Effective Software Obfuscation by Mixing Instructions and Data

米澤研究室
学部４年　31010
佐藤秀明

概要

- 命令とデータの区別を難しくする難読化手法の研究

難読化とは

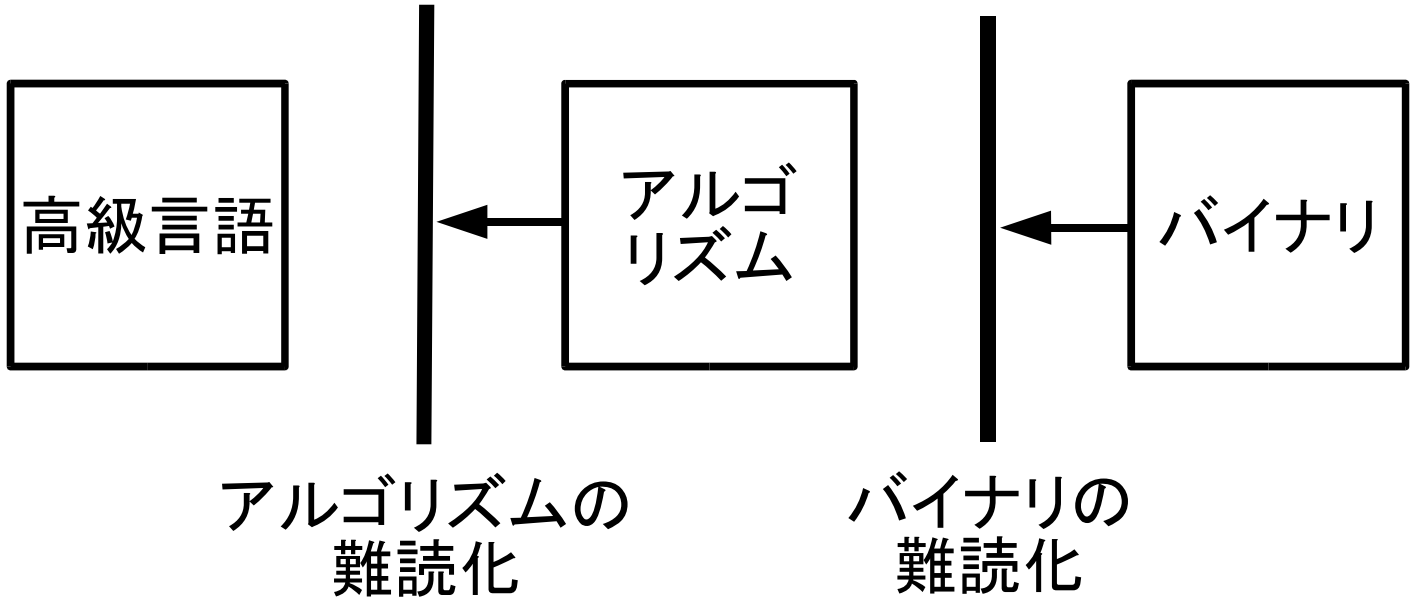
- プログラムを理解しにくいものに変換すること
  - 元のセマンティクスはそのまま保存

プログラムを難読化する理由

- リバースエンジニアリングの防止
  - 知的財産を保護する
  - 脆弱性を攻撃者が発見しにくくする

難読化技術の分類

- バイナリの難読化
  - アルゴリズムの正しい抽出を難しくする技術
  - 既存研究は少ない
- アルゴリズムの難読化
  - 可読性の高い高級言語の復元を難しくする技術
  - 既存研究は多数



バイナリ難読化の既存研究 (1)

- x86 上でのバイナリ難読化 [Linn et al., 2003]
  - 命令間に junk byte を挿入
    - 誤った命令を逆アセンブラに抽出させる
  - jump 命令を特殊な関数への呼び出しに置換
    - 制御の追跡を困難にする
- 問題
  - 固定長命令の環境では junk byte 法が使えない
  - 彼らの難読化手法の弱点が指摘されている [Kruegel et al., 2004]

バイナリ難読化の既存研究 (2)

- 動的コード生成を用いたバイナリ難読化 [神崎ら、2004]
  - 実行前と実行後はダミー命令で本来の命令を偽装
  - 実行時のみ本来の命令が出現

我々の目的

- 新たなバイナリ難読化手法の提案
  - 様々なアーキテクチャに適用可能
    - cf. x86 依存 [Linn et al., 2003]
  - 小さなオーバーヘッドで高い難読化効果

