SecHack365

zkprobeの開発

学習駆動コース　坂井ゼミ　吉田侑生

1. zkprobeの概要と作成経緯

概要:zkprobe=Zephyr+kprobe

- OSSのRTOS(Real Time OS)であるZephyrにkprobeを実装する
- 利点:Zephyrでkernel空間に処理を加えて実行する際に、再ビルドする必要がない
- ユーザがkernel空間に対して好きな処理を入れて試すことができる!

作成経緯:

- 勉強会でeBPFを知り、興味を持った
- 興味を持ったeBPFを実装して学ぼうと思い、まずはeBPFで使われている kprobeを実装して学びたいと考えた
- OSSで活動が活発なZephyrを対象にした

単語の説明:

eBPF:

- Linux kernelの安全な仮想マシン上で任意のイベントに対して処理を実行できる
- kernel空間内の操作をフックしてトレースできる

Zephyr:

- OSSのRTOS(Real Time OS)の一種
- たくさんのボードをサポートしている(180種類以上)

kprobe:

- kernel空間で関数やアドレスをフックし、フックした前後に自分で定義した処理を試す仕組み
- kernelのデバッグやパフォーマンス解析に利用される

2.SecHack365での取り組み

作品に関して試したこと:

- kprobeのソースコードの調査
- Zephyrのソースコードの調査(メモリ周り、IRQの処理、fatalの処理など)
- Zephyrでのint3命令のhandlerの設定
- zkprobeの関数の一部実装

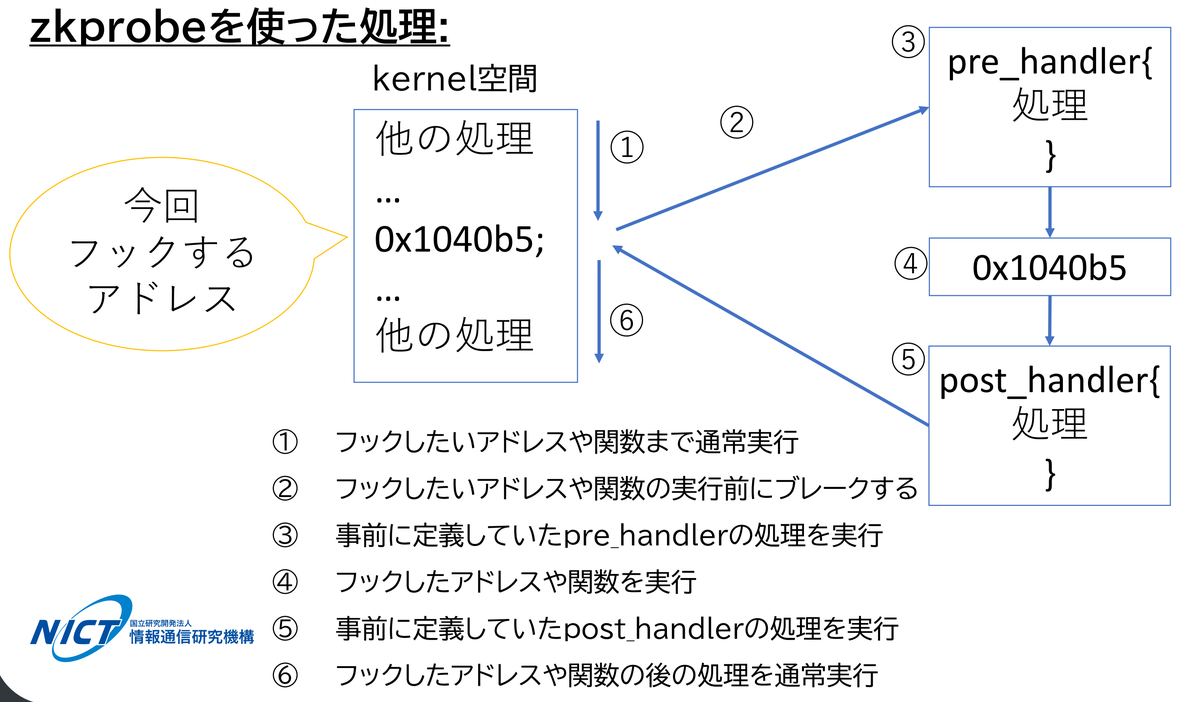
```
-- west build: running target run
[3/13] Linking C executable zephyr/zephyr_prebuilt.elf

[9/13] Linking C executable zephyr/zephyr.elf
Memory region         Used Size  Region Size  %age Used
           RAM:         112 KB         3 MB      3.65%
       LOCORE:          32 KB        60 KB     53.33%
[12/13] To exit from QEMU enter: 'CTRL+a, x'[QEMU] CPU: qemu64,+x2apic
qemu-system-x86_64: warning: TCG doesn't support requested feature: CPUID.01H:ECX.x2a
pic [bit 21]
qemu-system-x86_64: warning: TCG doesn't support requested feature: CPUID.01H:ECX.x2a
pic [bit 21]
SeaBIOS (version zephyr-v1.0.0-0-g31d4e0e-dirty-20200714_234759-fv-az50-zephyr)
Booting from ROM..*** Booting Zephyr OS build zephyr-v2.6.0-825-gdb9756045e1a  ***
hello
----------------------
break success
----------------------
finish
```

int3実行時の画面

3.zkprobeでできること

- 通常の処理とは異なり、Zephyrのkernel空間で自分のフックしたいアドレスや関数の前後に、好きな処理を差し込み実行できる!
- pre_handlerやpost_handlerで自分が実行したい処理を事前に定義できる

zkprobeを使った処理:



