

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. 23.

INTRODUCTION

- 기존의 방법으로는 코드 난독화 기술에 의해 코드 탐지 기술 적용 불가
 - clone detection
 - binary similarity detection
 - software plagiarism detection
- 난독화에도 대처 가능한 CoP라는 방법 제안

INTRODUCTION

- CoP, obfuscation-resilient method 소개
 - longest common subsequence(LCS)를 기초로 하여 similarity를 계산
 - basic block, path, whole program 세 단계로 semantics 모델링
 - basic block level – symbolic execution을 이용하여 input-output 관계를 표현하는 symbolic formula를 얻음
symbolic formula를 비교하여 pair-wise equivalence를 확인하고 원본 블록의 결과값 변수들과 같다고 판정된 개수의 percentage를 계산하고 임계값을 두어 난독화에 어느정도 대처할 수 있도록 함
 - path level – 두 path에 대해 LCS를 수행하여 의미적으로 같은 블록의 수를 계산

OVERVIEW

Methodology

- 프로그램 semantics의 정형화
 - 프로그램은 다양한 형태로 모델링이 가능하지만 난독화에 취약
 - syntax 기반 – 같은 의미를 다양하게 표현이 가능
 - system call graph 기반 – 시스템 콜이 다른 시스템 콜로 대체 가능
 - equivalence relation 대신에 similarity를 이용
 - 두 비교 대상이 같은 정형화된 semantics를 갖는다면 같다고 봄
 - 두 논리적 표현의 similarity를 판단하는 방법 제시

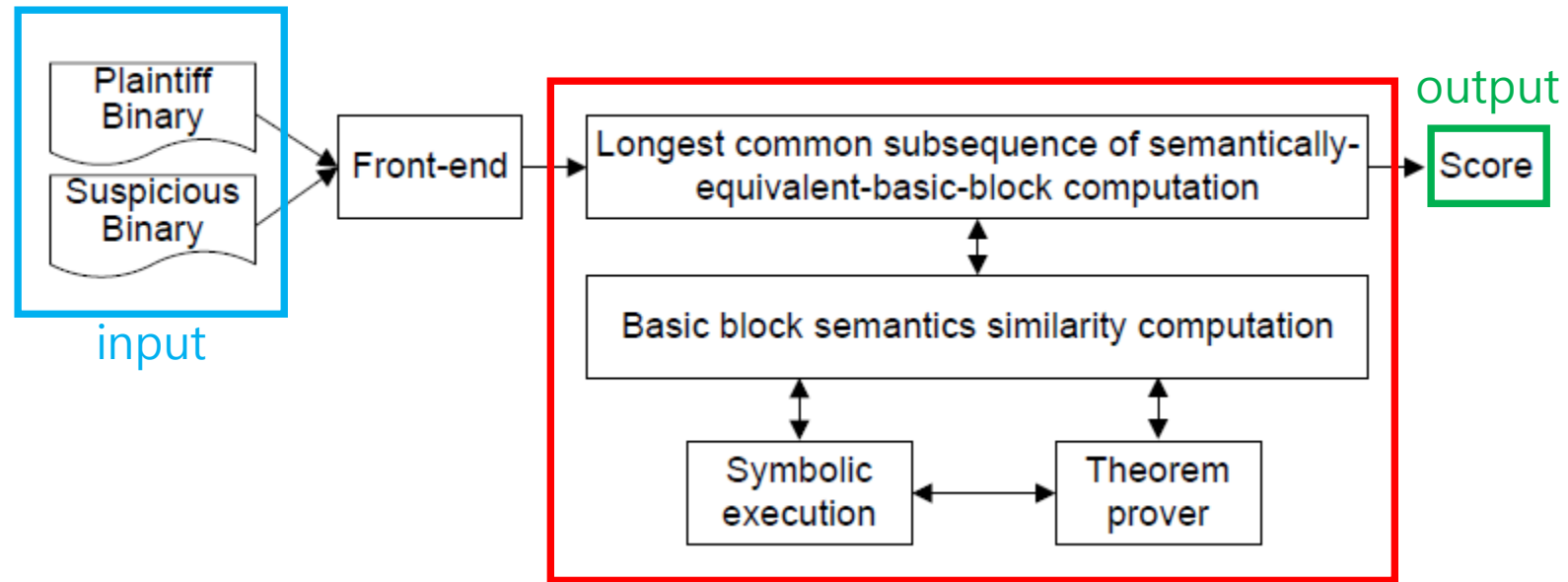
OVERVIEW

Methodology

- Similarity를 판단하기 위한 방법
 - symbolic execution을 이용하여 basic block간의 의미적 동등함 비교
 - LCS를 이용하여 의미적으로 같은 basic block의 수를 세고 path similarity 계산

