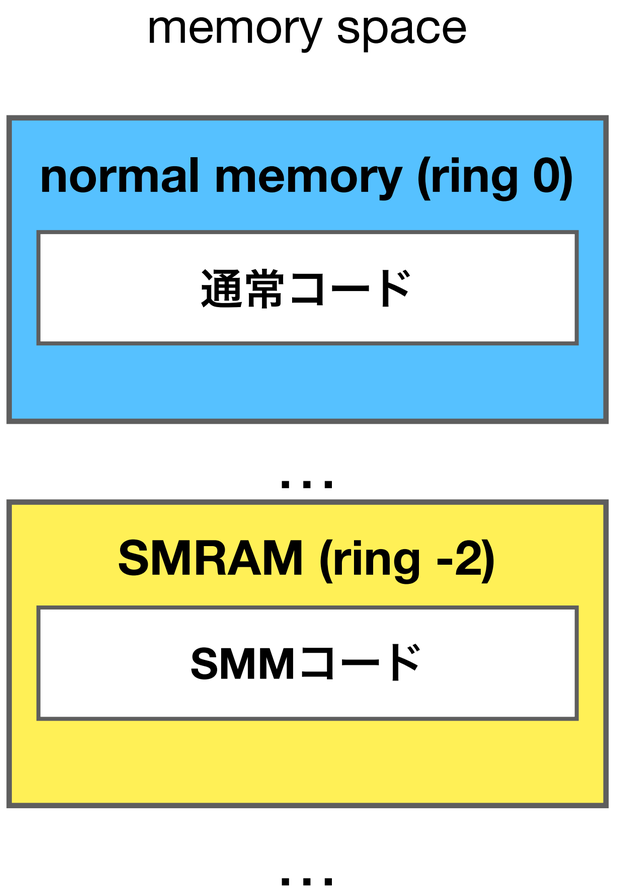
SMM Calloutの解説と静的解析で検出する方法

2026/06/26, @retrage, Ring Minus Security

前提知識

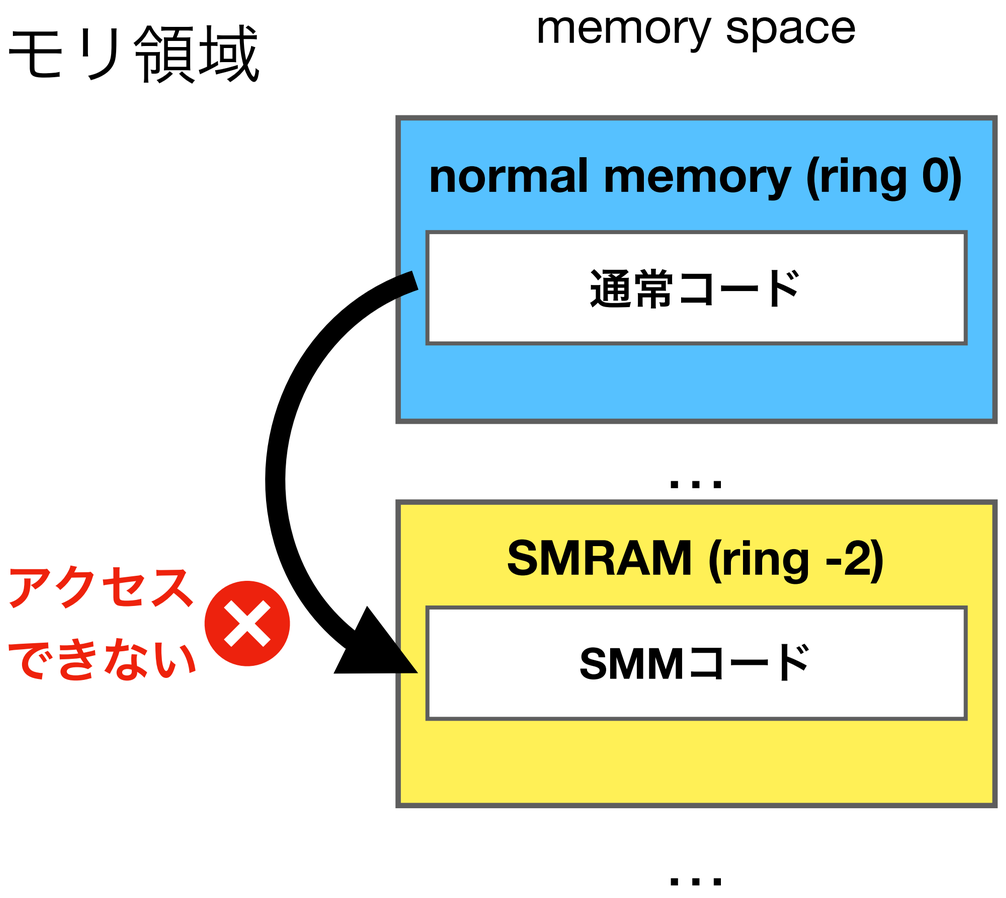
ring 0からはring -2のSMRAMにはアクセスできない

- すべてのSMMコードはSMRAMというメモリ領域で完結



