バイナリを書き換えて システムコールをフックする

Kernel/VM探検隊online part4
2021年 11月 20日

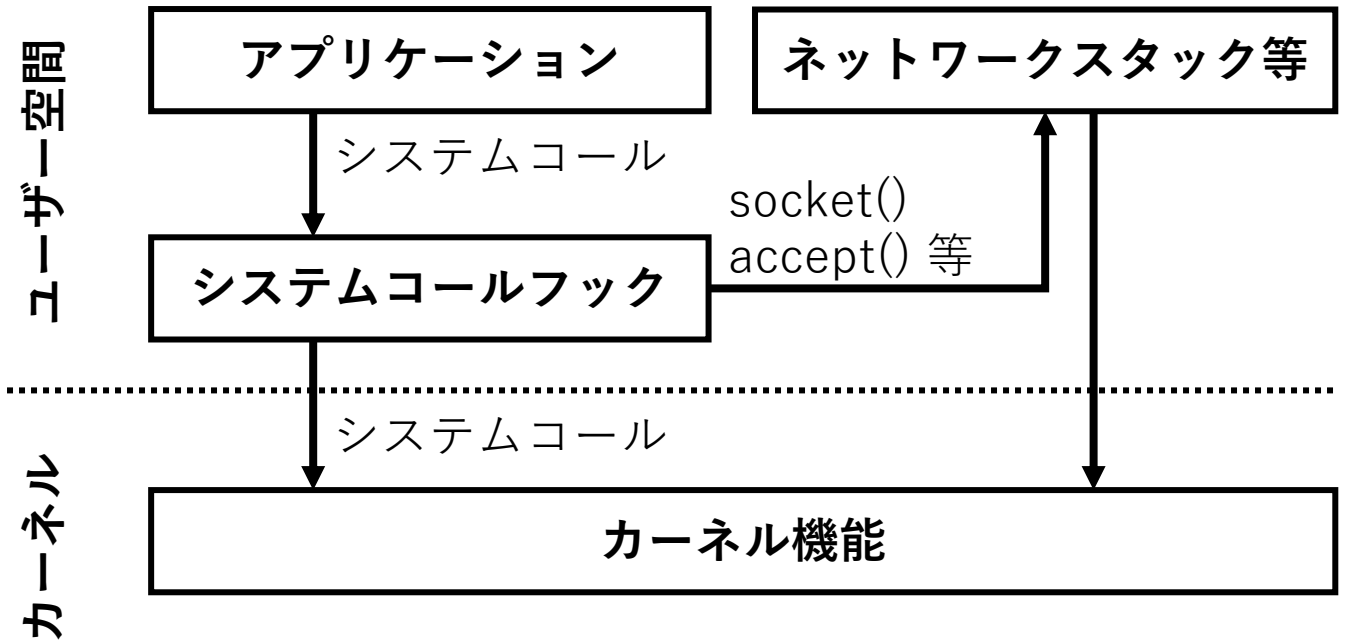
yasukata

発表について

- Zpoline というバイナリを書き換えることでシステムコールをフックする仕組みを紹介します
- x86-64 CPU 上で動作する Linux を想定しています
- ソースコードは以下の URL よりご参照ください
  https://github.com/yasukata/zpoline