① フックしたいアドレスや関数まで通常実行
② フックしたいアドレスや関数の実行前にブレークする
③ 事前に定義していたpre_handlerの処理を実行
④ フックしたアドレスや関数を実行
⑤ 事前に定義していたpost_handlerの処理を実行
⑥ フックしたアドレスや関数の後の処理を通常実行

NICT 国立研究開発法人 情報通信研究機構

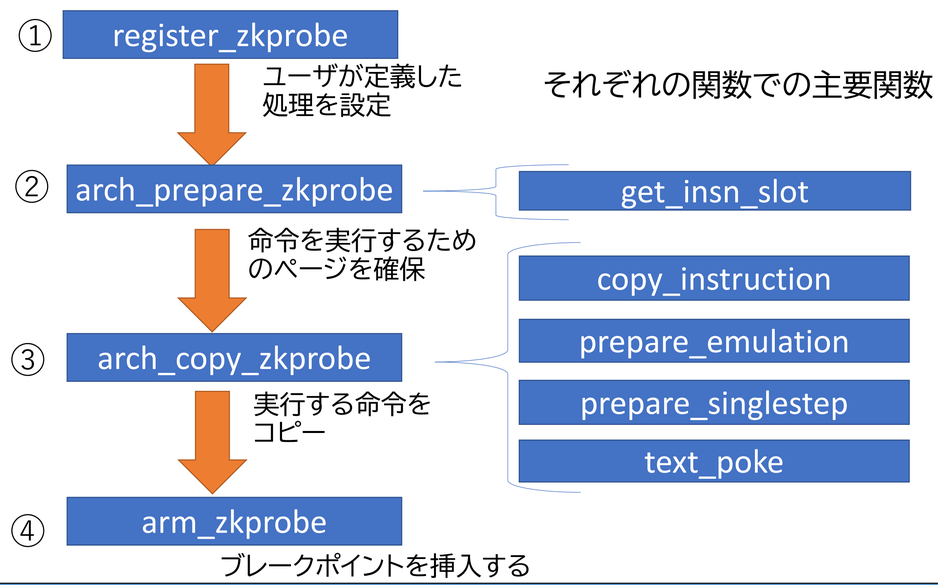
4. zkprobeの特徴と仕組み

特徴:

- モジュールではなく、システムコールでの実装
- Linuxのkprobeとは実装方法が異なる

(Linuxでのstruct pageの代わりに、Zephyrでは、struct page_frameを利用するなど)

仕組み:



それぞれの関数での主要関数

5. セキュリティ意義

レジスタ情報取得:

zkprobeを用いて、レジスタを表示する → 攻撃の解析に繋がる

対象者:
セキュリティアナリスト、
Zephyrのユーザ

右図はイメージ画像

```
[ 5156.316108] pre kprobe:p->addr = 0x0000000039822845, ip = ffffffff9491b041
                    ax=0xffffffff9491b120,bx=0x0
                    cx=0x0,dx=0x88802
                    sp=0xffffb5400062bf18
[ 5156.316110] post kprobe:p->addr = 0x0000000039822845, ip = ffffffff9491b045
                    ax=0xffffffff9491b120,bx=0x0
                    cx=0x0,dx=0x88802
                    sp=0xffffb5400062bf18
```

レジスタ情報取得のイメージ

可視化:

zkprobeを用いて関数の実行回数を可視化 → 通信やプロセスの異常検知に繋がる

対象者:セキュリティアナリスト

右図はイメージ画像

```
[ 1112.326949] pre_handler
[ 1112.326950] <do_sys_open> post_handler
[ 1112.326950] success
[ 1112.326951] 46894
[ 1112.326977] pre_handler
[ 1112.326977] <do_sys_open> post_handler
[ 1112.326978] success
[ 1112.326978] 46895
[ 1112.327005] pre_handler
[ 1112.327005] <do_sys_open> post_handler
[ 1112.327006] success
[ 1112.327006] 46896
```

可視化のイメージ

6. 今後の予定

- zkprobeの開発継続
- 開発の詳しい状況などはQRコード、下記URLにて公開予定
  https://y0sh1da.hatenablog.com/

7.感想

- 今回初めて、OSSに関する開発をした
- 想像よりもソースコードの調査や実験に時間がかかり、低レイヤー開発の難しさ・大変さ(具体例:実装に必要な知識の多さ、ソースコード1行の影響の大きさ、動くものを作るまでの工程の多さなど)を知った
- 低レイヤーでの作成は大変なこともあるが、自分の思った通りに動いた時の嬉しさについても改めて気づけた
- これからもzkprobeの開発を続けていきたい