OVERVIEW

Architecture



BLOCK SIMILARITY COMPARISON

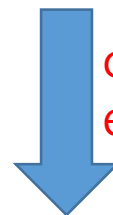
Strictly Semantic Equivalence

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

symbolic
execution


$$\begin{array}{l} p = f_1(a, b) = a + b \\ q = f_2(a, b) = a - b \end{array}$$
$$\begin{array}{l} s = f_3(x, y) = x + y \\ t = f_4(x, y) = x - y \end{array}$$

checking
equivalence


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

q와 t의 관계도 비슷하게 도출

BLOCK SIMILARITY COMPARISON

Strictly Semantic Equivalence

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

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



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

BLOCK SIMILARITY COMPARISON

Semantic Equivalence

- 원본 블록의 output variable을 대상 블록의 output과 대응하는지 체크함.

	$u = x+10;$
$p = a+b;$	$v = y-10;$
$q = a-b;$	$s = u+v;$
	$t = x-y;$
	$r = x+1;$



$$\begin{array}{l} a = x \wedge b = y \implies p = s \\ a = x \wedge b = y \implies q = t \end{array}$$

BLOCK SIMILARITY COMPARISON

Formalization

- define a pair-wise equivalence formula of input variable

$$p(X, \pi(Y)) = \bigwedge_{i=0}^n (X_i = \pi_i(Y))$$

- define a output equivalence formula

$$\begin{aligned} \forall y_1 \in Y_1. \exists y_2 \in Y_2, p(X_1, \pi(X_2)). \\ p(X_1, \pi(X_2)) \implies f_1(X_1) = f_2(X_2). \end{aligned}$$

PATH SIMILARITY COMPARISON

Starting Blocks

- 원본 프로그램과 대상 프로그램에서 시작 지점을 어떻게 찾아낼 지 제시
 - 브랜치로 끝나는 첫 번째 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  $\delta$ 
4:   Initialize the  $\sigma$  array to all zero
5:    $r \leftarrow 0$  // set the current row of table  $\delta$ 
6:   while Q is not empty do .....
7:     currNode  $\leftarrow$  deq(Q)
8:     for each neighbor u of currNode do
9:       LCS(u,P)
10:    end for
11:  end while .....
12:   $\hat{h} = \max_{i=0}^r(\delta(i,n))$  // get the the highest score
13:  if  $\hat{h} > \theta$  then // higher than the threshold
14:    RefineLCS()
15:     $\hat{h} = \max_{i=0}^r(\delta(i,n))$ 
16:  end if
17:  return  $\hat{h}$ 
18: end function
```

P : 원본 프로그램의 path
G : 대상 프로그램의 path
s : 대상 path의 starting block

모든 블록에 대해 처리한 후, 반복 종료
LCS() 함수 내부에서 처리해야 할 블록을 큐에 넣는 작업을 함

PATH SIMILARITY COMPARISON

Longest Common Subsequences of Semantically Equivalent Basic Blocks

