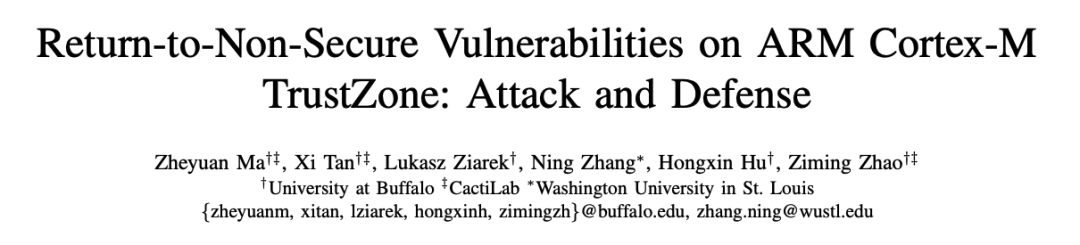
G.O.S.S.I.P 阅读推荐 2023-06-27

2023-06-27 19:38

今天要为大家推荐的论文是来自DAC 2023的Return-to-Non-Secure Vulnerabilities on ARM Cortex-M TrustZone: Attack and Defense，由纽约州立大学布法罗分校的 CactiLab与华盛顿大学圣路易斯分校合作完成并投稿。作者展示了如何利用 Cortex-M TrustZone 的快速状态切换机制（fast state switch mechanism）在非安全状态下实施提权与任意代码执行，并提出了一个新的攻击方式—— ret2ns，同时作者还在两个 Cortex-M 硬件系统上验证了四种 ret2ns 的攻击变种的可行性，并设计了两种具有微小性能开销的验证机制以防御 ret2ns 攻击。

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



背景介绍

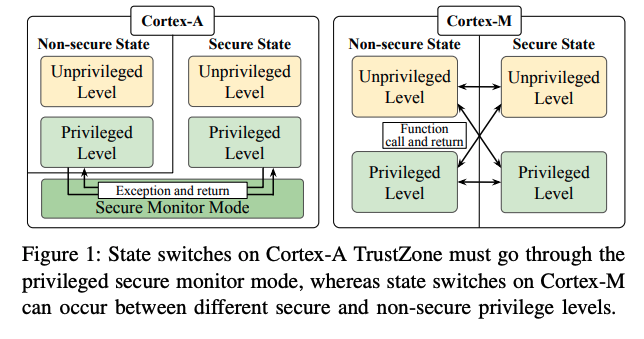
# 背景介绍

TrustZone 是 ARM 引入的硬件隔离机制，它将处理器隔离成非安全状态和安全状态。安全状态可以访问两个状态的所有资源包括内存、外设和总线，非安全状态只能访问属于自己状态的资源。

Cortex-M 的处理器和 Cortex-A 的处理器在 TrustZone 的实现上有很大的差异。如图1所示，Cortex-A 除了非安全状态和安全状态以外，额外添加了 monitor 模式，非安全状态只能调用 SMC 指令通过 monitor 来进行状态切换，即只有一个入口。

而 Cortex-M 的非安全状态和安全状态的区分于内存所在地址，分为安全内存，非安全内存和非安全可调用内存。非安全状态通过SG指令进入到非安全可调用区域从而切换到安全状态，安全状态通过 bxns 和 blxns 返回或者调用非安全状态的函数。

这个切换机制被称为快速状态切换，且 Cortex-M 上的状态切换入口数量不受限制。



在 TrustZone 设计中，不同状态存在语义鸿沟，即安全状态无法验证非安全状态通过安全状态再返回非安全状态的访问权限，从而导致 “Confused Deputy Attack”。