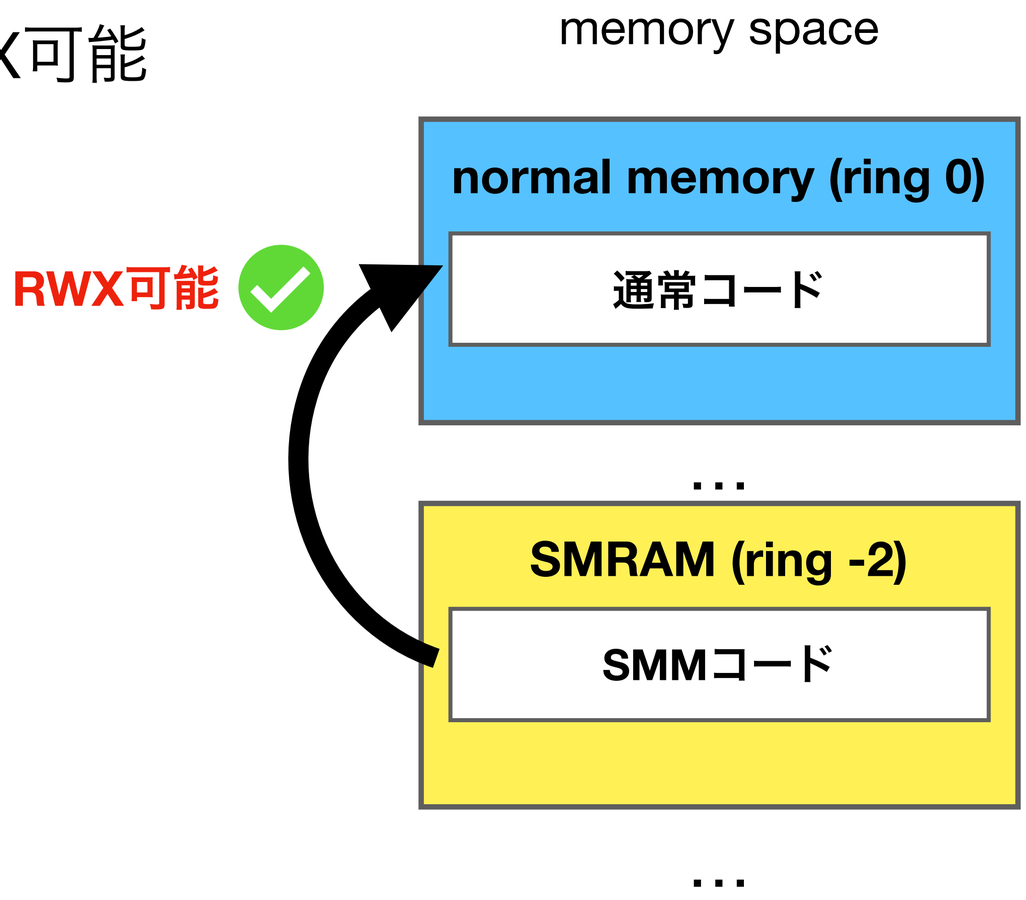
ring 0からはring -2のSMRAMにはアクセスできない

- すべてのSMMコードはSMRAMというメモリ領域で完結
- 外部からSMRAMにアクセスできない
- 当たり前



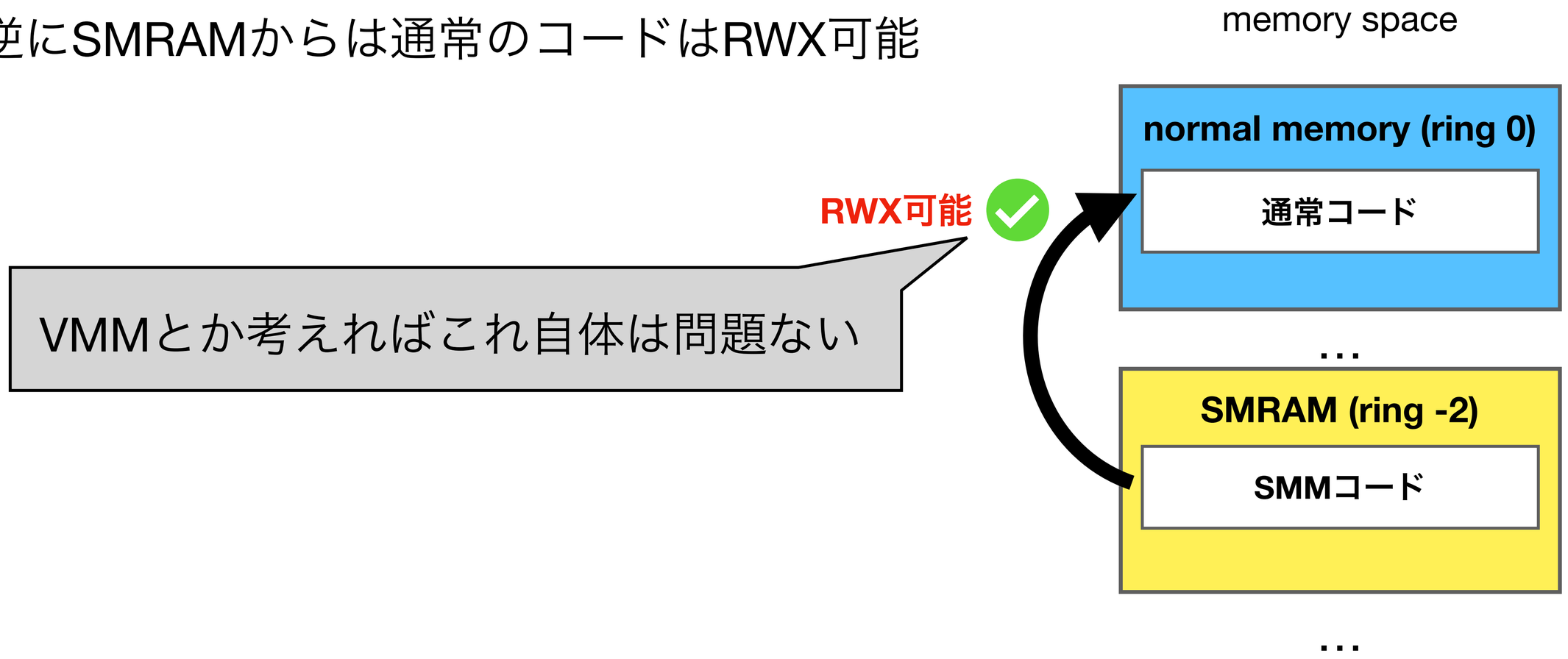
逆にSMMからnormal memoryにアクセス可能