```
19: function LCS(u,P)
20:    $\delta(u, 0) = 0$ 
21:   for each node v of P do
22:     if SEBB(u,v) then // semantically eq. blocks
23:        $\delta(u, v) = \delta(\text{parent}(u), \text{parent}(v)) + 1$ 
24:        $\gamma(u, v) = \nwarrow$ 
25:       if  $\sigma(u) < \delta(r, v)$  then
26:         r++
27:       end if
28:     else
29:        $\delta(u, v) = \max(\delta(\text{parent}(u), v), \delta(u, \text{parent}(v)))$ 
30:        $\gamma(u, v) = \leftarrow$  or  $\uparrow$ 
31:     end if
32:     if  $\sigma(u) < \delta(r, v)$  then
33:        $\sigma(u) = \delta(r, v)$ 
34:       enq(u,Q)
35:     end if
36:   end for
37: end function
```

u : 대상 path의 current block
P : 원본 프로그램의 path

첫 번째 LCS

line 22 P의 1과 G의 1이 의미적으로 같음
line 26 오리지널 값보다 증가했기 때문에 새로운 행(2행) 생성
line 29-30 for 반복문을 통해 2행의 값들을 업데이트
line 33-34 node 1을 큐에 삽입

두 번째 LCS

line 22 node 1과 인접한 node M과 node a가 원본 path에서
의미적으로 같은 것이 존재하지 않음
line 25 새로운 행 생성하지 않음

PATH SIMILARITY COMPARISON

Longest Common Subsequences of Semantically Equivalent Basic Blocks

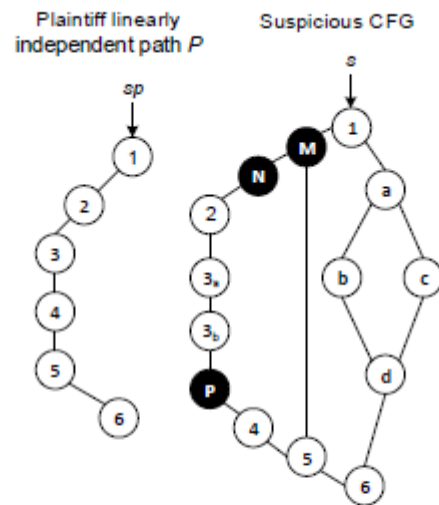


Figure 3: An example for path similarity calculation. The black blocks are inserted bogus blocks. There is an opaque predicate inserted in M that always evaluates to true at runtime which makes the direct flow to the node 5 infeasible.

	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

IMPLEMENTATION & EVALUATION

Experimental Settings

- MOSS, JPLag, Bdiff, DarunGrim2와 비교
- MOSS, JPLag
 - source code based, syntax-based
- Bdiff
 - binary code based, text를 위한 diff 유틸리티와 유사
- DarunGrim2
 - binary code based, 최신 패치 분석 도구
- 실험은 Core2 Duo CPU, 4GB RAM, Linux machine에서 진행
- 블록 유사도의 임계값은 0.7로 설정
- 선택된 path가 원본 path를 80% 이상 cover하는 것만 탐지

IMPLEMENTATION & EVALUATION

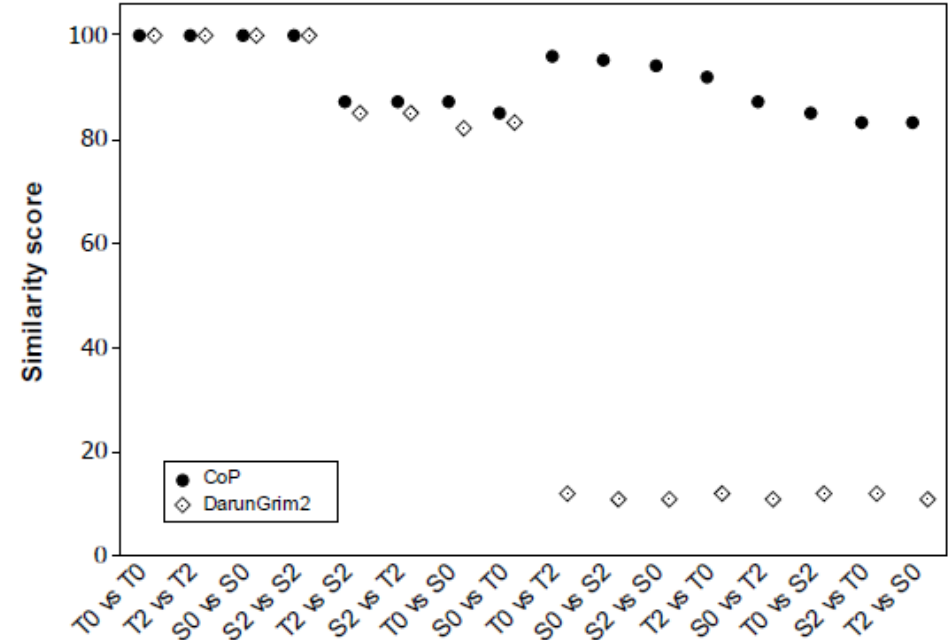
Thttpd

thttpd-2.25b, shttpd-2.26.4를 이용하여 실행파일 생성

