MPU_vStreamBufferDelete	pxBlockToInsert
24	xStreamBuffer	MPU_vStreamBufferDelete	pxNextFreeBlock
25	xStreamBuffer	MPU_vStreamBufferDelete	pxBlockToInsert
26	xStreamBuffer	MPU_vStreamBufferDelete	pxNextFreeBlock

Key Takeaway: Comparison Among Different RTOSes

Issue \ RTOS	ThreadX	arm MBED	TIZEN	RT-Thread	Zephyr
Missing/Incomplete Check	☹	☹	☹	☹	☺
Mistaken MPU configuration	☺	☹	☹	☹	☺

Future Work

- Continue exploitation
  - Identify different regions with different privileges in MPU_based RTOS firmware
    - Identify the trampoline functions in MPU_based RTOS firmware
    - Gadgets in kernel space are not accessed by user space
- Protection
  - Finer granularity isolation if performance allows
    - MPU Virtualization

Thank You !

Twitter: @Y1nKoc

Email: yenkoclike@gmail.com

Personal Page: https://minghaolin2000.github.io/