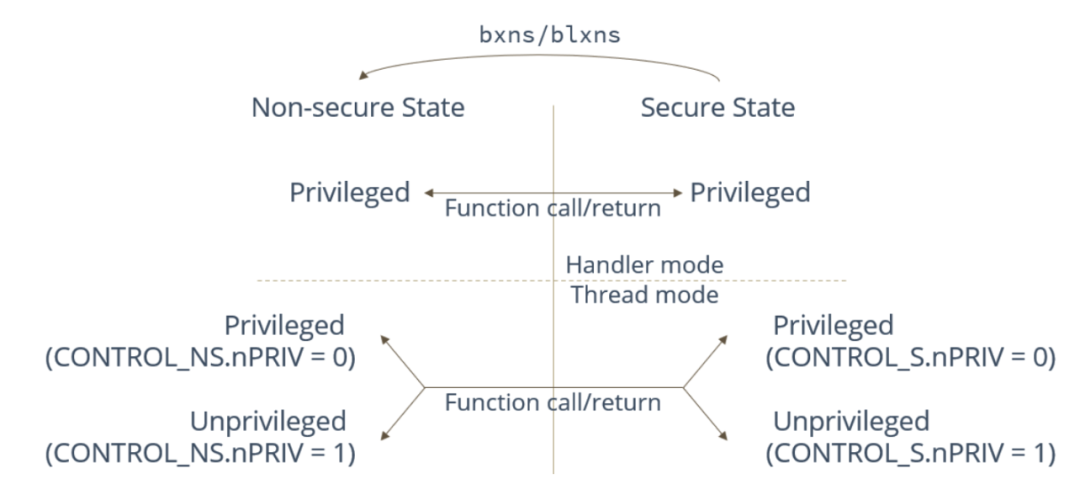
本文作者提出 ret2ns 攻击，利用前文提到的 Cortex-M 的快速状态切换机制，可以完成非安全状态非特权代码的提权和任意代码执行，能影响所有启用 TrustZone 的 Cortex-M 处理器。

ret2ns 攻击不同于以往的ret2usr [1] 和 boomerang [2]，因为 ret2usr 并不针对 TrustZone 的状态切换，而 boomerang 不会直接导致任意代码的执行。

Ret2ns 攻击

# Ret2ns 攻击

启用 TrustZone 扩展的 Cortex-M 在非安全状态和安全状态下都有两种执行模式—— handler 和 thread 模式。handler 模式始终处于特权等级，运行系统异常处理代码，包括 supervisor call（SVC）。thread 模式可以在高特权等级或者低特权等级执行，通常高特权等级用于内核空间，低特权等级用于用户空间。



在 Cortex-M 中，IPSR 寄存器中的非零值表示 handler 模式，CONTROL 寄存器的 nPRIV 域的值决定 thread 模式下的特权等级，0为非特权等级，1为特权等级。

由于 IPSR 寄存器共享于两个状态，所以在 handler 模式下发生的状态切换会一直停留在 handler 模式。而 CONTROL 寄存器在不同状态下有独立存储，因此 thread 模式下状态切换后的特权等级取决于各自状态中的 CONTROL 寄存器。

# 会一直停留在 handler 模式

# 取决于各自状态中的 CONTROL 寄存器

而以上两个寄存器的机制促使了 ret2ns 攻击的发生。

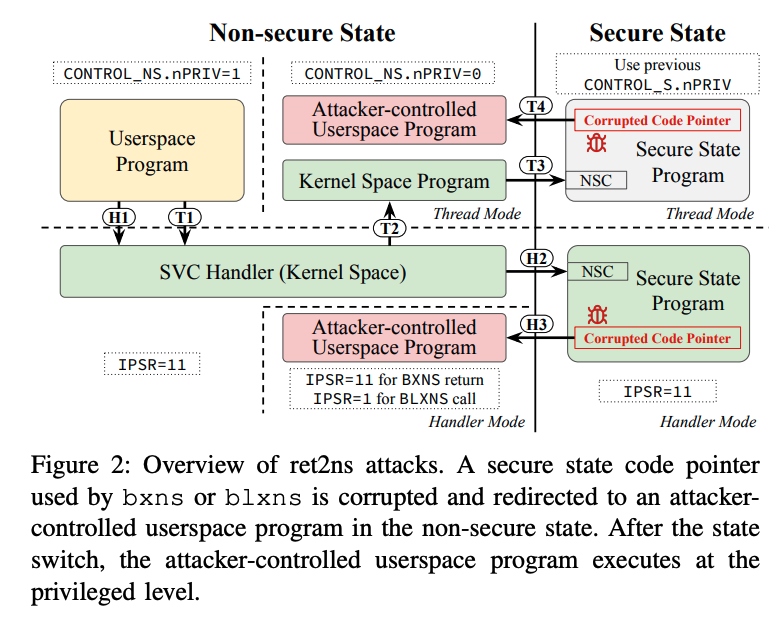


图2解释了 ret2ns 的攻击方式，基于攻击出发点所在非安全状态的执行模式，作者将 ret2ns 攻击分为两类：基于 handler 模式的攻击和基于 thread 模式的攻击。需要注意的是，在 TrustZone 安全扩展的设计中，当程序执行从安全状态到非安全状态的函数返回或函数调用时，编译器将采用专门的 bxns 或 blxns 指令，而在各自安全状态内，会使用 bx 或 blx 指令。

