bit-Level Taint Analysis

SCAM 2014

Babak Yadegari, Saumya Debray

김영철

2019. 5. 3.

Introduction

- 동적 테인트 분석
  - 프로그램 실행 중 데이터 흐름을 추적하고 관심 있는 특정 데이터 근원에서 얻을 수 있는 모든 데이터를 마킹
  - 테인트를 유지하고 전파하는 것이 중요 포인트

  - 테인트 분석 기법의 무마
    (1) 공격자는 리턴 주소나 함수 포인터같은 콘텐츠를 제어하려고 함.
      - 하지만 under-tainting이나 false-negative가 많이 발생함.
    (2) 공격자는 난독화를 적용해서 코드의 기능을 숨기려고함.
      - 하지만 over-tainting이나 false positive를 유발함.
  - 논문은 (2)에 집중함.

Introduction

- two main contributions
  - over-tainting의 이유를 논의
  - over-tainting을 다루는 enhanced bit-level 테인트 분석을 기술함

- 두 가지 방법으로 테인트 분석 알고리즘을 확장해서 over-taint로 발생하는 문제를 다룸.
  - 다양한 오퍼레이션들의 테인트 효과를 모델링하는 매핑 함수를 가진 fine-grained bit-level analysis 사용
  - 서로 다른 테인트 소스를 구별하고 추적함

Background and Motivation

- two kinds of imprecision in taint analysis
  - over-tainting : 분석에 의해 테인트로 식별된 데이터가 실제로 테인트 소스의 영향을 받지 않을 때
    ➔ 과도한 탐지로 혼란을 줄 수 있음.
  - under-tainting : 테인트 소스에 영향을 받는 데이터가 분석에 의해 테인트되지 않은 것으로 확인될 때
    ➔ 크리티컬한 문제를 탐지할 때 좋지 않음.

- 테인트 분석의 정확성을 향상시키는 방법에 대한 연구가 부족
- 이와 관련된 공격 문제에 대한 연구는 있었지만, 해결하려는 노력이나 향상시키려는 연구는 하지 않았음.

Background and Motivation

```
1    int a, b, c
2    a = read()
3    b = ~a
4    c = (a & b)
5    if (c == 0){
6        // True
7    } else {
8        // False
9    }
```

```
1    char a, b, w1, w2
2    char odd_bits = 01010101b
3    char even_bits = 10101010b
4    a = read()
5    b = 10
6    w1 = (a ∧ even_bits)
          ∨ (b ∧ odd_bits)
7    w2 = (a ∧ odd_bits)
          ∨ (b ∧ even_bits)
          ...
i    a = (w1 ∧ even_bits)
          ∨ (w2 ∧ odd_bits)
i+1  b = (w1 ∧ odd_bits)
          ∨ (w2 ∧ even_bits)
i+2  print(b)
```

```
1    edx = flags
2    eax = 0x6b30f626
3    edx = edx ∨ 0xfffff73e
4    eax = edx ∧ eax
5    edx = eax
6    eax = rotate_eight(eax, 0x10)
7    dx = rotate_left(dx, 0x3)
8    edx = edx + 0x6c7caf05
9    edx = edx ⊕ 0xe0395c49
10   ax = ax ⊕ dx
11   edx = eax
          ...
i    if((eax ∧ 0xe0000000) == 1){
i+1      True branch
i+2  } else {
i+3      False branch
i+4  }
```

<over-taint 유발 예제들>

Approach

- Overview
  - taint analyses consist of three main components
    - taint markings
      - 테인트 되었는지 아닌지는 1비트로도 표현 가능
      - 하지만 다른 테인트 소스인 경우 구별 불가
        - split variable 기법이 적용되면 마킹할 수 없게됨.
      - 컨디션 코드 플래그에 대해서만 테인트 소스 추적을 함.
      - mapping function은 테인트 마킹 전파 정책을 정의함.

Approach

- Overview
  - taint analyses consist of three main components
    - mapping functions
      - 일반적인 테인트 분석에서는 source operand가 테이트 되었으면 destination operand를 항상 테인트로 마킹함.
        ➔ over-taint 유발, granularity를 바꾸는 것은 도움이 안됨.
      - 각 operation에 따라 테인트 마킹 방법을 다르게 해야함.
    - granularity
      - 설정한 granularity level에 따라 적절한 mapping function을 설계 해야함.

