卒論ミーティング (11/24)

31010 佐藤秀明

概要

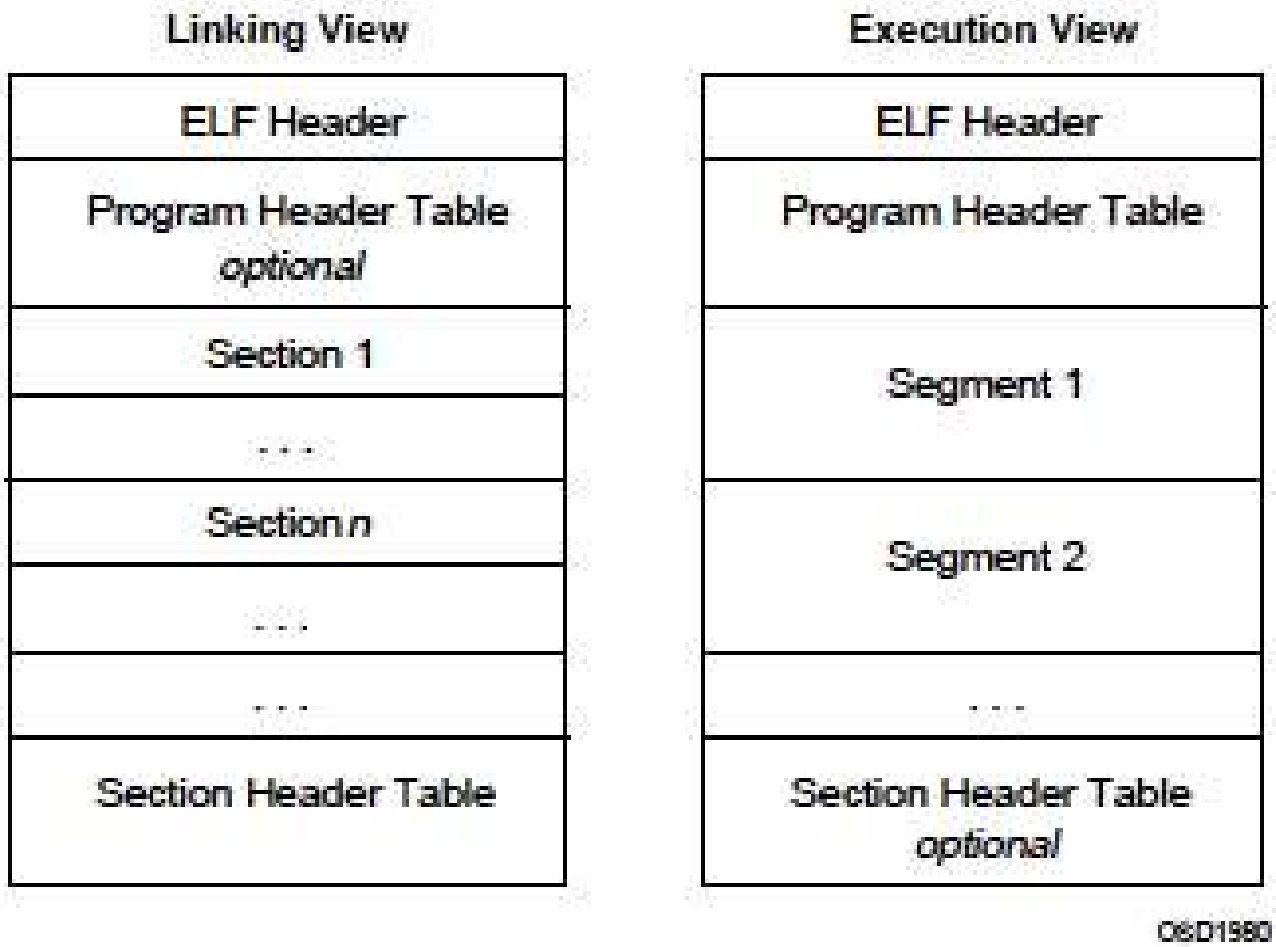
- ELF format について
  - Section と segment
- ファイルの実行に影響を与えない情報を詐称
- 実装するシステムの構造
- これからの予定

ELF Format(1)

- Executable and Linkable Format[1]
- Unix 系で広く採用されているオブジェクトファイルのフォーマット
- ファイルの内部構造を示すヘッダが 2 種類ある
  - Section header table　…　リンカからの視点
    - ファイルを section という単位に分割する
  - Program header table　…　ローダからの視点
    - ファイルを segment という単位に分割する

ELF Format(2)

- ELF format なファイルの内部構造 (模式図)



ELF Format(3)

- 2 種類のヘッダを必ずしも両方持っていなければならないというわけではない
  - この先リンクの操作を必要としないなら section header table は必要ない
  - 実行可能ファイルでないなら program header table は必要ない

Tamperproofing の性質 (再掲)

- 通常の instruction はデータ領域を参照して演算を行う
  - メモリに対する load 、store
- Instruction 領域自体を参照する instruction は珍しい　→　怪しい
  - Tamperproof を行っている証拠
- どこがデータ領域でどこが instruction 領域か分かりづらくするとよい

情報を詐称する

- リンクまで済んだ実行可能ファイルについて、実行に影響を与えない内部情報を詐称する
  - Section header table
  - Symbol table
    - 一般的な逆アセンブラは上記 2 者の情報を元に逆アセンブルを行う
- Program header table も難読化する
  - Read/Write/Execute のアクセス制限をより厳しくするとプログラムの実行に支障が出る場合があるので注意する

システムの概要 (1)

1)ソースコードをコンパイルし、アセンブリを得る

2)Direct jump を indirect jump に変換する

3)アセンブリ中の各 section をぐちゃぐちゃにする

- データ領域に instruction を埋め込んだりする

4)プログラムの正当性をチェックするコードを埋め込む

- アドレスが解決してからでないと決まらないパラメータの部分はとりあえず blank にしておく

システムの概要 (2)

5)コードを動的に生成するコードを埋め込む
  - 元からあったコードの一部を静的にではなく動的に配置する

6)アセンブル・リンクし、実行可能ファイルを得る

7)Section header file と symbol table をぐちゃぐちゃにする

8)Program header file を難読化する

9)4) で blank にしておいたパラメータを計算して正しい値を埋め込む

ねらい

- 静的な解析は無理
  - Indirect jump
  - Self-generating code
- 動的な解析を困難にしたい
  - データ領域とinstruction 領域の境界を曖昧にする

これからの予定

- 実装に入る
  - アセンブリの操作
  - バイナリの操作
  - ELF header の操作

Reference

- [1]TIS Commitee. Tool Interface Standard, Executable and Linking Format Specification, Version 1.2, May 1995. http://www.x86.org/ftp/manuals/tools/elf.pdf.