Semantics-Based Obfuscation-Resilient Binary Code Similarity Comparison with Applications to Software Plagiarism Detection

FSE 2014
Lannan Luo, Jiang Ming, Dinghao Wu, Peng Liu, Sencun Zhu

김영철

2016. 9. 7.

INTRODUCTION

- 코드 난독화 기술에 의해 코드 탐지 기술 적용 불가
  - clone detection
  - binary similarity detection
  - software plagiarism detection

INTRODUCTION

- CoP, obfuscation-resilient method 소개
  - longest common subsequence(LCS)
  - basic block, path, whole program 세 단계로 semantics 모델링
    - basic block level – symbolic execution 이용
    - path level – LCS 를 이용하여 path similarity 계산

OVERVIEW

Methodology

- 프로그램은 다양한 레벨로 모델링이 가능
  - syntax 기반 – 같은 의미를 다양하게 표현이 가능
  - system call graph 기반 – 시스템 콜이 다른 시스템 콜로 대체 가능

- 프로그램 semantics의 정형화
  - equivalence relation 대신에 similarity를 이용
  - 두 비교 대상이 같은 정형화된 semantics를 갖는다면 같다고 봄
  - 두 논리적 표현의 similarity를 판단하는 것은 어려움

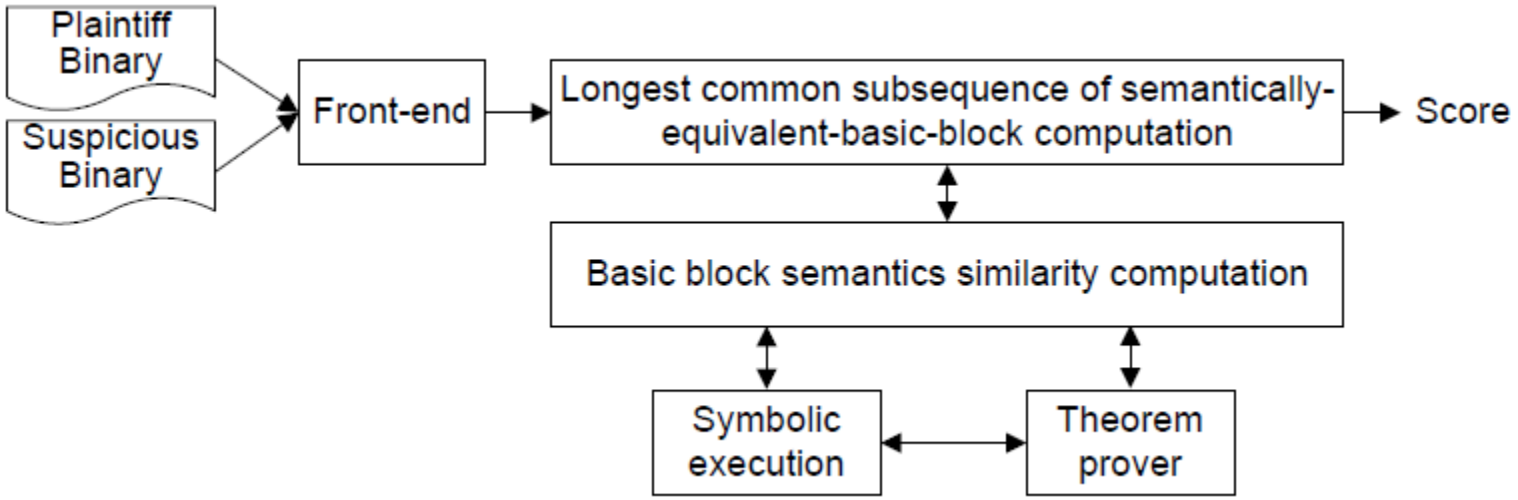
OVERVIEW

Methodology

- Similarity를 판단하기 위한 방법
  - 바이너리 코드 베이직 블록 단계에서 semantics를 모델링
    - semantics equivalence + similarity semantically
  - LCS 를 이용한 path 비교

OVERVIEW

Architecture



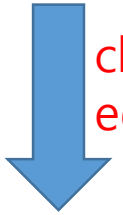
BLOCK SIMILARITY COMPARISION

Strictly Semantic Equivalence

```
p = a+b;  |  s = x+y;
q = a-b;  |  t = x-y;
```

symbolic execution

$$p = f_1(a,b) = a + b \qquad s = f_3(x,y) = x + y$$
$$q = f_2(a,b) = a - b \qquad t = f_4(x,y) = x - y$$



checking equivalence

$$a = x \wedge b = y \implies p = s$$

q와 t의 관계도 비슷

BLOCK SIMILARITY COMPARISION

Strictly Semantic Equivalence

- 두 코드 세그먼트의 input과 output의 수가 같아야 함
- 두 세그먼트 사이의 output formula를 같은 쌍으로 만드는 순열이 존재함. (????)

BLOCK SIMILARITY COMPARISION

Semantic Equivalence

- strictly semantic equivalence 대신에 원본 블록의 output variable을 대상 블록의 output과 대응하는지 체크함.