モチベーション

やりたいと思ったこと



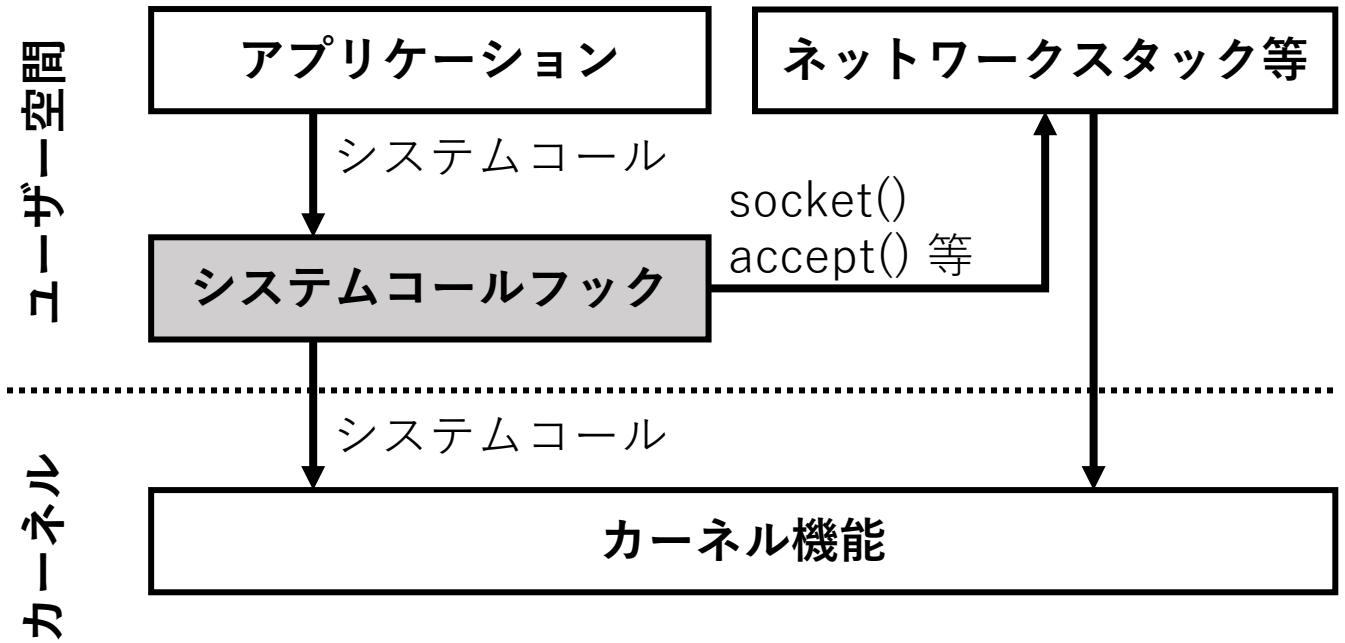
カーネルに実装されている機能を
アプリケーションを変更せずに
ユーザー空間で置き換えたい

例えばネットワークスタック等

システムコールをフックして
適宜ユーザー空間実装を実行する
ようにすればよさそう

モチベーション

やりたいと思ったこと



カーネルに実装されている機能を
アプリケーションを変更せずに
ユーザー空間で置き換えたい

例えばネットワークスタック等

システムコールをフックして
適宜ユーザー空間実装を実行する
ようにすればよさそう

もしかすると、この用途に合ったシステムコールをフックする仕組みがない？？

モチベーション：具体的なフックの要件

1. フック適用後のアプリケーションの性能劣化が小さい
2. フック適用の確度が高い（フックし損ねない）
3. ユーザー空間プログラムの再コンパイルが不要
4. カーネルの変更が不要、カーネルモジュールも不要

モチベーション：既存の仕組みと問題点

既存の仕組み	性能	確度
既存のカーネル機能 (ptrace, Syscall User Dispatch)		✓
ライブラリ関数の置き換え (LD_PRELOAD)	✓	
既存のバイナリ書き換え手法	✓	

問題：既存の仕組みでは性能と確度を両立できない！

今回のモチベーション：両立できる仕組みが欲しい！

今回紹介する仕組み：Zpoline

- バイナリ書き換えでシステムコールをフックする仕組み

性能劣化を抑えやすい！
が、既存の仕組みでは確度に欠ける（フックし損ねることがある）

Zpoline が取り組む課題

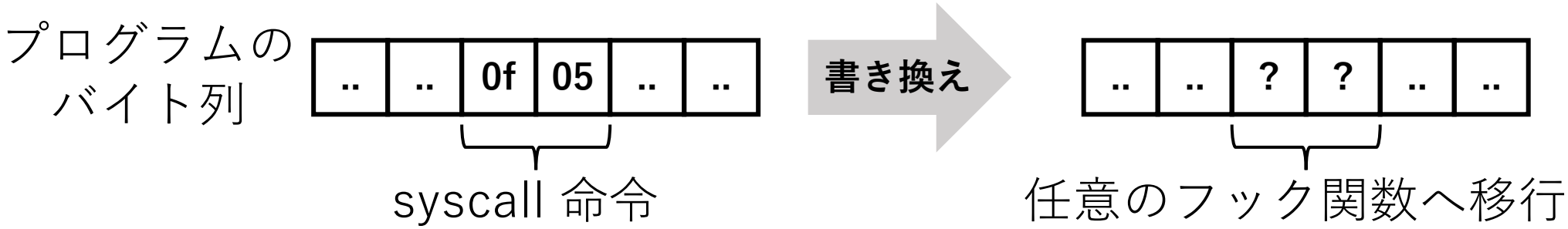
どうすれば、バイナリ書き換えでフックし損ねないようにできるか？

具体的なバイナリ書き換え固有の難しさ

- x86-64 CPU でシステムコールを発行する CPU 命令
  - syscall 命令：オペコード 0x0f 0x05
  - sysenter 命令：オペコード0x0f 0x34