- 逆にSMRAMからは通常のコードはRWX可能



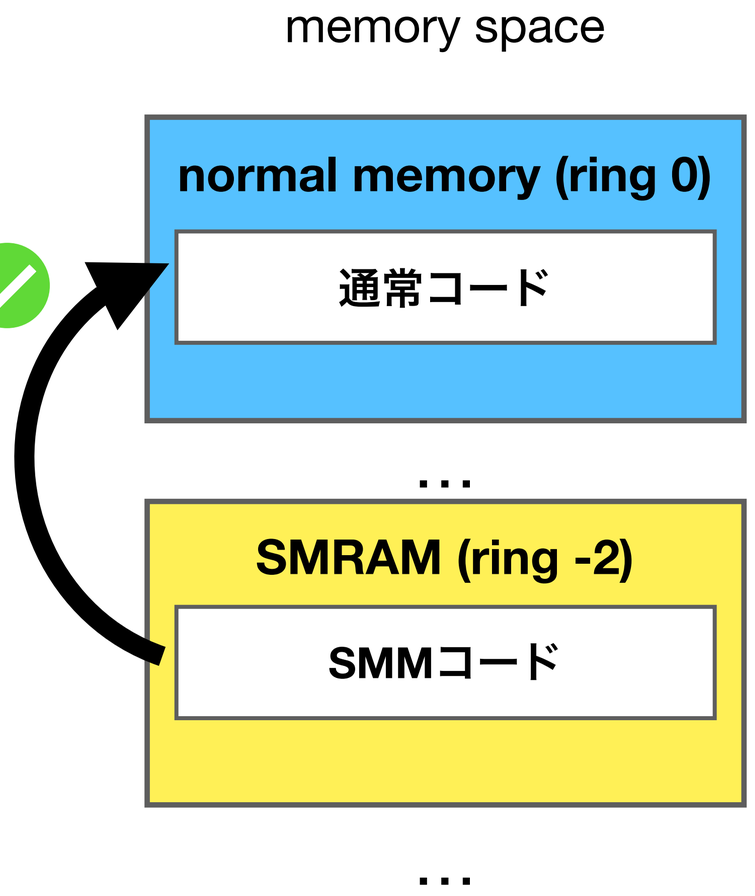
逆にSMMからnormal memoryにアクセス可能

- 逆にSMRAMからは通常のコードはRWX可能



SMMの設計の欠陥

- SMMから見てメモリ空間が分離されていない
- SMMでMMU無効の場合は4GiBまでのすべてのメモリにアクセス可能
- **-> 通常コードにSMMから意図せずアクセスできてしまう**



