全体ミーティング (12/14)

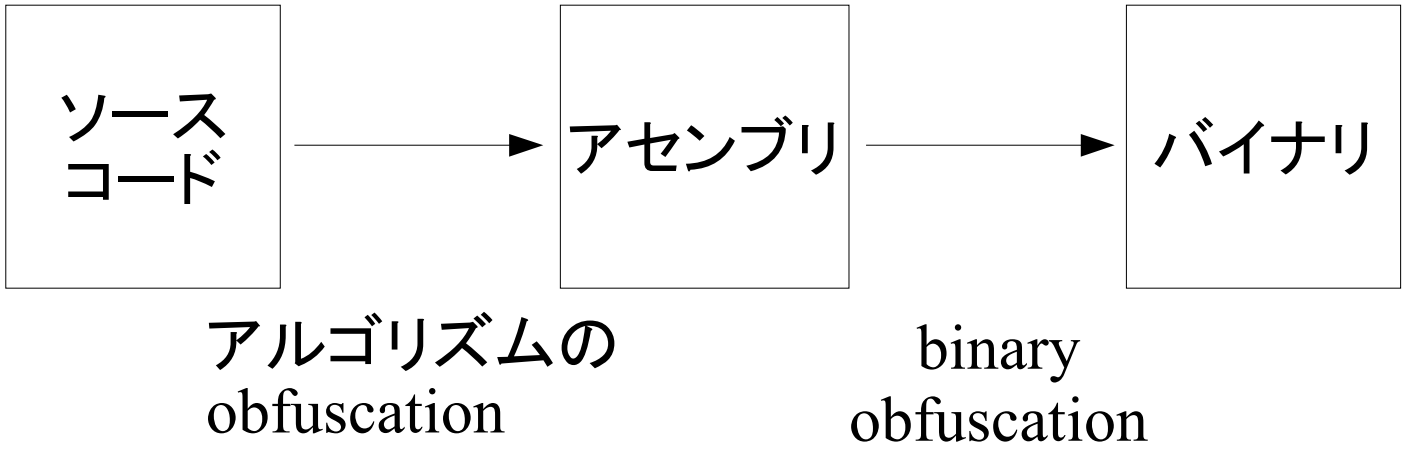
学部４年　31010　佐藤秀明

概要

- 卒論の進捗報告
  - 導入とねらい
    - RISC に対する binary obfuscation 、tamperproofing
    - 逆アセンブルと改変を難しくする
  - 実現方法
    - ELF header の詐称
    - Self-generating code
  - 実装
  - 今後の予定

Binary Obfuscation

- バイナリ (実行可能ファイルなど) を難読化し、逆アセンブルされにくくする技術 [2]
  - cf. アルゴリズムの obfuscation(逆コンパイルを難しくする)



Tamperproofing

- ソフトウェアが改変されたことを、そのソフトウェア自身が検知するシステム [1][3]
  - 改変を監視する対象は一般にプログラムのコード部
  - 検知システムはその存在を攻撃者に気づかれにくいものでなければならない (→Obfuscation との関連)

Binary obfuscation の既存研究

- Static な逆アセンブルに対して効果的な binary obfuscation の方法が存在 [2]
  - 命令と命令の間に junk byte をはさむ
    - 命令が可変長であるアーキテクチャ (x86) にしか使えない
- 固定長命令の環境で binary obfuscation を行うには別のアプローチが必要

Tamperproofing の既存技術 (1)

- コードの checksum やハッシュ値を比較 [1]
  - あらかじめ計算しておいた値と異なる場合、データが破壊される

```
start:
   [codes to be checked]
end:
   ...
   add %sp, -checksum
   mov start, %r1
for:
   cmp %r1, end
   bg next
   ld [%r1] , %r2
   add %sp, %r2, %sp
   add %r1, 4, %r1
   jmp for
next:
   ...
```

Tamperproofing の既存技術 (2)