Algorithm

1. Identifying taint sources
   - 어떤 값이든 테인트 소스가 될 수 있음.
   - 현재 구현에서는 모든 시스템 콜의 결과를 테인트 소스로 고려함.

2. Taint markings
   - 알고리즘 초기에 마킹을 정의하고 시작함.
   - 테인트 소스의 모든 비트는 서로 다른 마킹으로 표시될 수 있음. (?)
   - 테인트 소스의 크기에 따라 다른 크기의 마킹 사이즈가 필요함.
     (제어 플래그 비트의 경우, 각 플래그 당 1개의 마킹이 필요)

Algorithm

3. Mapping functions

- x86 instructions' three major categories
  1) Data handling & memory operations
     - 단순히 테인트 소스의 마킹을 destination 오퍼랜드에 매핑하는 것으로 충분함.
  2) Arithmetic & logic operations
     - 정확하게 전파하지 않으면 over-taint 유발
     - 테인트 마크에 대해 연산을 에뮬레이션 함.
  3) Control flow operations
     - 데이터 흐름과 명시적으로 연관되어있지 않기 때문에 신경쓰지 않음.

Algorithm

- Taint analysis algorithm
  - line 1: identifyTaintSources
  - line 3: initializeMarkings
  - line 7: map

Algorithm 1: Taint Analysis Algorithm

Input: an execution trace T
Result: annotated trace T'

```
1 T' ← IdentifyTaintSources(T)
2 TaintedVars ← InitializeTaintedVars(T')
3 Markings ← CreateMarkings(T')
4 s = T(0)
5 while s ← T'.NextInstruction()! = ∅ do
6 |   T' ← Annotate(Markings, s)
7 |   TaintedVars, Markings ← Map(Markings, s)
8 end
```

Algorithm

- Map function
  - case 1) Data handling & memory
  - case 2) Arithmetic & logic
    - operation에 따른 다른 map
  - case 3) Others (Control flow op)

Algorithm 2: Map function

```
1  Procedure Map (Markings, s)
3      src ← IdentifySources(s)
5      dst ← IdentifyDestinations(s)
7      switch s do
8          case s ∈ Data handling and memory operations
9              dst.markings ← src.markings
10         end
11         case s ∈ Arithmetic and logic operations
12             dst.markings =
               ArithmeticMap(s, src, src.markings)
13         end
14         otherwise
15             Continue
16         end
17     endsw
19     return
```

Algorithm

- xor operation's map function
  1) 1 taint, 1 constant
     상수과 마킹 비트에 대해 연산 작업

  2) 2 taints
     case 1) 같은 marking
        xor 결과는 0, not tainted
     case 2) 보수 관계의 marking
        xor 결과는 1, not tainted
     case 3) 서로 다른 marking
        xor 결과를 알 수 없음. 새로운 마킹, tainted

```
foreach (bit i of srcs):
  if (src1[i] is constant):
    if (src1[i] is 1):
      dst[i].t = ~src2[i].t
    else:
      dst[i].t = src2[i].t
  elif (src2[i] is constant):
    if (src2[i] is 1):
      dst[i].t = ~src1[i].t
    else:
      dst[i].t = src1[i].t
  else:
    dst[i].t =
      scr1[i].t ⊕ src2[i].t
```

Algorithm

- add operation's map function
  1) 1 taint, 1 constant

     case 1) 상수가 0 ➔ 마킹 비트 그대로

     case 2) 상수가 1

     ➔ 캐리 가능성, 다음 비트까지 마킹

  2) 2 taints (?)

     Second where both operands are tainted, then we can sum the markings of operands. For example the taint mark of adding two tainted bits with marks T_1 and T_1 would be $T_1 + T_2$. The advantage here again is if $T_2=\overline{T_1}$, then we are adding a bit with its complement so the result should be 0 and not tainted as is the case here.

```
foreach (bit i of srcs):
  if (src1[i] is constant):
    if (src1[i] is 1):
      dst[i].t = src2[i].t
      dst[i+1].t = src2[i].t
    else:
      dst[i].t = src2[i].t
  elif (src2[i] is constant):
    if (src2[i] is 1):
      dst[i].t = src1[i].t
      dst[i+1].t = src1[i].t
    else:
      dst[i].t = src1[i].t
  else:
    dst[i].t =
      scr1[i].t + src2[i].t
```

Algorithm

- Map function
  - shift operation의 경우, shift 사이즈는 operand에 의해 결정됨.
    - operand가 테인트인 경우, shift 결과는 새로운 테인트로 마킹됨.
  - not과 같은 single operation의 경우 매우 단순함.
    - source operand의 테인트 마킹에 대해 operation의 semantices를 적용하여 destination operand에 테인트 마킹을 하면 됨.

E N D