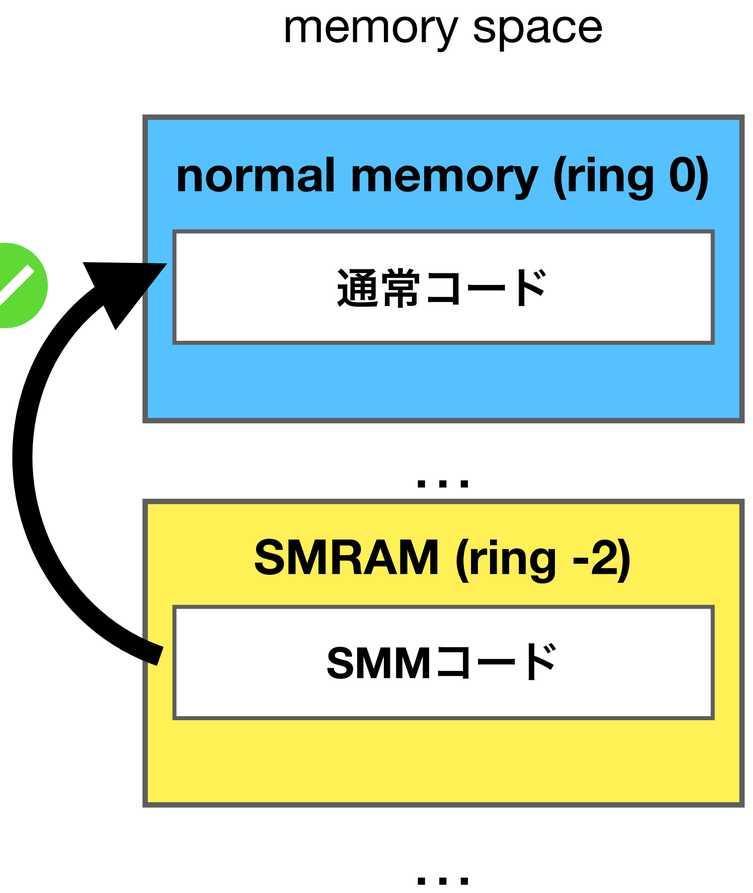
SMMの設計の欠陥

- SMMから見てメモリ空間が分離されていない
- SMMでMMU無効の場合は4GiBまでのすべてのメモリにアクセス可能

RWX可能

- **-> 通常コードにSMMから意図せずアクセスできてしまう**

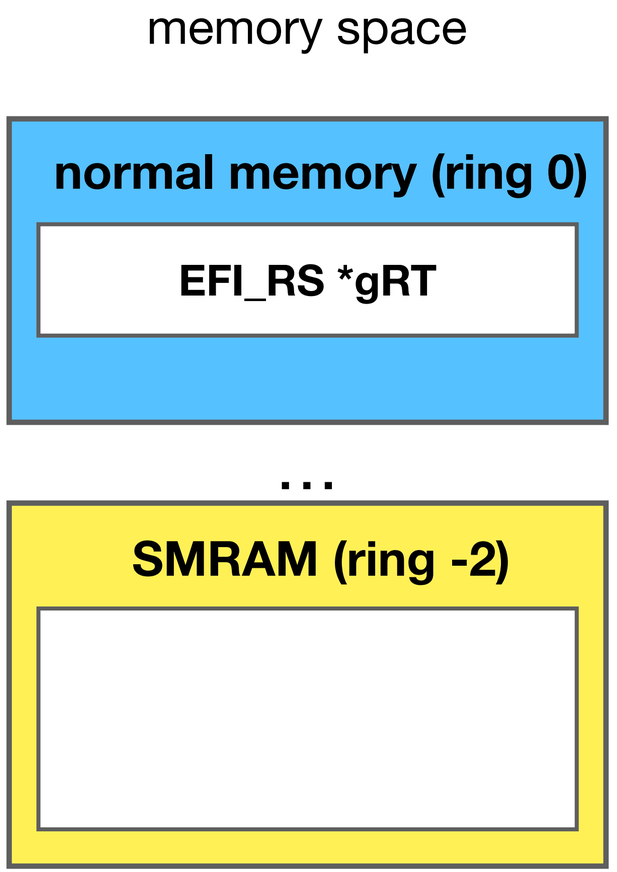
明示的にマップしないとアクセスできないなどで本来あるべき



SMM Callout:
SMMの設計に起因する脆弱性

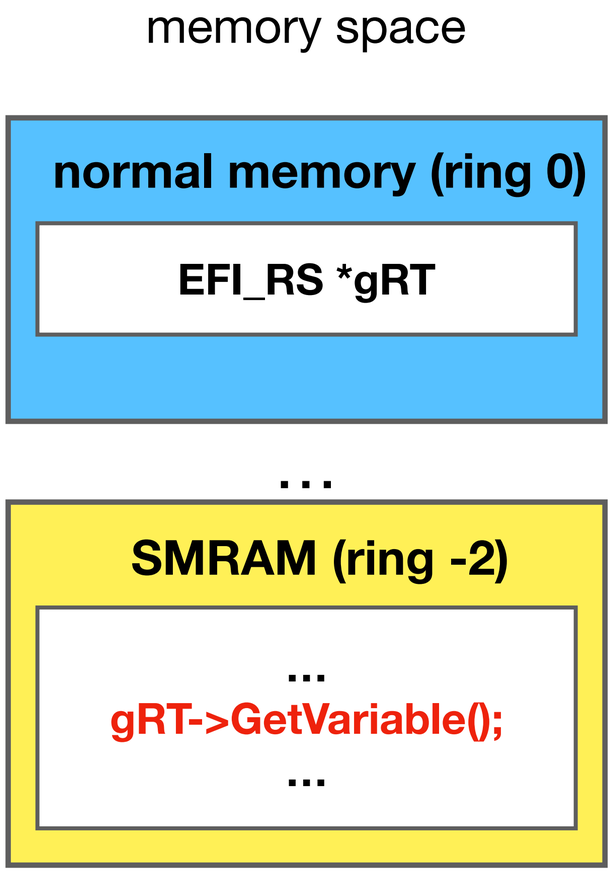
SMM Calloutの基本原理

- normal memoryにgRTがあるとして



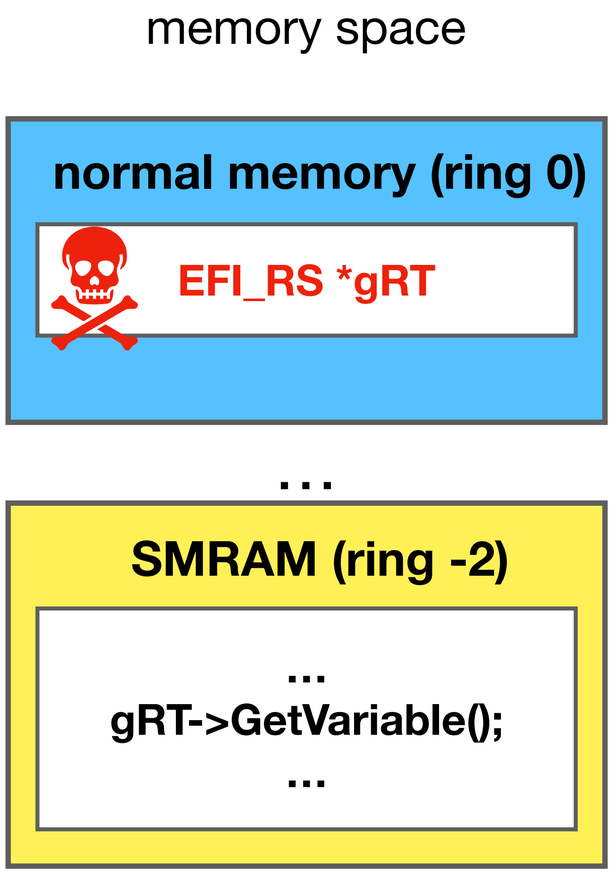
SMM Calloutの基本原理

- normal memoryにgRTがあるとして
- SMRAM内のコードでgRTを使ってしまった場合
  普通にSMMからgRTが使えてしまう



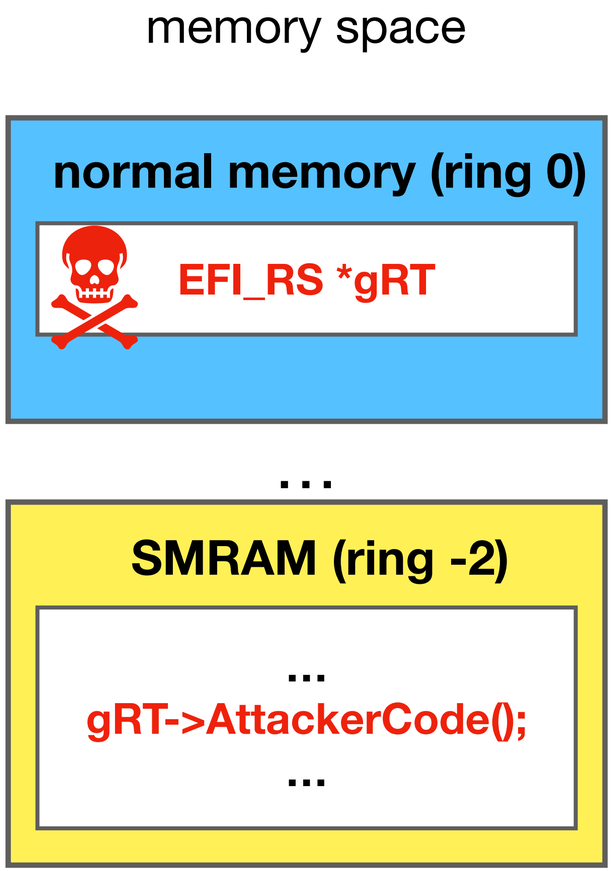
SMM Calloutの基本原理