それぞれ**2バイト**の命令

<u>やりたいこと</u>： syscall / sysenter 命令を置き換えて
任意のフック関数のアドレスへジャンプしたい



具体的なバイナリ書き換え固有の難しさ

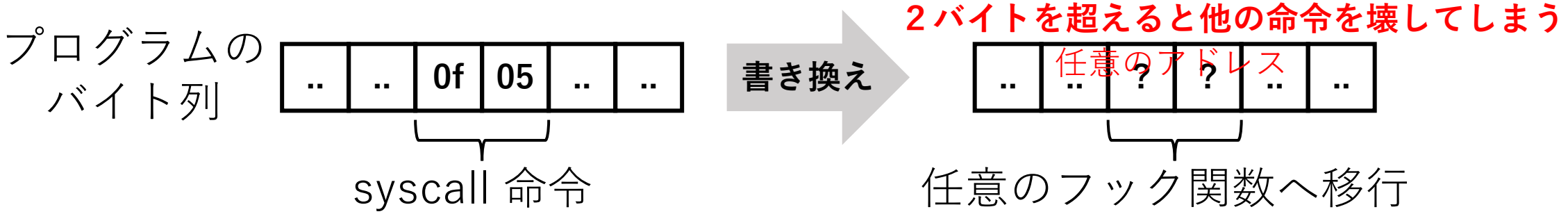
- x86-64 CPU でシステムコールを発行する CPU 命令
  - syscall 命令：オペコード 0x0f 0x05
  - sysenter 命令：オペコード0x0f 0x34

それぞれ**2バイト**の命令

やりたいこと： syscall / sysenter 命令を置き換えて
任意のフック関数の**アドレス**へジャンプしたい

難しさ：任意のアドレスを指定するのに2バイトでは小さい！

この問題のために、既存のバイナリ書き換えの仕組みは確実な置き換えを保証できない



Zpoline のアイデア

- ２バイトでジャンプ先のアドレスを指定するのは難しそう、、、

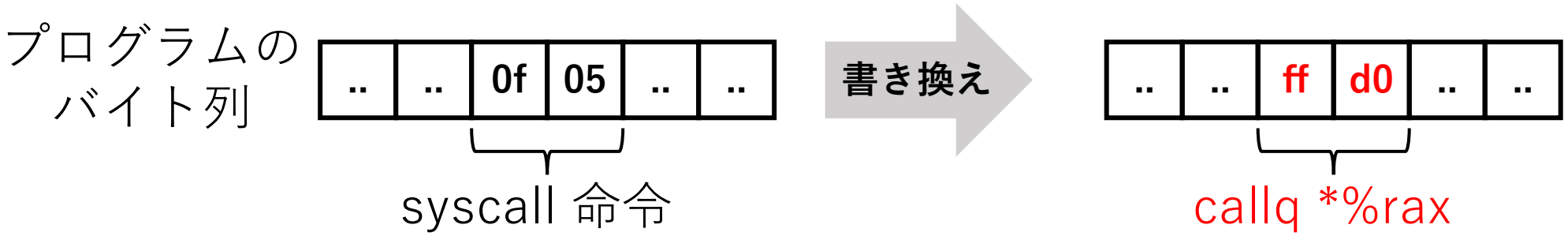
Zpoline のアイデア

1. システムコールの呼出規約を利用した書き換えを行い
2. 適切にトランポリンコードを用意する

Zpoline でのバイナリ書き換え

- Zpoline は syscall / sysenter 命令を **callq *%rax** へ書き換える

ポイント：callq *%rax はオペコードが 0xff 0xd0 の 2 バイトなので、
syscall / sysenter 命令をそのまま置き換えられる！



callq *%rax：rax レジスタの値を宛先アドレスとしてジャンプする

➡ どうなる？

x86-64 CPU 上の Linux の

システムコールの呼出規約 (呼び出し方)