- 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

方針

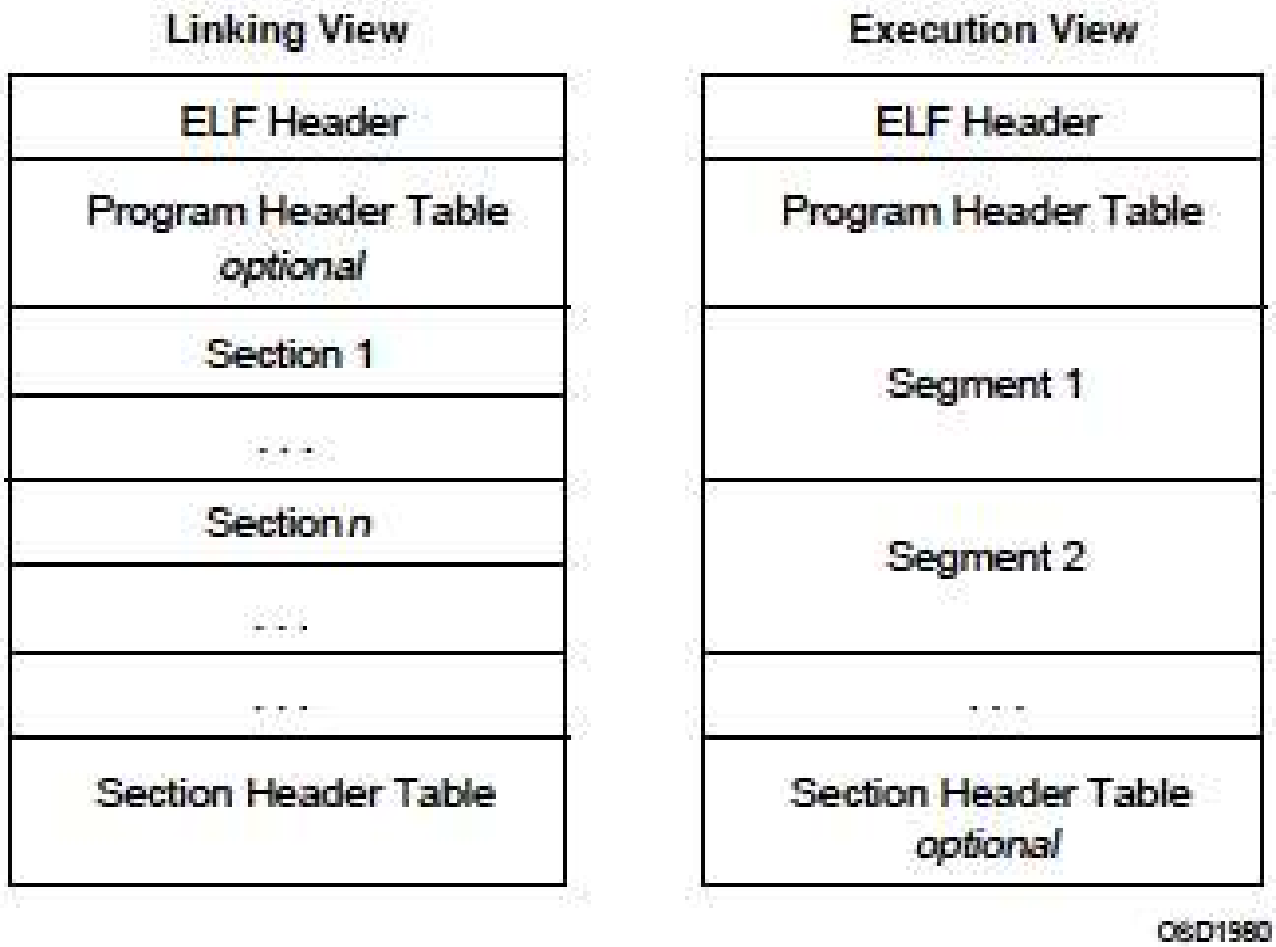
- とりあえず、静的な解析にはじゅうぶん対抗できるしくみを目指す
  - 攻撃者を油断させるようなもの、手抜きをすると解析に失敗するようなものにする
- 2 つのアプローチを組み合わせて用いる
  - ELF header の詐称
    - instruction 領域とデータ領域の境界をあいまいにする
  - Self-generating code[5]
    - instruction コードを動的に生成する

