Return-to-Non-Secure Vulnerabilities on ARM Cortex-M TrustZone: Attack and Defense

Zheyuan Ma†‡, Xi Tan†‡, Lukasz Ziarek†, Ning Zhang*, Hongxin Hu†, Ziming Zhao†‡
†University at Buffalo ‡CactiLab *Washington University in St. Louis
{zheyuanm, xitan, lziarek, hongxinh, zimingzh}@buffalo.edu, zhang.ning@wustl.edu
Background

- TrustZone on Cortex-A vs Cortex-M
  - Cortex-M's rapid state switch has security implications
  - The semantic gap resulting in potential confused-deputy attacks
Background

In Secure State, IPSR != 0
Handler mode

In Non-secure State, IPSR = 0
Thread mode

Privileged Level

Unprivileged Level

CONTROL.nPRIV = 0
Privileged Level

CONTROL.nPRIV = 1
Unprivileged Level

IPSR register is shared between states; CONTROL register is banked for each state.
**Background**

- **bxns/blxns**
- **Non-secure State**
- **Secure State**
- **Privileged**
  - Function call/return
  - **Handler mode**
  - **Thread mode**
  - Privileged (CONTROL_NS.nPRIV = 0)
  - Unprivileged (CONTROL_NS.nPRIV = 1)
- **Privileged**
  - Function call/return
  - Privileged (CONTROL_S.nPRIV = 0)
  - Unprivileged (CONTROL_S.nPRIV = 1)

IPSR register is shared between states; CONTROL register is banked for each state.
Related Work

- Ret2user, ret2dir, and boomerang attacks
  - Targeting microprocessors with MMUs and modern OSes like Linux
- Secure communication mechanisms on Cortex-M TrustZone
  - Our attack does not rely on tempering cross-state messages

The ret2ns Attacks – Threat Model

Goal: user-space attacker in NS conducts privilege escalation

Assumptions:
- memory corruption vulnerability in S
- attacker utilizes NS system calls (SVC) for S interaction
- arbitrary code execution in S not possible for attacker

Target: to corrupt code pointer used by bxns/blxns in S
The ret2ns Attacks – Overview

Handler-mode-originated attacks
- IPSR is shared between states

Thread-mode-originated attacks
- CONTROL.nPRIV is banked between states
The ret2ns Attacks – A Walking Example

```c
#define MAX_LEN 128
int32_t _driver_LCD_ready();
int32_t _driver_LCD_print(char *);

/*@ Non-secure callable function */
int32_t print_LCD_nsc(char *msg)
__attribute__((cmse_nonsecure_entry));

int32_t print_LCD_nsc(char *msg)
{
    char buf[MAX_LEN] = {0};
    if (_driver_LCD_ready())
    {
        printf(buf, "%s %s: %s", _TIME_STAMP,
            _SYSTEM_STATUS, msg); /* Buffer overflow */
        return _driver_LCD_print(buf); /* bxns return */
    }
    else
    {
        return -1; /* bxns return */
    }
}
```

User Input: print_LCD

1. SVC call

SVC_Handler

IPSR=0

Non-secure state

Secure State

IPSR=11

Non-secure state
The ret2ns Attacks – A Walking Example
Defense 1

MPU-assisted Address Sanitizer

- Validate memory access permissions for NS target
- Verify NS destination address against NS MPU configuration before bxns/blxns branches

```
1  ldr  R0, [SP, #12] ; only need for bxns
2  mrs  R3, IPSR ; read IPSR value
3  cbnz R3, #6 ; check if IPSR is zero
4  mrs  R3, CONTROL_NS ; read CONTROL_NS
5   lsls R3, R3, #31 ; get CONTROL_NS.nPRIV bit
6    bne #28 ; check if nPRIV bit zero
7     tta R0, R0 ; test target on target addr
8    lsls R3, R0, #15 ; get MRVALID bit
9       bpl #20 ; check if region is valid
10      uxtb R0, R0 ; get MPU region number
11  movw R3, #0xED98 ; load MPU_NS->RNR addr
12    movt R3, #0xE002 ; 0xE002ED98
13     str R0, [R3, #0] ; set MPU_NS->RNR to region
14    ldr R0, [R3, #4] ; get MPU_NS->RBAR
15   lsls R0, R0, #30 ; get UP read permission bit
16    beq #2 ; check UP read permission
17    cpsid i ; disable IRQ
18    b . ; error handling
```
Defense 2

Address Masking

- Assume user/kernel space programs in distinct, known memory regions
- Apply bit-wise masking to NS target address

```asm
1 ldr  R1, [SP, #4] ; get return addr (not for blxns)
2 mrs  R2, IPSR ; read IPSR
3 cbnz R2, #6 ; check if IPSR is zero
4 mrs  R2, control_ns ; read CONTROL_NS
5 lsls R2, R2, #31 ; get CONTROL_NS.nPRIV bit
6 bne  #8 ; check nPRIV bit
7 cmn  R1, #0x100 ; Is EXC_RETURN? (not for blxns)
8 it   cc ; not EXC_RETURN (not for blxns)
9 movtcc R1, #0x21 ; address masking
10 strcc R1, [SP, #4] ; store result (not for blxns)
```
Defense Evaluation

A modified Blinky application with S and NS states

- Higher $T$: less frequent routine communications between states
- Higher $N$: less frequent service requests from NS to S

<table>
<thead>
<tr>
<th>$T, N$</th>
<th>Blinky</th>
<th>MPU-assisted Addr Sanitizer</th>
<th>Address Masking</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^7, 10^7$</td>
<td>1,200,503,441</td>
<td>1,200,508,359 (0.0004%)</td>
<td>1,200,506,190 (0.0002%)</td>
</tr>
<tr>
<td>$10^5, 10^5$</td>
<td>12,503,869</td>
<td>12,508,385 (0.0361%)</td>
<td>12,506,793 (0.0234%)</td>
</tr>
<tr>
<td>$10^7, 10^5$</td>
<td>12,473,897</td>
<td>12,474,465 (0.0046%)</td>
<td>12,474,243 (0.0028%)</td>
</tr>
<tr>
<td>$10^5, 10^7$</td>
<td>1,203,433,289</td>
<td>1,203,892,073 (0.0381%)</td>
<td>1,203,674,733 (0.0201%)</td>
</tr>
</tbody>
</table>

The Blinky application is a cross-world project with both non-secure and secure state programs, and it works on a system with 3 LEDs and a UART peripheral. $T$ controls the triggering frequency of the SysTick timer, and the $N$ defines the number of nop instructions before each NSC function call.
Q&A

Open-sourced at: https://github.com/CactiLab/ret2ns-Cortex-M-TrustZone

Zheyuan Ma • CactiLab

zheyuanm@buffalo.edu