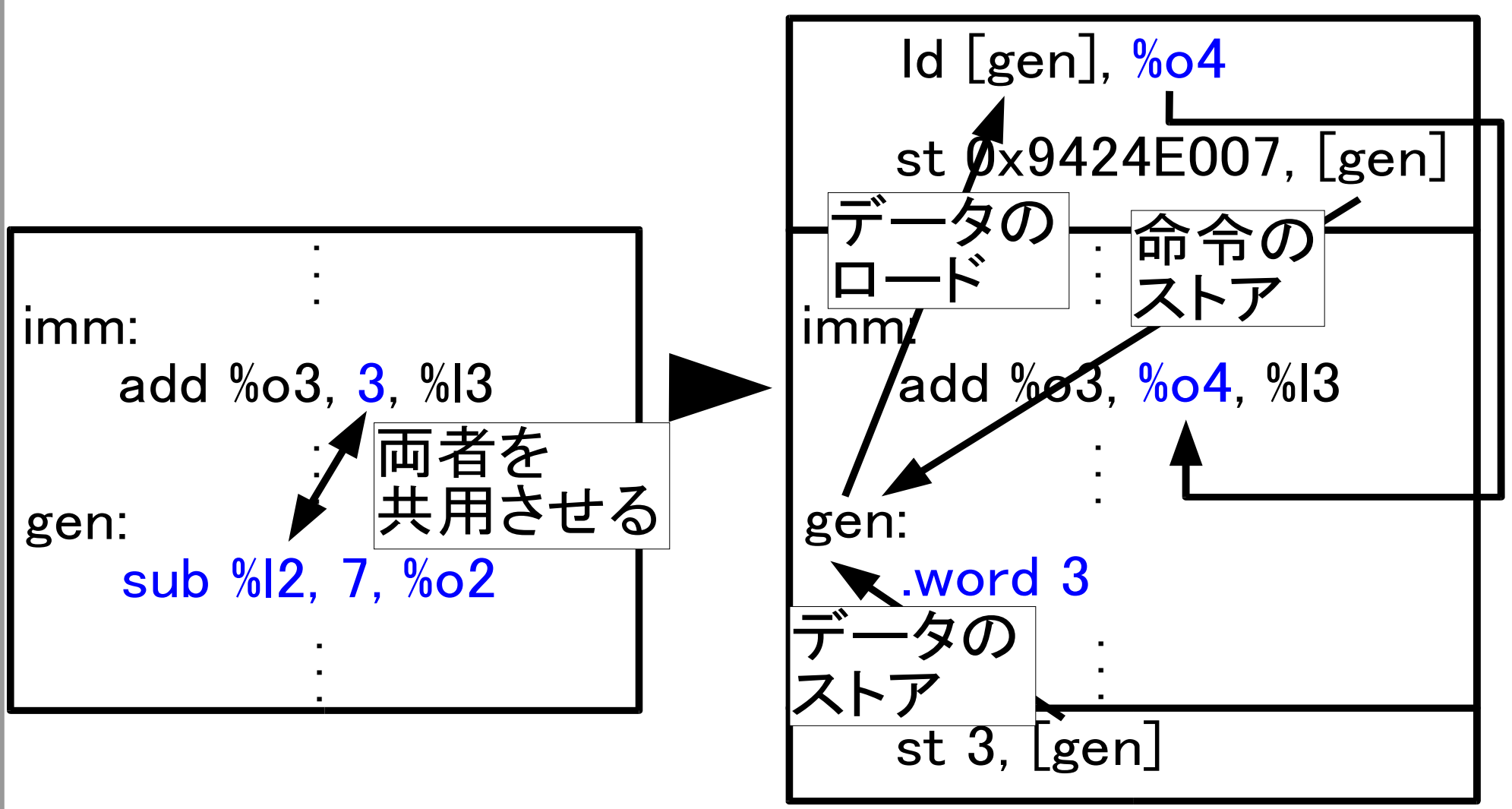
我々のアプローチ

- 命令とデータの区別を難しくするしくみを導入
  - 本来の命令を実際に使用されるデータで隠蔽
    - cf. 本来の命令をダミー命令で偽装 [神崎ら、 2004]
  - 動的生成された命令をデータだと誤解させる