基于 handler 模式的攻击（H1→H2→H3）

# H1→H2→H3

非安全状态的非特权代码调用 SVC 指令进入 handler 模式后（H1），通过非安全可调用函数（NSC）进入安全状态（H2）。在此过程中，IPSR 寄存器将会保持同样的非零值。如果安全状态的代码有漏洞，在返回非安全状态时，并没有返回到对应的 handler 函数，而是会返回到攻击者控制的内存区域（H3）。此时 IPSR 依然是之前的非零值，代表 CPU 仍然运行在特权等级，将会导致攻击者的代码提权。在论文中有针对此攻击的详细示例。

# H1

# H2

# H3

基于 thread 模式的攻击（T1→T2→T3→T4）

# T1→T2→T3→T4

相反，在基于 thread 模式的攻击中，由于 CONTROL 寄存器在两个状态下分别拥有独立备份，如果程序之前在非安全状态下的内核空间执行（T1&T2），并切换到安全状态（T3），在返回到非安全状态时（T4），CPU 将会沿用该状态下以前保存的 CONTROL 寄存器。这意味着即使返回目标是非安全状态的非特权代码区域，程序都会使用内核空间的特权等级，从而达到提权的目的。

# T1&T2

# T3

# T4

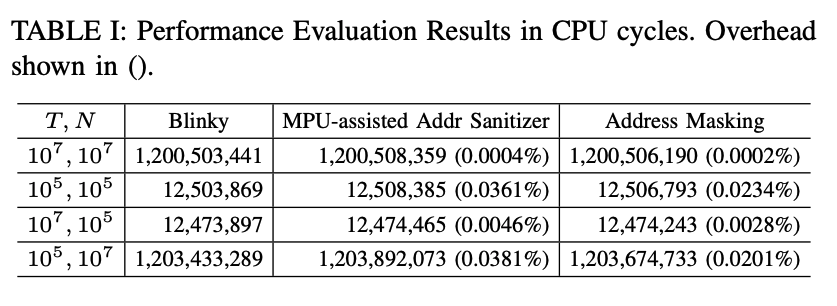
Ret2ns 的防护

# Ret2ns 的防护

作者提出了两种防御机制来抵御 ret2ns 攻击，防御的关键点在于限制在非安全模式下的 用户空间程序在高特权等级的执行。

第一个方法是加入由内存保护单元（MPU）辅助的地址检查：每当从安全状态跳转到非安全状态之前，总是验证在非安全状态下的目标地址的内存访问权限。具体来说，首先假设在非安全状态下已经正确设置了 MPU，然后通过检查 MPU 的配置来交叉验证跳转的目标地址，以确定在跨状态跳转之前该地址的合法性。

第二个方法是通过地址掩码（Address Masking）来确保目标地址必定落入预设的范围。在这个方法中，需要开发者提前将用户和内核空间的程序分配到独立的、定义明确的内存区域。防御的方法是在切换到非安全状态时，对每个目标地址都进行一次应用掩码操作，这就可以确保它永远不会落入一个不适当的地址范围。虽然这种方法是粗粒度的，但如果配置正确的话可以同时兼顾速度与防御。



作者对两种防御机制都进行了性能评估。T 值控制了调用 blxns 进行状态切换的频率，N 值控制了调用 bxns 进行状态切换的频率。结果显示两种方法的性能开销都是非常小的。其主要原因是这些防御机制的实现都仅引入了固定数量的额外指令，同时每次状态切换时的开销只有大约20个 CPU 周期。

论文下载：https://cactilab.github.io/assets/pdf/ret2ns2023.pdf

代码开源：https://github.com/CactiLab/ret2ns-Cortex-M-TrustZone

投稿团队介绍：

Ret2ns 作者团队共6名研究人员，分别来自纽约州立大学布法罗分校的 CactiLab（https://cactilab.github.io/）和圣路易斯华盛顿大学。其中第一作者 Zheyuan Ma（麻哲源） 和第二作者 Xi Tan（谭曦）都是纽约州立大学布法罗分校计算机系 Ziming Zhao（赵子铭）教授的博士生。其它作者包括 Lukasz Ziarek 教授、Hongxin Hu（胡宏新）教授和圣路易斯华盛顿大学计算机系的 Ning Zhang（张宁）教授。

[2] Machiry, Aravind, et al. "BOOMERANG: Exploiting the Semantic Gap in Trusted Execution Environments." NDSS, 2017.