- normal memoryにgRTがあるとして
- SMRAM内のコードでgRTを使ってしまった場合
  普通にSMMからgRTが使えてしまう
- Ring 0のメモリに書き込み可能な攻撃者がgRTを
  書き換えた場合



SMM Calloutの基本原理

- normal memoryにgRTがあるとして
- SMRAM内のコードでgRTを使ってしまった場合
  普通にSMMからgRTが使えてしまう
- Ring 0のメモリに書き込み可能な攻撃者がgRTを
  書き換えた場合
- SMMで任意コード実行できてしまう



現在でもSMM Calloutは見つかる

- こんなに簡単な脆弱性なのに未だに報告される
- CVE-2025-54502: AMDのSMM driver

Summary

Incorrect use of LocateProtocol Service by a privileged attacker with local access could potentially result in privilege escalation from Ring 0 to System Management Mode (SMM) potentially resulting in Arbitrary Code Execution.

CVE Details

Refer to Glossary for explanation of terms

CVE	CVE Description	CVSS Score
CVE-2025-54502	Incorrect use of boot service in the AMD Platform Configuration Blob (APCB)	7.1 (High) CVSS:4.0/AV:L/AC:H/AT:N/PR:H/UI:N/VC:H/VI:H/VA:H/SC:N/SI:N/SA:N

SMM Calloutの検出

静的解析での典型的なSMM Calloutの検出方法

- 目的: **gBS/gRTをSMRAMで参照する典型例**を検出
- IDA/Ghidraなどバイナリからの型復元が可能な静的解析ツールがあれば芋づる式に以下が可能
  - SMIハンドラ(SMM割り込みハンドラ)の関数の特定
  - 関数内でgBS/gRTが使われているかの判定
- この二つの条件を満たしていれば典型的なSMM Calloutとして検出可能

