卒論ミーティング (11/11)

31010 佐藤秀明

概要

- 路線変更
  - Static analysis に耐える Tamperproofing を眼目に
- Self-generating code
- 新しいシステムの説明
- これからの予定

前回の方法の問題点(1)

- コードをチェックする関係を幾重にも重ねて強度を上げる方法は断念
  - チェックするコード同士が循環依存を起こす問題を根本的に解決するのは困難
  - 表面上は循環的なチェックを行っているかのように見せかけてごまかす方法ならある[1]

前回の方法の問題点 (2)

- 循環依存によって計算が止まらなくなる例

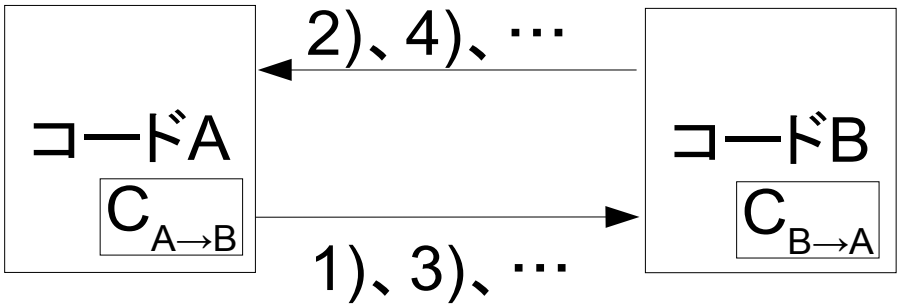
  1)B をチェックするコード $C_{A\to B}$ を A に埋め込む

  2)A をチェックするコード $C_{B\to A}$ を B に埋め込む

  3) 2) で B が変化したのに合わせて $C_{A\to B}$ を修正する

  4) 3) で A が変化したのに合わせて $C_{B\to A}$ を修正する

  5) 4) で B が…



路線変更

- とりあえず、静的な解析を用いた攻撃に耐えうる tamperproofing のしくみを作る
  - 動的な解析に対する防御法はまた今度
- Self-generating code の考え方を利用
  - チェックを行うコードを動的に生成する

静的な解析の一方法(1)

- 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

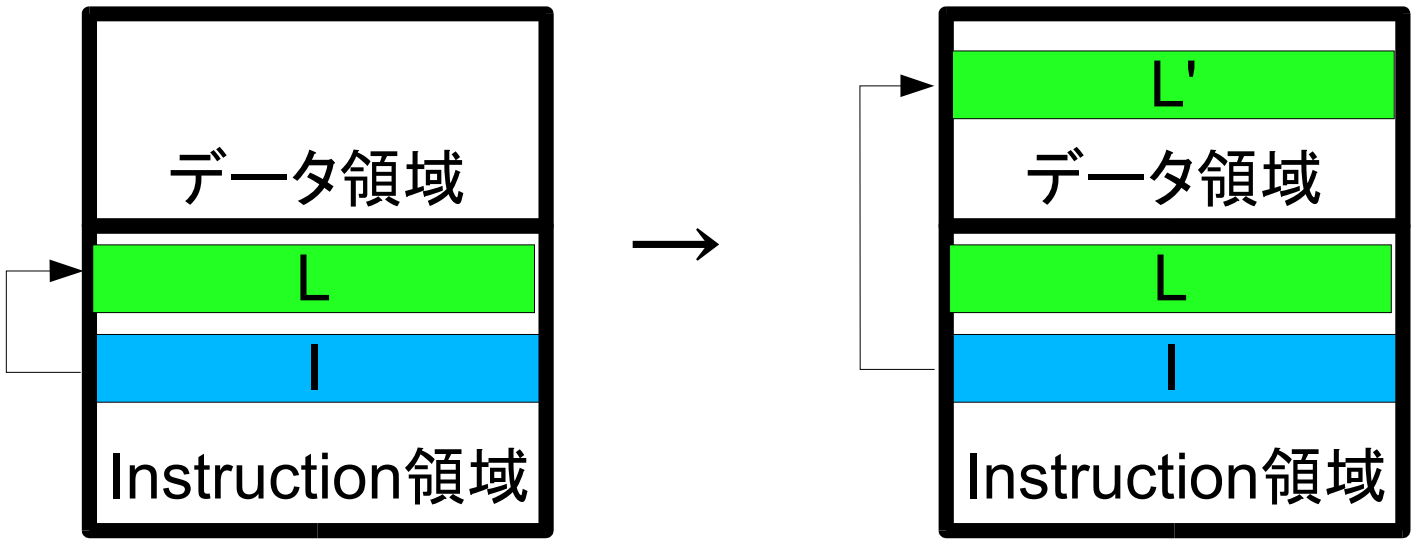
静的な解析の一方法(2)

- ある instruction I が instruction 領域中のアドレス L を直接参照している場合

  1)L の内容をデータ領域中のアドレス L' にコピーする

  2)I には L ではなく L' を参照させる

- 直接参照による Tamperproofing は比較的簡単に外せる



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
- コードの正しさをチェックするコードを実行中に生成することにすれば、静的な解析には引っかからない

Tamperproof されたプログラムの挙動

1) チェック用コードを生成する
   - 新しいコードを置く場所は、静的に確保されたデータ領域 (可能なら動的に確保した領域)
2) チェック用コードを実行する
   - チェック用コードへは branch function[2] などを用いて implicit にジャンプする
     - 静的な解析では間接ジャンプを追うことが困難 [3][4]
   - チェック用コードは単に元コードのチャックサムやハッシュ値を取り、正しい値と比較する
3) 元コードに復帰して通常の instruction を実行する

Tamperproof の詳細 (1)

- 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情報としては不十分
- 「チェック用コードを作成」するコード自体はチェックされない
  - 循環依存を避けるため
  - 目立たないようにしなければならない
    - 元コードに既にある静的・動的データ領域を共用してチェック用コードを置く
    - チェック用コードを値として利用する instruction を多数設置する
  - 幸い、チェック用コードの生成作業は単なる配列への値の代入に見える

Tamperproof の詳細 (2)

- オプションとして、同時に obfuscation[5] を施す
  - 間接ジャンプの多用
  - Calling convention のかく乱
  - 制御フローの改変
  - Opaque predicate の導入

これからの予定

- ELF file format の構造と操作方法の勉強
  - LibElf で内部を操作できるらしい
- 省サイズで効率や数学的性質のよいハッシュ関数のコードを作る

References(1)

- [1] 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.

References(2)

- [2] 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.
- [3] 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(3)

- [4] Christopher Kruegel, William Robertson, Fredrik Valeur, and Giovanni Vigna. Static Disassembly of Obfuscated Binaries. In Proceedings of the 13th USENIX Security Symposium, pages 255–270, 2004.
- [5] Christian Collberg, Clark Thomborson, and Douglas Low. A taxonomy of obfuscating transformations. Technical Report 148, The University of Auckland, New Zealand, 1997.