SecHack365

prpara

既存プロセスを動作させたまま コードを寄生させるRust製ツール

学習駆動コース　坂井ゼミ　松田来央

プロセスを止めずに修正や更新したいこと、ありませんか？

一般的な更新

修正や更新はプロセスを新たに立ち上げるのが一般的でもサーバ等は動作を止めたくない…

prparaでの更新

prparaであれば、**動作中のプロセスを止めずに書き換え**可能！

サーバで動かすようなプログラムは常に動作させることが求められます。その一方で、プログラムを安全に保つためには定期的な更新が必要です。
prparaは、動作中のプロセスに後から共有ライブラリ(.so)をロードさせ、関数の機械語コードをロードした共有ライブラリへジャンプさせるように書き換えるツールです。
prparaを利用することで、**「脆弱性が見つかったけど、稼働中で止められない…」**といったような状況でも、**修正を適用するといったことが実現**できます。

prparaで解決！

特徴

特別な追加作業は不要！

prparaではLinuxカーネルに存在するptraceとmmapというシステムコールを利用して動作を実現しています。プロセス起動時に特別な共有ライブラリを読み込んだりせず、**prparaとLinuxの標準システムコールのみで動く**ため、”この瞬間に”動いているプロセスに対しても使えます。

Rust製！

prparaは全てRustで書かれているため、**コンパイラにより安全性**を高めつつ、prparaを利用した新たなプログラムが書きやすくなっています。

オープンソース！

prparaは**GPLでオープンソースとして公開**しています。prparaは一歩間違えると対象プロセスに危害を加えてしまうため、あえてGPLを使いprpara自体とそれを利用したプログラムにも**透明性**を高めています。

名前の由来

ptrace　**r**ust　**para**site

prparaの名前は、構成要素の
- ptraceの**p**
- Rust言語の**r**
- 寄生’parasite’の**para**

から取って名付けました。

使い方

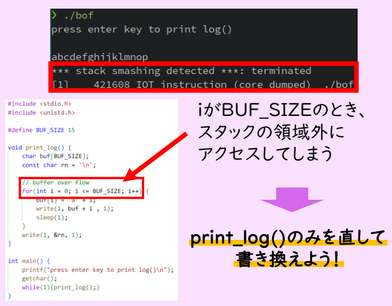
CUIアプリとしての使用方法

Shell(superuser)

```
# prpara -p [PID] -l [.so file path] -f [old symbol]:[new symbol]
```

CUIアプリデモ

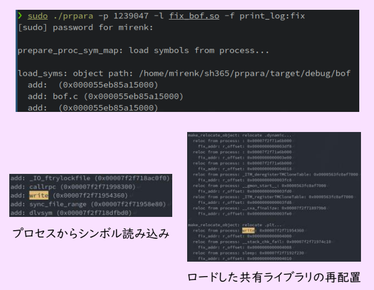
buf[]を超えると落ちてしまう。



iがBUF_SIZEのとき、スタックの領域外にアクセスしてしまう

print_log()のみを直して書き換えよう！

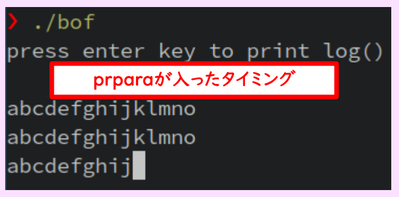
落ちる前にprparaで関数を更新！



プロセスからシンボル読み込み

ロードした共有ライブラリの再配置

以前落ちていた箇所で落ちなくなる！



Rustのクレートとして利用

Cargo.toml

```
...
[dependencies]
+ prpara = { git = "https://github.com/Mirenk/prpara" }
```

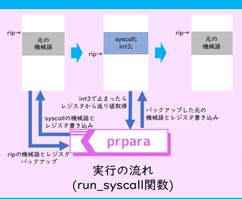
prpara大解剖！内部処理の解説

対象プロセスへの処理

対象プロセス”で”mmap(2)実行

```
// set args to regs
regs.rax = SYS_mmap as u64;
regs.rdi = addr as u64;
regs.rsi = len as u64;
regs.rdx = prot as u64;
regs.rcx = flags as u64;
regs.r10 = flags as u64;
regs.r8 = fd as u64;
regs.r9 = offset as u64;
```

mmap()呼び出し準備
(prepare_mmap関数)



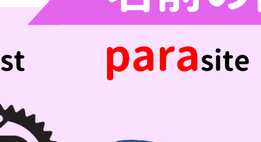
実行の流れ
(run_syscall関数)