コード難読化の例

難読化前のコード

難読化後のコード



我々の手法の利点

- 静的なリバースエンジニアリングでは検知不可能
  - cf. 動的リバースエンジニアリング…コスト大

実装

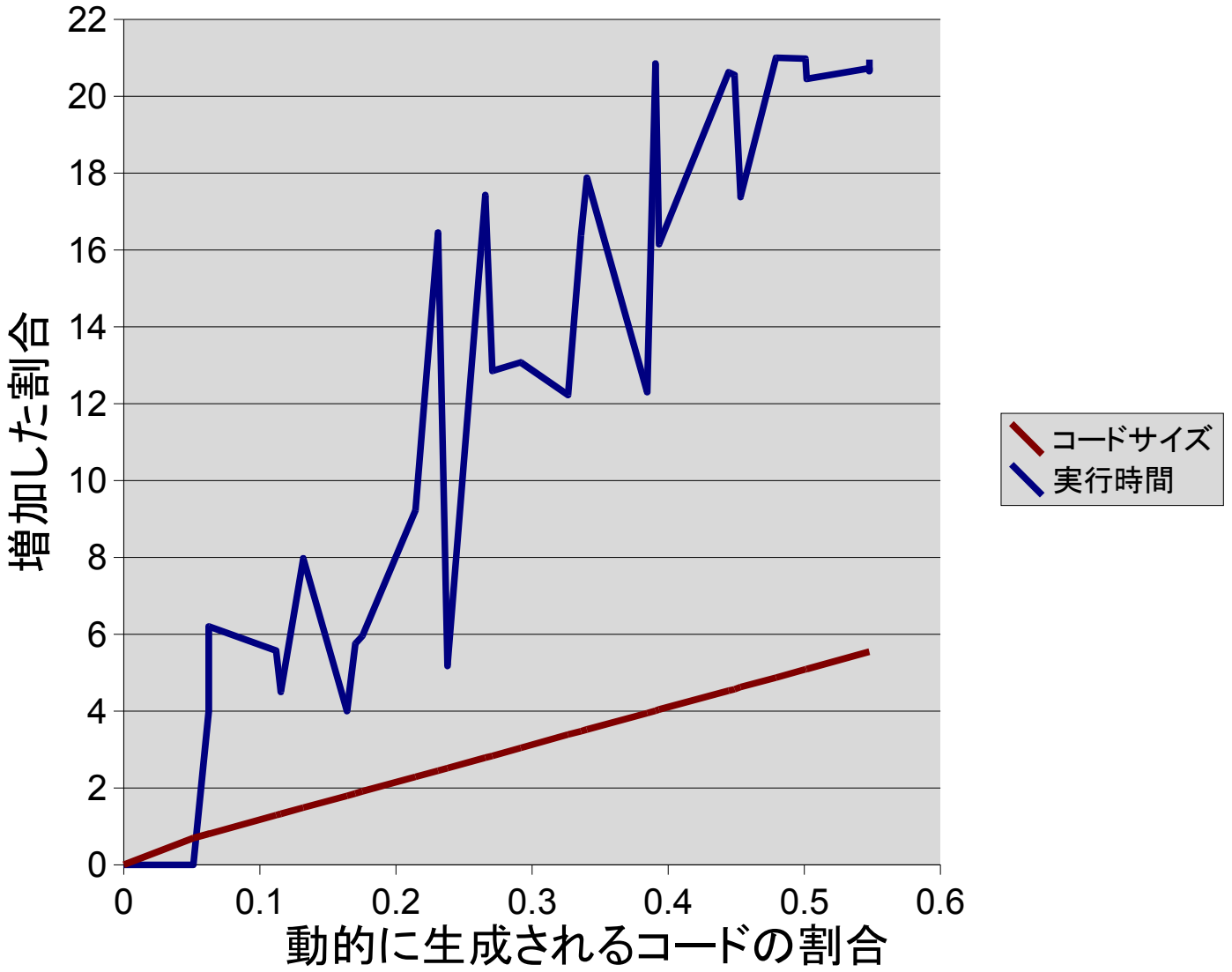
- 環境 :SPARC
- アセンブリ to アセンブリの変換器として実装

評価 (1)

- 逆コンパイラは逆アセンブルの途中で異常終了
  - 逆コンパイラ :boomerang
  - 動的コード生成に対処できない

評価 (2)

- コードサイズと実行時間の測定 (プログラムは LU 分解)



考察

- 同じ動的生成率に対する実行時間の分散が大
  - 繰り返し動的に生成される命令が実行時間に影響大
- 実行時間と難読化レベルとのトレードオフ
  - 難読化の割合に対し実行時間は飛躍的に増加

今後の予定

- 詳細な評価
  - より実用的なプログラムで実験
  - 商用のリバースエンジニアリングソフトを用いた強度測定
    - [神崎ら、 2004] の手法との比較
- 手法の改良
  - 変換によって追加されるコードの最適化
  - 繰り返し実行される命令を検知して動的生成を抑止

まとめ

- 新しいバイナリ難読化の手法を提案
  - 命令とデータの区別を難しくするアプローチ
- 静的リバースエンジニアリングを防止
- 実行時間の増加を抑えることが課題