- ユーザー空間プログラムは利用したい
  システムコールの番号を
  rax レジスタへ入れた後、
  syscall / sysenter 命令を実行する

システムコール番号

read	0
write	1
…	400~500 くらい

ポイント　syscall / sysenter が実行されるときには
rax レジスタにシステムコール番号が入っている

syscall / sysenter を callq *%rax で置き換えると
アドレス 0 から 400 ~ 500 程度までのジャンプになる！

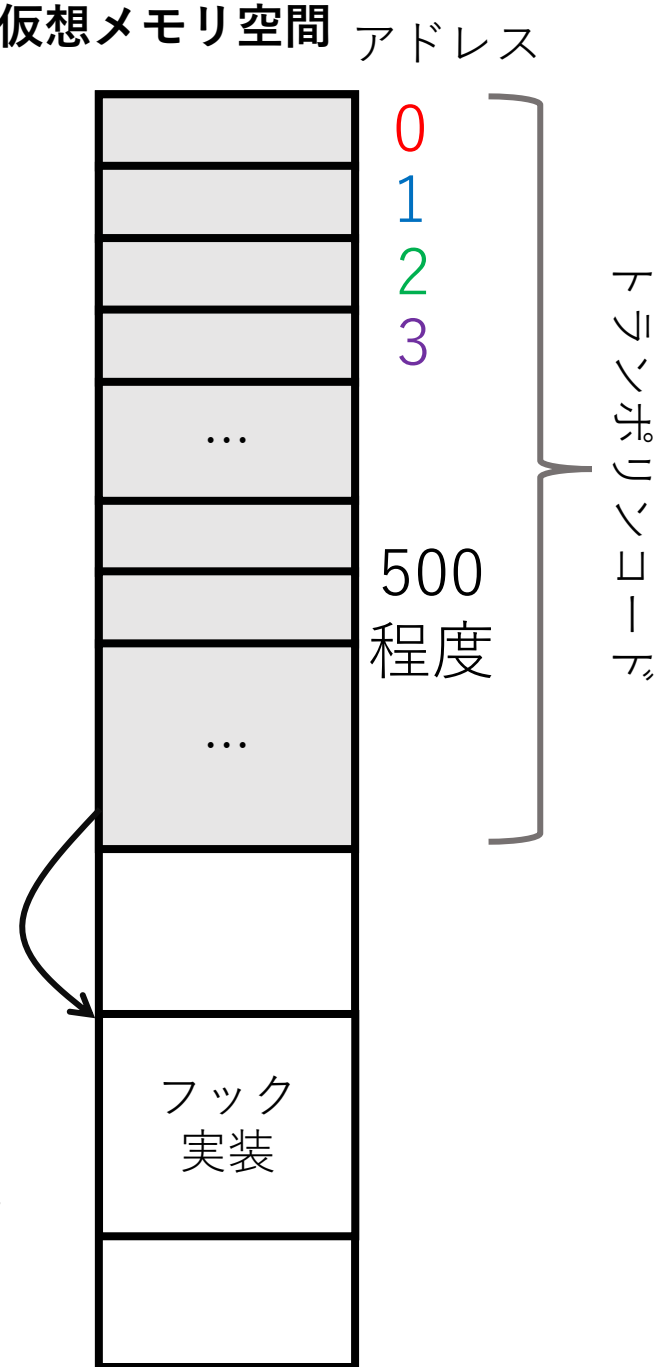
トランポリンコード

- callq *%rax でジャンプしてくるアドレス 0 から 500 程度までを含む領域にトランポリンコードを用意する

- Linux では、以下のように procfs から設定すると mmap でアドレス 0 にメモリを確保できるようになります

```
$ sudo sh -c "echo 0 > /proc/sys/vm/mmap_min_addr"
```

Zpoline の名前はアドレス 0 (**Z**ero) に置かれる tram**poline** コードから来ています



トランポリンコード

- システムコール番号の数だけ
  先頭を nop (0x90) で埋める
- 次にフックへのジャンプのコードを置く
- callq *%rax で飛んだ後は、nop を辿って
  フックへジャンプする処理まで到着する