- 컴파일러 최적화 옵션에 따른 변화에 대한 Resilience
 - 다른 최적화 옵션을 비교할 때 차이남
 - 다른 레지스터를 할당할 때,
 - 명령어를 대체할 때,
 - DarunGrim2에서 성능 급격히 저하

변화를 유발하는 요소

- 다른 레지스터 할당
- 명령어 대체
- 블록 splitting & combination
- function inline & outline



IMPLEMENTATION & EVALUATION

Thttpd

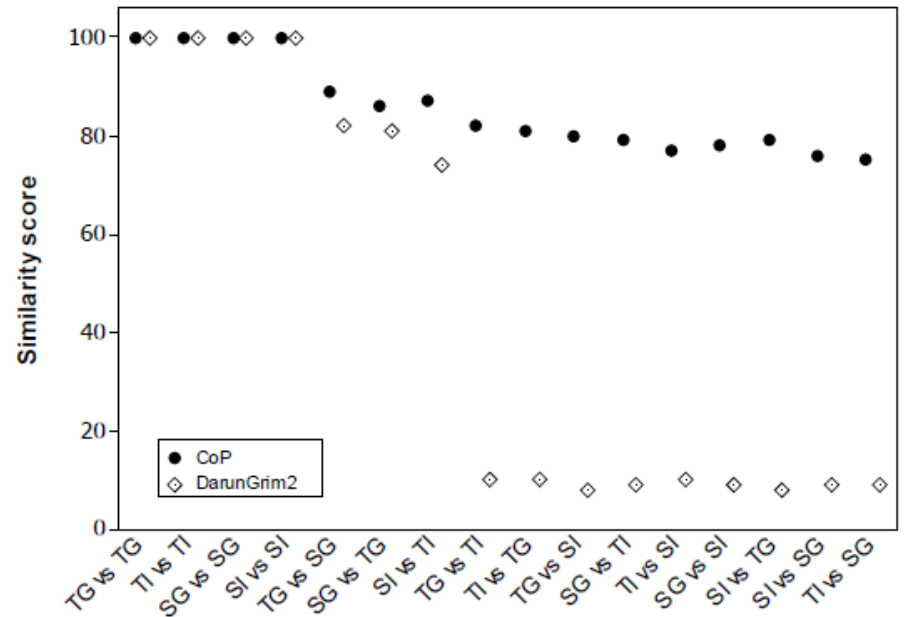
- 컴파일러 최적화 옵션에 따른 변화에 대한 Resilience
 - thttpd-2.25b 의 경우 최적화 옵션에 따라 블록 분리 & 결합
 - -O2 옵션의 경우, cmovs를 이용하여 블록 하나로 결합
 - -O0 옵션의 경우, 두 개의 블록으로 분리

IMPLEMENTATION & EVALUATION

Thttpd

thttpd-2.25b, shttpd-2.26.4를 GCC, ICC -O2 컴파일

- 컴파일러에 따른 변화에 대한 Resilience
 - compilation, optimization 알고리즘 사용하는 C library에 따른 차이 발생
 - DarunGrim2 보다 CoP의 성능이 더 좋음



IMPLEMENTATION & EVALUATION

Tthttpd

- 난독화에 대한 Resilience

Obfuscation		Similarity score (%)				
Category	Transformation	Source code based		Binary code based		
		MOSS	JPLag	DarunGrim2	Bdiff	CoP
Layout	Remove comments, space, and tabs	47	62	100	100	100
	Replace symbol names, number, and strings	22	90	100	100	100
Control	Insert opaque predicates	—	—	47	43	95
	Inline method	—	—	32	34	91
	Outline method	—	—	38	33	90
	Interleave method	45	40	32	19	89
	Convert multiple returns to one return	75	91	98	86	97
	Control-flow flattening	—	—	5	3	86
	Swap <i>if/else</i> bodies	72	78	81	73	98
	Change <i>switch/case</i> to <i>if/else</i>	74	51	69	51	94
Data	Replace logical operators (&&, ?:, etc.) with <i>if/else</i>	79	95	97	88	96
	Split structure object	83	87	93	82	100
	Insert bogus variables	93	88	86	75	100


IMPLEMENTATION & EVALUATION

Openssl

openssl-1.0.1f의 MD5를 gcc -O2로 컴파일

비교대상 : openssh-6.5p1, cyrus-sasl-2.1.26, libgcrypt-1.6.1, acl-2.2.52, attr-2.4.47

비교대상	openssl-1.0.1f base 유무	similarity score
openssh-6.5p1	O	87%~100%
libgcrypt-1.6.1	O	12%~30%
cyrus-sasl-2.1.26	O	12%~30%
acl-2.2.52	X	2% 이하
attr-2.4.47	X	2% 이하

 md5 re-implement

 completely irrelevant