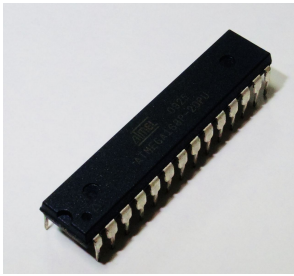
『Binary Hacks Rebooted』の紹介

Binary Hacks Rebooted ～ Forkwell Library#68 (2024/10/03)

Akira Kawata
https://akawashiro.com/
@a_kawashiro

河田 旺 (かわた あきら)



- 『Binary Hacks Rebooted』のまとめ役
  - ELF Hack の章を担当
- https://akawashiro.com/
- @a_kawashiro
- https://github.com/akawashiro/
- 仕事では機械学習用ASICのコンパイラ・ランタイムを作っています
- 最近 実行ファイルのスペースにホワイトスペースを入れる という発表をしました

『Binary Hacks Rebooted』とは

- 低レイヤプログラミングのテクニックを集めた本
- 2006年に出版された『Binary Hacks』の後継
  - 内容は全て刷新
  - 重複はないです
  - 『Binary Hacks』を持っていても (むしろ持っている方が) 楽しめる

最も伝えたいこと

低レイヤプログラミングは楽しい

想定読者層

- 低レイヤプログラミングを楽しみたい人
  - 目の前のパソコンで何が起こっているのか知りたい
  - 好きだけどいまいち何をしたらいいのかわからない
  - なんとなく興味はあるが取っ掛かりが欲しい
- 低レイヤ関連で困っている方
  - 明文化されていないが常識とされていることを知りたい
  - もう一歩踏み込んだ理解をしたい
  - よくわからないが動かなくなった、をデバッグしたい

『Binary Hacks Rebooted』の読み方

- 頭から読んで全部読んでいく本ではない
- パラパラ眺めて面白そうなところを読むのがオススメ

『Binary Hacks Rebooted』の章立て

- 1章　イントロダクション
- 2章　ELF Hack (河田)
- 3章　OS Hack (佐伯さん)
- 4章　コンテナHack (佐伯さん)
- 5章　デバッガ・トレーサHack
- 6章　セキュリティHack
- 7章　数値表現とデータ処理Hack
- 8章　言語処理系Hack (光成さん)
- 9章　そのほかのHack

『Binary Hacks Rebooted』推しポイント

- 1つ1つの Hack の品質が高い
  - 非常に深い
  - 前回は、Hacks 的なノリで書いてた気がするけど、今回はみんな論文かよというくらいのレビューをしていて、品質は Hacks 的なノリを超えてると思いますね。
- 遊び心が満載
  - ページ番号が 2進数と16進数でかいてあったりとか
  - 謎の模様が入っていたりとか

推し Hack の紹介

「#3 Hello, World! 再訪」

- Hello, World! と出力するシンプルなプログラムを掘り下げる Hack
- Hello, World! を出力する方法を 9ページにわたって詳細に解説
- 身近なプログラムを深く理解することは重要

素朴な Hello, World!

```
#include <stdio.h>
int main() {
    puts(“Hello, World!\n”);
    return 0;
}
```

素朴な Hello, World!

```
#include <stdio.h>
int main() {
    puts(“Hello, World!\n”);
    return 0;
}
```

ここでは **printf** を使っていない。
printf を書式文字列なしで利用すると
puts に置き換えられてしまうことが多く、
バイナリを読んだときに混乱するため

Hello, World! の中身

```
$ objdump --disassemble=main -M intel hello_c
0000000000001149 <main>:
    1149:     f3 0f 1e fa              endbr64
    114d:     55                       push   rbp
    114e:     48 89 e5                 mov  rbp,rsp
    1151:     48 8d 05 ac 0e 00 00     lea  rax,[rip+0xeac]  #  2004<_IO_stdin_used+0x4>
    1158:     48 89 c7                 mov  rdi,rax
    115b:     e8 f0 fe ff ff           call   1050 <puts@plt>
    1160:     b8 00 00 00 00           mov  eax,0x0
    1165:     5d                       pop  rbp
    1166:     c3                       ret
```