BLOCK SIMILARITY COMPARISION

Formalization

- define a pair-wise equivalence formula of input variable

- define a output equivalence formula

PATH SIMILARITY COMPARISON

Starting Blocks

- 원본 프로그램과 대상 프로그램에서 시작 지점을 어떻게 찾아낼 지 제시
  - routine code를 피하기 위해 처음 브랜치 하는 베이직 블록을 선택
  - 대상 프로그램에서 선택한 베이직 블록과 의미적으로 같은 것을 탐색
  - 여러 개가 발견되면 LCS를 계산하여 선택

PATH SIMILARITY COMPARISON

Linearly Independent Paths

- 원본 프로그램으로부터 linearly independent path set 선택
  - 각각의 loop는 한 번 unroll하고 DFS로 path set을 찾음
- 원본 프로그램의 starting block과 대상 프로그램의 starting block 후보군을 식별하고 path 추적
- LCS를 수행하여 path embedding score 계산

PATH SIMILARITY COMPARISON

Longest Common Subsequences of Semantically Equivalent Basic Blocks

- highest LCS score를 찾는 과정은 NP-complete
- back edge가 없는 directed acyclic graphs로 표현
- graph의 weight에 보수를 취하여 shortest path problem으로 변환
- BFS와 LCS를 결합한 방법 선택

PATH SIMILARITY COMPARISON

Longest Common Subsequences of Semantically Equivalent Basic Blocks

```
1: function PathSimilarityComparison(P,G,s)
2:     enq(s,Q) // Insert s into queue Q
3:     Initialize the LCS table δ
4:     Initialize the σ array to all zero
5:     r ← 0 // set the current row of table δ
6:     while Q is not empty do
7:         currNode ← deq(Q)
8:         for each neighbor u of currNode do
9:             LCS(u,P)
10:        end for
11:    end while
12:    ħ = max_{i=0}^{r}(δ(i,n)) // get the the highest score
13:    if ħ > θ then // higher than the threshold
14:        RefineLCS()
15:        ħ = max_{i=0}^{r}(δ(i,n))
16:    end if
17:    return ħ
18: end function
```

```
19: function LCS(u,P)
20:    δ(u,0) = 0
21:    for each node v of P do
22:        if SEBB(u,v) then // semantically eq. blocks
23:            δ(u,v) = δ(parent(u), parent(v)) + 1
24:            γ(u,v) = ↖
25:            if σ(u) < δ(r,v) then
26:                r++
27:            end if
28:        else
29:            δ(u,v) = max(δ(parent(u),v), δ(u, parent(v)))
30:            γ(u,v) = ← or ↑
31:        end if
32:        if σ(u) < δ(r,v) then
33:            σ(u) = δ(r,v)
34:            enq(u,Q)
35:        end if
36:    end for
37: end function
```

PATH SIMILARITY COMPARISON

Longest Common Subsequences of Semantically Equivalent Basic Blocks

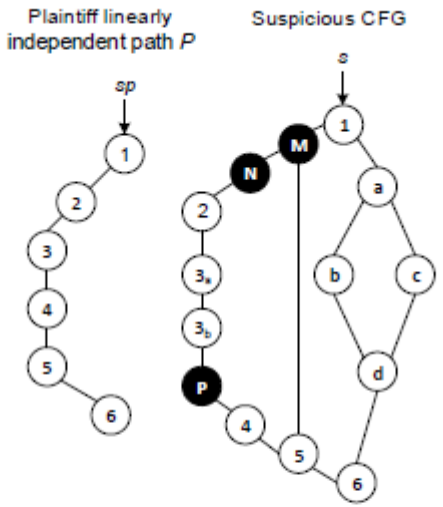


Figure 3: An example for path similarity calculation. The black blocks are inserted bogus blocks. There is an opaque predicate inserted in M that always evaluates to true at runtime which makes the direct flow to the node 5 infeasible.

	v	1	2	3	4	5	6
u	0	0	0	0	0	0	0
1	0	↖1	←1	←1	←1	←1	←1
5	0	↑1	↑1	↑1	↑1	↖2	←2
2	0	↑1	↖2	←2	←2	←2	←2
6	0	↑1	↑1	↑1	↑1	↑2	↖3
4	0	↑1	↑2	↑2	↖3	←3	←3
5	0	↑1	↑2	↑2	↑3	↖4	←4
6	0	↑1	↑2	↑2	↑3	↑4	↖5

Figure 4: The δ and γ tables store the intermediate LCS scores and the directions of the computed LCS, respectively. The three arrows on the left indicate the parent-child relationship between two nodes in the suspicious program during the LCS computation. For example, in the computed LCS, the parent node of node 2 is node 1, instead of node 5.

PATH SIMILARITY COMPARISON

Longest Common Subsequences of Semantically Equivalent Basic Blocks

- how to deal with opaque predicate insertion
  - node M의 경우
    - path exploration이 모든 분기를 고려하기 때문에 해결할 필요 없음
- how to deal with obfuscated
  - P의 node 3과 G의 3a, 3b의 경우
    - SSEB()에서 다르다고 판정
    - 이런 경우를 처리하기 위해 LCS refinement 를 개발

PATH SIMILARITY COMPARISON

Refinement

- Conditional obfuscation
  - flag(CF, ZF …)와 같은 conditionals이 난독화 될 수 있음
  - 블록 병합을 통해 처리
    - 난독화 된 블록들을 합쳐 similarity를 detect할 수 있음
- Basic block splitting and merging
  - LCS는 block 분할과 병합에 대한 처리가 불가능
    - LCS refinement
    - CoP는 역추적을 통해 대상 프로그램의 semantically equivalent 하지 않는 연속된 블록 시퀀스를 찾아 하나의 코드 덩어리로 만듦