システムコール番号

```
read  0
write 1
open  2
close 3
…
…     N
```

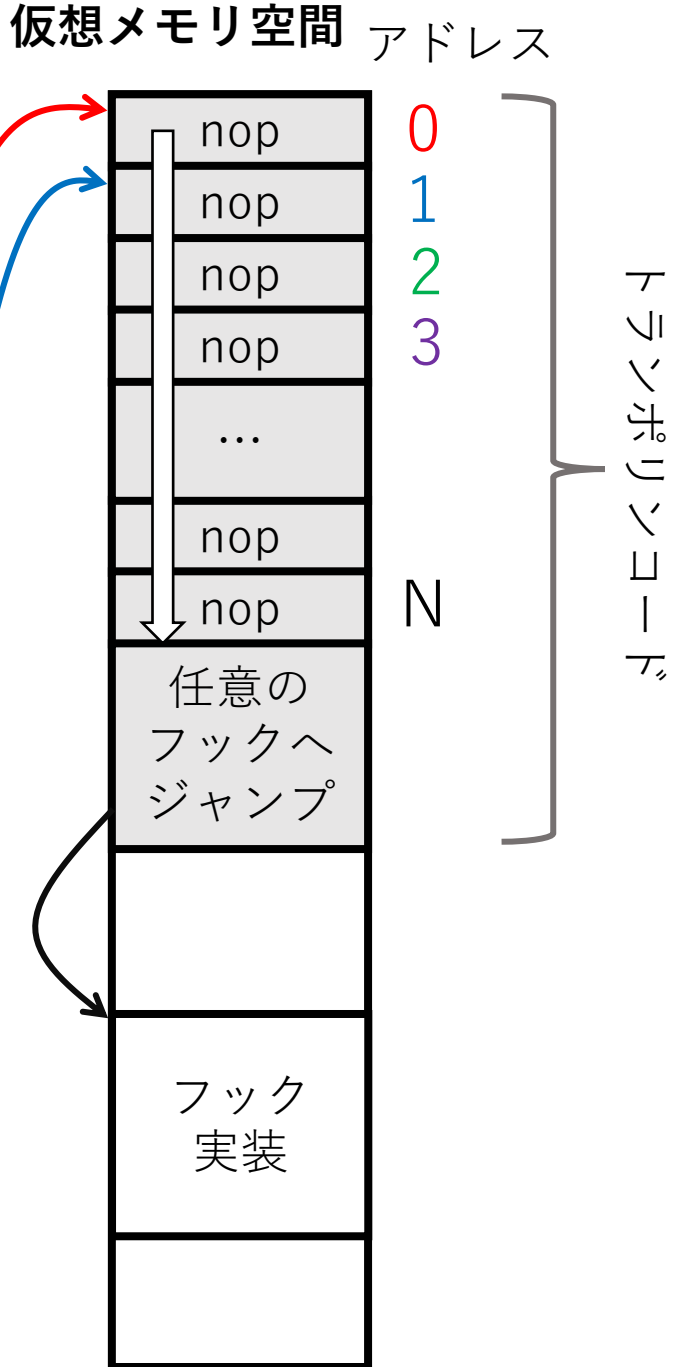
最大

read システムコール

```
movq $0, %rax
callq *%rax
```

write システムコール

```
movq $1, %rax
callq *%rax
```



実装

- 今回、Zpoline の初期化は LD_PRELOAD でロードされることを想定した共有ライブラリとして実装

- 以下のような感じで実行すると、トランポリンコードの用意とバイナリ書き換えを a.out 内の main() 開始前に実行

```
$ LD_PRELOAD=libzpoline.so ./a.out
```

- バイナリ書き換えはメモリにロードされたプログラムに対して行うので、プログラムファイル自体は変更しない

フックのオーバーヘッド

- 環境：Linux 5.11 on Intel Xeon E5-2640 v3 CPU 2.60 GHz
- システムコールフック適用後に、getpid() を 1 回実行するために必要な CPU サイクル数を計測
  - getpid システムコールを実行して結果を返す（pid キャッシュなし）
  - メモリ上にキャッシュした pid の値を返す（pid キャッシュあり）

フックの仕組み	pid キャッシュなし	pid キャッシュあり
ptrace	17820	16403
Syscall User Dispatch	5957	4563
Zpoline	1459	**138**

ptrace より 100 倍以上高速！

まとめ

- Zpoline というバイナリを書き換えることでシステムコールをフックする仕組みを紹介しました

- Zpoline のソースコードは以下の URL にありますので、是非、試してみてください（GitHub）

  https://github.com/yasukata/zpoline

- 具体的なフック関数のプログラミングの方法については、以下の URL の記事をご参照ください（ブログ）

  https://yasukata.hatenablog.com/entry/2021/11/19/144708

その他リソースへのリンク

- 少しだけ詳しい説明（ブログ）
  https://yasukata.hatenablog.com/entry/2021/10/14/145642

- 少しだけ詳しい英語での説明（GitHub）
  https://github.com/yasukata/zpoline/blob/master/Documentation/README.md