GhidraでのSMM Callout検出

- efiXplorer[BHEU '20]のSMM Callout検出機能をGhidraに再実装した
  - IDA向けのプラグイン https://github.com/REhints/efiXplorer

GhidraでのSMM Callout検出

- efiXplorer[BHEU '20]のSMM Callout検出機能をGhidraに再実装した
  - IDA向けのプラグイン https://github.com/REhints/efiXplorer
- efiSeek (https://github.com/DSecurity/efiSeek)をベースに実装
  - Ghidra上でUEFIバイナリにGUID/プロトコル情報を与えてくれる

GhidraでのSMM Callout検出

- efiXplorer[BHEU '20]のSMM Callout検出機能をGhidraに再実装した
  - IDA向けのプラグイン https://github.com/REhints/efiXplorer
- efiSeek (https://github.com/DSecurity/efiSeek)をベースに実装
  - Ghidra上でUEFIバイナリにGUID/プロトコル情報を与えてくれる
  - SMIハンドラも見つけてくれる <- とてもうれしい

GhidraでのSMM Callout検出

- efiXplorer[BHEU '20]のSMM Callout検出機能をGhidraに再実装した
  - IDA向けのプラグイン https://github.com/REhints/efiXplorer
- efiSeek (https://github.com/DSecurity/efiSeek)をベースに実装
  - Ghidra上でUEFIバイナリにGUID/プロトコル情報を与えてくれる
  - SMIハンドラも見つけてくれる <- とてもうれしい
  - upstreamの更新が2022年4月で止まっている <- とてもかなしい

GhidraでのSMM Callout検出 (1)

- gBS/gRTなどのグローバルポインタのアドレスを記録

```
while (test.hasNext()) {
    Varnode var = test.next().getOutput();
    if (var != null && var.isAddress()) {
        String name = null;
        switch (var.getHigh().getDataType().getName()) {
        case "EFI_SYSTEM_TABLE *":
            name = "gST_" + this.nameCount++;
            gSTAddresses.add(var.getAddress());
            break;
        case "EFI_BOOT_SERVICES *":
            name = "gBS_" + this.nameCount++;
            gBSAddresses.add(var.getAddress());
            break;
        case "EFI_RUNTIME_SERVICES *":
            name = "gRS_" + this.nameCount++;
```

GhidraでのSMM Callout検出 (2)

- GhidraのpCode (中間表現) でSMIハンドラを登録しているコードを特定

```
switch (pCode.getInput(0).getHigh().getDataType().getName()) {
case ("EFI_SMM_POWER_BUTTON_REGISTER2"):
    funcName = "pwrButtonHandler";
    fdefName = "EFI_SMM_POWER_BUTTON_REGISTER2";
    break;
case ("EFI_SMM_SX_REGISTER2"):
    funcName = "sxHandler";
    fdefName = "EFI_SMM_SX_REGISTER2";
    break;
case ("EFI_SMM_SW_REGISTER2"):
    funcName = "swSmiHandler";
    root = this.swSmi;
    fdefName = "EFI_SMM_SW_REGISTER2";
    isSwSmiHandler = true;
```

GhidraでのSMM Callout検出 (3)

- 特定したSMIハンドラについて、gRT/gBSを参照しているかを判定

```
for (PcodeOp pCode : pCodeOps) {
    if (pCode.getOpcode() == PcodeOp.COPY) {
        Varnode input0 = pCode.getInput(0);
        Varnode output = pCode.getOutput();
        if (input0.isAddress()
        && (funcParamForwarding.getgBSAddresses().contains(input0.getAddress())
            || funcParamForwarding.getgRSAddresses().contains(input0.getAddress()))
        && output.isRegister()) {
            Msg.warn(this, "Potential SMM callout detected at "
            + func.getName() + "(" + func.getEntryPoint().toString() + ")"
            + " : " + inst.getAddress().toString());
            calloutAddresses.add(inst.getAddress());
        }
```

まとめ

- SMM CalloutはSMMがnormal memoryにあるgBS/gRTを使ってしまう脆弱性
  - normal memoryに特別な操作なしに触れてしまう設計上の欠陥に起因する
- SMM Calloutは静的解析で検出可能
  - SMIハンドラ内でのgBS/gRTの参照を判定すればよい
- GhidraでもefiXplorerと同様のSMM Calloutの検出は可能
  - efiSeekのupstreamが止まっているのをforkメンテナンスしたい！
    - https://github.com/retrage/efiSeek/tree/efi-xplorer