SecHack365

Depth ~実行プログラム基盤~

学習駆動 坂井ゼミ
菅原大和

Motivation

情報技術の進歩とプログラミング技術の進歩は密接にあると考える,
その中でも高級な言語を低級な言語に翻訳する「コンパイラ」技術は,
全ての技術の基盤であり,今日に至るまで盛んに研究され続けている.
しかし,コンパイラ自作に挑戦する人たちの中でも
“フルスクラッチ”でバイナリフォーマット準拠の実行プログラムまで
生成できる「コンパイラドライバ」を開発する人は少ない.
私はx86_64アーキテクチャ向けのコンパイラドライバを実際に開発する事で,
「ミニマムな実行プログラム生成基盤」の実装に挑戦した.
また,生成基盤に多数の実行プログラム解析ツールを組み込む事で,
最終的に「実行プログラム基盤」となるまでを実装した.

History

神奈川回まで : Golangによる実装
北海道回まで : Go実装全削除, Rustでの実装開始
福岡回まで　 : 全ソースコード削除, 再度実装しなおし
宮城回まで　 : 実行可能プログラムの生成に成功
愛媛回まで　 : LLVM IRの生成パスを追加,全文法に対応
沖縄回まで　 : readelf, checksec, 独自セキュリティ機構

```
depth 14:57:58 drumato (master)$ cloc src/
      58 text files.
      58 unique files.
       0 files ignored.

github.com/AlDanial/cloc v 1.81  T=0.02 s (2541.9 files/s, 394077.4 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Rust                            56            287            167           8451
YAML                             1              0              0             86
JSON                             1              0              0              1
-------------------------------------------------------------------------------
SUM:                            58            287            167           8538
-------------------------------------------------------------------------------
```

実行プログラム生成基盤

コンパイラ

- x64 Intel assembly
- Linux向けのみ対応
- LLVM IR パスも存在

アセンブラ

- ELF64 Object file
- 再配置可能な状態

ELFローダ

- ユーザ空間で実装
- execve(2)を用いている

リンカ

- ELF64 static link

協調動作

```
~ 15:11:11 drumato $ cat d.dep
struct Depth {
  foo: i64
  bar: i64
}

func main() :: i64 {
  let depth : Depth = Depth{ foo : 30, bar: 60 }
  let mut x : i64 = 0
  if (depth.foo == 30) {
    x = depth.foo
  } else {
    x = depth.bar
  }
  return x
}
~ 15:11:14 drumato $ depth d.dep --run ; echo $?
30
```

コンパイラがアセンブリを生成し,
アセンブラがオブジェクトファイルに変換.
それをリンカが受け取って
実行可能なプログラムに変換した後,
ローダがメモリアロケート,
プログラムをロードして実行している.

実行プログラム解析基盤

readelf

- 基本的な機能は網羅
- 独自セクションのパースに対応

checksec

- 他のコンパイラドライバのバイナリが対象
- ELFバイナリのフラグをチェック

独自セクション
・バイナリに自然言語コメントを埋め込み
・シンボルの名前・型情報等も埋め込み
・独自readelfを使ってパース,表示可能

協調動作

```
~ 15:48:31 drumato $ cat d.dep
type PtrToInt = Pointer<i64>

@info add two integer then return pointer to the result #
func returnI64(x: i64, y: i64) :: PtrToInt {
  let x : i64 = x + y
  return &x
}

@info this is a entry point of a program #
func main() :: i64 {
  let ptr : PtrToInt = returnI64(10, 20)
  return *ptr
}
~ 15:48:33 drumato $ depth d.dep ; depth --readelf --debug a.out


Debug table '.dbg.depth' contains 2 entries:
+-------------+------------+------+---------+
| ReturnType | ArgNumbers | Name | ArgType |
+-------------+------------+------+---------+
| Pointer<i64> | 0x2 | returnI64 | (i64, i64) |
+-------------+------------+------+---------+
| i64 | 0x0 | main | void |
+-------------+------------+------+---------+

+------+-----------+
| Name | Documents |
+------+-----------+
| returnI64 | add two integer then return pointer to the result |
+------+-----------+
| main | this is a entry point of a program |
+------+-----------+
```

実行プログラム生成基盤が
実行プログラムにデバッグセクションを埋め込み,
独自readelfがそのセクションを読み込んでいる.
実行プログラムに埋め込まれる事で
プロセスメモリにロードすることができ,
プログラム実行時のエラーを表示する際に
このセクションを利用して詳細な情報を
共有できる.(未実装)

NICT 国立研究開発法人 情報通信研究機構