Hello, World! の中身

```
$ objdump --disassemble=main -M intel hello_c
0000000000001149 <main>:
    1149:     f3 0f 1e fa             endbr64
    114d:     55                      push   rbp
    114e:     48 89 e5                mov    rbp,rsp
    1151:     48 8d 05 ac 0e 00 00    lea    rax,[rip+0xeac]  #  2004<_IO_stdin_used+0x4>
    1158:     48 89 c7                mov    rdi,rax
    115b:     e8 f0 fe ff ff          call   1050 <puts@plt>
    1160:     b8 00 00 00 00          mov    eax,0x0
    1165:     5d                      pop    rbp
    1166:     c3                      ret
```

Intel CET というセキュリティ機構を実現するための命令。
Hack 57 を参照。

Hello, World! の中身

```
$ objdump --disassemble=main -M intel hello_c
0000000000001149 <main>:
    1149:     f3 0f 1e fa             endbr64
    114d:     55                      push   rbp
    114e:     48 89 e5                mov    rbp,rsp
    1151:     48 8d 05 ac 0e 00 00    lea    rax,[rip+0xeac]  #  2004<_IO_stdin_used+0x4>
    1158:     48 89 c7                mov    rdi,rax
    115b:     e8 f0 fe ff ff          call   1050 <puts@plt>
    1160:     b8 00 00 00 00          mov    eax,0x0
    1165:     5d                      pop    rbp
    1166:     c3                      ret
```

フレームポインタをスタックに退避。
Hack 59を参照。

素朴な Hello, World!

```
$ objdump --disassemble=main -M intel hello_c
0000000000001149 <main>:
    1149:    f3 0f 1e fa              endbr64
    114d:    55                       push   rbp
    114e:    48 89 e5                 mov    rbp,rsp
    1151:    48 8d 05 ac 0e 00 00     lea    rax,[rip+0xeac]  #  2004<_IO_stdin_used+0x4>
    1158:    48 89 c7                 mov    rdi,rax
    115b:    e8 f0 fe ff ff           call   1050 <puts@plt>
    1160:    b8 00 00 00 00           mov    eax,0x0
    1165:    5d                       pop    rbp
    1166:    c3                       ret
```

フレームポインタを更新。この main 関数にはローカル変数がないので rbp = rsp となるが、一般には rsp を減算してスタックを確保する処理がこの直後に入る

Hello, World! の中身

```
$ objdump --disassemble=main -M intel hello_c
0000000000001149 <main>:
    1149:     f3 0f 1e fa              endbr64
    114d:     55                       push   rbp
    114e:     48 89 e5                 mov    rbp,rsp
    1151:     48 8d 05 ac 0e 00 00     lea    rax,[rip+0xeac]  #  2004<_IO_stdin_used+0x4>
    1158:     48 89 c7                 mov    rdi,rax
    115b:     e8 f0 fe ff ff           call   1050 <puts@plt>
    1160:     b8 00 00 00 00           mov    eax,0x0
    1165:     5d                       pop    rbp
    1166:     c3                       ret
```

rax レジスタに “Hello, World!” の先頭アドレスを設定

素朴な Hello, World!

```
$ objdump --disassemble=main -M intel hello_c
0000000000001149 <main>:
    1149:    f3 0f 1e fa              endbr64
    114d:    55                       push   rbp
    114e:    48 89 e5                 mov    rbp,rsp
    1151:    48 8d 05 ac 0e 00 00     lea    rax,[rip+0xeac]  #  2004<_IO_stdin_used+0x4>
    1158:    48 89 c7                 mov    rdi,rax
    115b:    e8 f0 fe ff ff           call   1050 <puts@plt>
    1160:    b8 00 00 00 00           mov    eax,0x0
    1165:    5d                       pop    rbp
    1166:    c3                       ret
```

rdi レジスタにrax の値を代入。x86-64 向けSystem V ABI では第一引数は rdi レジスタに入れることになっている。この rdi の値が puts の引数になる。Hack 83 を参照。