共有ライブラリを配置するためには、対象プロセスに実行可能なメモリ領域を確保する必要があります。ptrace(2)で呼ぶときにはラッパー関数が使えないため、自力で呼び出し規則に従わなければなりません。
幸いx86_64 Linuxのmmap(2)システムコールの呼び出し規則は、全ての引数をスタックではなくレジスタに格納する仕様となっています。
prparaでレジスタの準備をした後、syscall命令とint3命令(0x0f05cc)をアタッチ直後のripレジスタが指すアドレスに書き込んで、実行再開することで実現しています。

指定アドレスへのジャンプコード

```
let mut jmp_buf: Vec<u8> = Vec::new();
jmp_buf.push(0xff);
jmp_buf.push(0x25);
jmp_buf.push(0x00);
jmp_buf.push(0x00);
jmp_buf.push(0x00);
jmp_buf.push(0x00);
```

書き換え後の命令:
jmp QWORD PTR [rip+0x0](直後を指す)
0x00007fxxxxxxxxxx (mmap+offset)

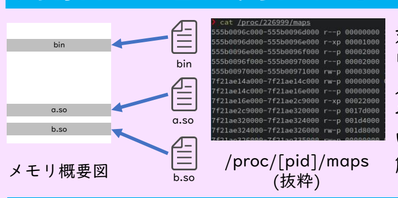


メモリ概要図

元の関数の位置に、ロードした共有ライブラリの関数アドレスにジャンプする命令を書かなくてはならないのですが、x86_64では元のアドレスからmmap(2)を行ったアドレスまで4byte以上の距離があり、相対ジャンプができません。
したがって、ジャンプコードの直後に絶対アドレスを置き、ジャンプコードを「相対アドレスに格納されているアドレスにジャンプ」として対応しました。

共有ライブラリのロード

対象プロセス内シンボルのアドレス解決

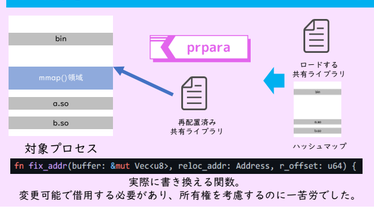


メモリ概要図

/proc/[pid]/maps
(抜粋)

対象プロセスへ共有ライブラリを読み込む前に、対象プロセス内で使っているバイナリがどのように仮想アドレスに割り当たっているかを調べる必要があります。
メモリへのマップ情報は/proc/[pid]/mapsで確認することができます。
それぞれのバイナリの先頭アドレスを調べ、バイナリのシンボルテーブルに書かれているオフセットを加算し、prparaでハッシュリストとして保持することで、シンボル名で解決できるようにしています。

共有ライブラリのアドレス書き換え



対象プロセス

```
fn fix_addr(buffer: &mut Vec<u8>, reloc_addr: Address, r_offset: u64) {
```

実際に書き換える関数。
変更可能で借用する必要があり、所有権を考慮するのに一苦労でした。

読み込む共有ライブラリでprintf()を呼んでいるような、「共有ライブラリが対象プロセスの既にある関数を呼ぶ」という動作を実現するためには、アドレスの再配置が必要になります。
前項で対象プロセスの全シンボルがハッシュリストで保持されているため、ロードする共有ライブラリの再配置テーブルを見て、アドレスを書き込んでいきます。
Rust言語に書き直した際、共有ライブラリをバッファに読み込んで書き換えるという動作が、Cのようにmallocして終わりではなく、とても意識を注ぐ必要がありました。

ソースコード

製作過程

当初はデバッガの拡張から始まり、様々な試作物を作りながらprparaを完成させました。試作物もGitHubで公開しています。

4月～6月	GDBの拡張を作成
6月～7月	ptraceと出会い、簡易デバッガ”predb”の作成
8月	ptraceを利用したLinux用FreeBSDエミュレータ”fbsd_emu”の作成
9月～12月	C言語でprparaの前身を開発
12月～	Rust言語でprparaを再実装

作品の今後とSecHack365の感想

この作品の今後として、現状x86_64のLinuxのみの対応となっているため、IoT機器に多く使われているarm64アーキテクチャに対応させたいと考えています。また、ブログで試作物等の詳細を書くことも検討しています。

今回のSecHack365では、初めてシステムコールプログラミングとRust言語を自ら学ぶきっかけになりました。また、初めてのハッカソンだったのですが、最初はやってみる・作ってみることでモノづくりの楽しさが改めて実感でき、その後どう活用するかの段階でアイディアが出ず、とても悩みました。そんな1年を通して得た貴重な経験と学びを、これからの開発で活かしていきたいです。

NICT National Institute of Information and Communications Technology
GItHub:Mirenk　Twitter:@M2renk　blog: https://blog.mirenk.net/　••• more: https://mirenk.net/