ELF Format(1)

- Executable and Linkable Format[4]
- 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

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

Self-Generating Code

- 実行中に新たなコードを作成するコード
  - cf. self-modifying code
    - 実行中に自分自身を書き換えるコード
  - JIT コンパイラやバッチプログラムなどで行われたりする
- コードの一部を実行中に生成することにすれば、静的な解析は失敗する [5]
- コード領域は通常 read-only なので、前述の ELF header 詐称と連携することが必要
  - Writable な領域の一部をコード領域として使用する
  - コード生成の操作を目立ちにくいものにする

Indirect jump の推奨

- ラベルを直接参照するジャンプはプログラムの理解を容易にする [6]
  - フローの流れを静的に追うことができる
- プログラム中の direct jump はすべて indirect jump に変換する

```
jmp label

      ↓

set label, %r
jmp %r
```

実装の概要

- アーキテクチャ :Sparc
- ターゲット言語 :gcc が C 言語をコンパイルして吐くアセンブリ
- 一部バイナリをいじることになる
  - アセンブリレベルだとラベルがアドレスにまだ解決されていないので、得られる情報としては不十分

実行手順 (1)

1) ソースコードをコンパイルし、アセンブリを得る
2) フロー解析、生存変数解析などを行う
3) Direct jump を indirect jump に変換する
4) 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
5) tamperproof のコードを埋め込む
   - 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

実行手順 (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
- コードを配置する場所にはダミーのコードを置いておく [5]

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

- 単純に静的な解析を行うと失敗する
  - Indirect jump
  - Self-generating code
- 実行コードは通常のものと同じであるように見せかける
  - エントリポイントから direct jump を頼りに追えるフローが、header の示す instruction 領域内で完結するようにする
  - Self-modifying code を発動させる前のコードは、正しくないフローを持つようにする

現状と予定

- 実装中
  - アセンブリの操作
  - バイナリの操作
  - ELF header の操作
- 今年中に実装と評価を終わらせたい

References(1)

- [1] H. Chang and M. Atallah. Protecting software code by guards. In Proc. 1st ACM Workshop on Digital Rights Management (DRM 2001), pages 160-175. Springer LNCS 2320, 2002.
- [2] C. Linn and S. Debray. Obfuscation of Executable Code to Improve Resistance to Static Disassembly. In Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS), pages 290-299, October 2003.

References(1)

- [3] Bill Horne, Lesley R. Matheson, Casey Sheehan, and Robert Endre Tarjan. Dynamic self-checking techniques for improved tamper resistance. In the ACM CCS-8 Workshop on Security and Privacy in Digital Rights Management, Lecture Notes in Computer Science 2320, pages 141–159, 2001.

Reference

- [4]TIS Commitee. Tool Interface Standard, Executable and Linking Format Specification, Version 1.2, May 1995.
- [5] 神崎雄一郎，門田暁人，中村匡秀，松本健一，" 命令コードの実行時置き換えによるプログラムの解析防止，" 電子情報通信学会技術報告，情報セキュリティ研究会，Vol. ISEC2002-98, pp.13-19, Dec. 2002.

References(2)

- [6] Christian Collberg, Edward Carter, Saumya Debray, Andrew Huntwork, Cullen Linn, and Mike Stepp. Dynamic path-based software watermarking. In SIGPLAN '04 Conference on Programming Language Design and Implementation, june 2004.