Hello, World! の中身

```
$ objdump --disassemble=main -M intel hello_c
0000000000001149 <main>:
    1149:    f3 0f 1e fa              endbr64
    114d:    55                       push   rbp
    114e:    48 89 e5                 mov    rbp,rsp
    1151:    48 8d 05 ac 0e 00 00     lea    rax,[rip+0xeac]  #  2004<_IO_stdin_used+0x4>
    1158:    48 89 c7                 mov    rdi,rax
    115b:    e8 f0 fe ff ff           call   1050 <puts@plt>
    1160:    b8 00 00 00 00           mov    eax,0x0
    1165:    5d                       pop    rbp
    1166:    c3                       ret
```

puts 関数を呼び出す

Hello, World! の中身

```
$ objdump --disassemble=main -M intel hello_c
0000000000001149 <main>:
    1149:    f3 0f 1e fa              endbr64
    114d:    55                       push   rbp
    114e:    48 89 e5                 mov    rbp,rsp
    1151:    48 8d 05 ac 0e 00 00     lea    rax,[rip+0xeac]  #  2004<_IO_stdin_used+0x4>
    1158:    48 89 c7                 mov    rdi,rax
    115b:    e8 f0 fe ff ff           call   1050 <puts@plt>
    1160:    b8 00 00 00 00           mov    eax,0x0
    1165:    5d                       pop    rbp
    1166:    c3                       ret
```

main 関数の戻り値である 0 を eax レジスタに代入。関数の返り値は eax レジスタを使うことが ABI で決まっている。

素朴な Hello, World!

```
$ objdump --disassemble=main -M intel hello_c
0000000000001149 <main>:
    1149:    f3 0f 1e fa              endbr64
    114d:    55                       push   rbp
    114e:    48 89 e5                 mov    rbp,rsp
    1151:    48 8d 05 ac 0e 00 00     lea    rax,[rip+0xeac]  #  2004<_IO_stdin_used+0x4>
    1158:    48 89 c7                 mov    rdi,rax
    115b:    e8 f0 fe ff ff           call   1050 <puts@plt>
    1160:    b8 00 00 00 00           mov    eax,0x0
    1165:    5d                       pop    rbp
    1166:    c3                       ret
```

スタックに退避していたフレームポインタを復帰する。

Hello, World! の中身

```
$ objdump --disassemble=main -M intel hello_c
0000000000001149 <main>:
    1149:    f3 0f 1e fa              endbr64
    114d:    55                       push   rbp
    114e:    48 89 e5                 mov    rbp,rsp
    1151:    48 8d 05 ac 0e 00 00     lea    rax,[rip+0xeac]  #  2004<_IO_stdin_used+0x4>
    1158:    48 89 c7                 mov    rdi,rax
    115b:    e8 f0 fe ff ff           call   1050 <puts@plt>
    1160:    b8 00 00 00 00           mov    eax,0x0
    1165:    5d                       pop    rbp
    1166:    c3                       ret
```

main 関数から戻る

もっと Hello, World!

- 「#3 Hello, World!再訪」 では更に以下を紹介
  - puts を使わない C言語での Hello, World!
  - アセンブリでの Hello, World!
  - バイナリファイル中に埋め込みやすい Hello, World!
  - 機械語を手で書く Hello, World!

まとめ

- 低レイヤは楽しい
  - いつも使っているもののがどう動いているかを知るのは楽しい
- 低レイヤは役に立つ
  - 自分が使っているレイヤの一段、二段下を知っているとデバッグ、パフォーマンスチューニングが大変捗る
  - 低レイヤから作る仕事に就職した場合はもっと役に立つ

終わり