정적 분석에 기반한 난독화 기술 평가

박지순 이광근
서울대학교 프로그래밍 연구실

1 연구 방향

- 문제 정의
  - S/W의 가치에 대한 인식 변화
  - S/W 자산을 보호하기 위한 기술로 난독화 등장
  - 현재 다수의 난독화 솔루션 존재. 그러나..

  난독화 기술들을 객관적으로 비교/평가할 수 있는 잣대가 없음

> 정적 분석을 이용해서 난독화 기술을 평가할 수 있지 않을까?!

- 연구 목표
  - 정적 분석 기술을 이용해서 난독화를 벤치마킹해보자!

2 구현: C 난독화기

- C 소스 코드 난독화기 구현
  - 가장 대표적인 난독화 알고리즘 선정

1. Number Transform(NT)
   분석 결과 top을 돌려주는 id 함수 구현
   정수값을 저장하는 모든 구문에 적용

```
int a = 3;
int b = a - 1;
```

```
int num2top(int n)
{
    int t;
    scanf ("%d", &t);
    return (t + n + (t % (t / t))) - t;
}

int a = num2top(3);
int b = num2top(a - 1);
```

2. Control Flow Flattening(CFF)
   순차적인 실행 흐름을 수평적인 형태로 변경

```
int f(int x, int y) {
    int z;
    scanf("%d", &z)
    while (x > 0) {
        y = y + 2;
        x = x - 1;
    }
    z = x + y + z;
    return 0;
}
```

```
int f(int x, int y) {
    int z;
    int pc = 1;
    while (pc > 0) {
        switch (pc)
            case 1:
                scanf("%d", &z);
                pc++; break;
            case 2:
                if (x > 0) pc++; else pc = 5;
                break;
            case 3:
                y = y + 2;
                pc++; break;
            case 4:
                x = x - 1;
                pc = 2; break;
            case 5:
                z = x + y + z;
                pc++; break;
            case6:
                return 0;
    }
}
```

3. Call site unification(CSU)
   모든 함수 호출이 특정 경로를 지나도록 변경

```
char g(char* s, int l);
void h(int x, int y);

void f() {
    char* s = "Hi There";
    int l;
    char c;

    l = strlen(s);
    c = g(s, l);
    h(l, 3);
}
```

```
void f() {
    ...
    long p[2];

    p[0] = s;
    l = callfunc(0, p);

    p[0] = s; p[1] = l;
    c = callfunc(1, p);

  p[0] = l; p[2] = 3;
  callfunc(2, p);
}

long callfunc(int c, long* p) {
    switch (c)
    case 0:
        return strlen(p[0]);
    case 1:
        return g(p[0], p[1]);
    case 2:
        h(p[0], p[1]); return;
}
```

4. 기타 난독화 기술들
   문자열 변환, 변수명 변경 등

```
char* msg = "\x48\x65\x6c\x6c\x6f\x20\x57\x6f\x72\x6c\x64\x0a";
```

```
unsigned char* masterKey = { ... };
```

```
unsigned char* a1038 = { ... };
```

3 실험 및 결과 분석

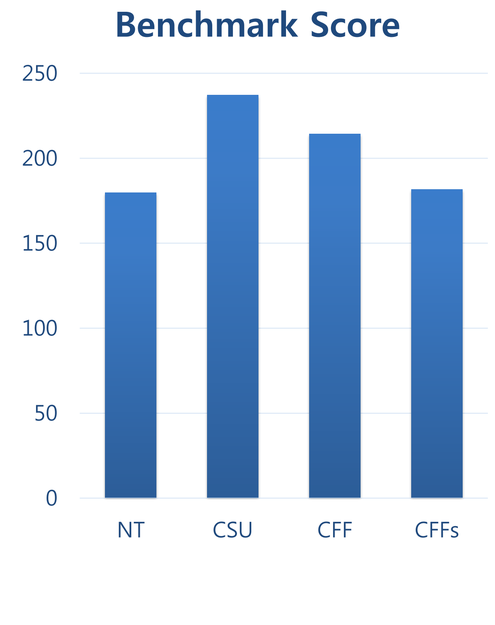
- 분석기: Zooberry
  - 안전한 C 소스코드 분석기
  - 다양한 분석 방법을 선택적으로 적용 가능

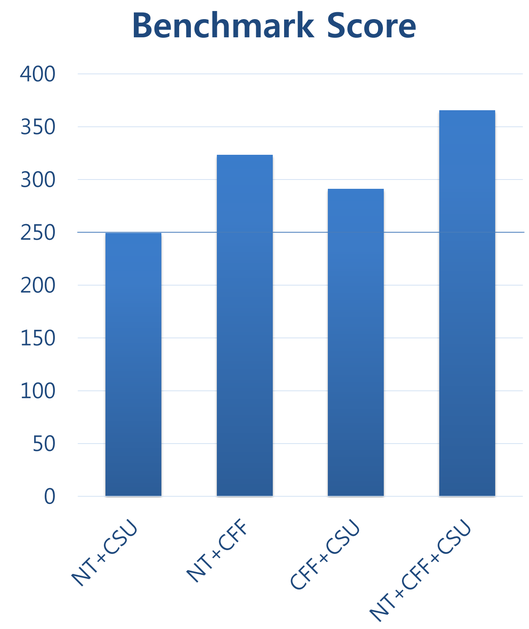
- 실험 방법
  - 각각의 난독화 기술 o_i와 샘플 프로그램 p_i에 대해 난독화 적용하지 않은 분석 결과 대비 증가율 계산

증가율	Dense		sparse	selective
	시간	#알람	시간	#알람
p_1	0.00	0.00	0.00	0.00
$o_1(p_1)$	0.15	0.10	0.25	1.14
$o_2(p_1)$	0.20	0.43	0.26	2.63
$o_3(p_1)$	0.11	0.14	0.37	5.37
$o_1(p_2)$	0.33	0.21	0.54	2.86
...				

- Benchmark 점수 계산
  1. 각 샘플 프로그램에 대해 항목 별로 표준 점수(편차치) 계산

     $T_i = \frac{10(x_i - \mu)}{\sigma} + 50$
  2. 난독화 알고리즘 별로 항목 별 표준 점수 합계 계산
  3. 샘플 프로그램별 점수 평균 계산





4 논의

- 장점과 한계
  - 장점
    - 객관적/합리적 근거에 의해 평가
    - 난독화 기술 구성(단독 또는 묶음)에 관계 없이 비교
    - 추후 성능 개선 가능: 분석 방식/샘플 프로그램 조정
  - 한계
    - 제약: 분석기 입력 형태 == 난독화 입/출력 형태
    - 절대적인 기준을 제시하는 것은 아님

- 계획
  - 샘플 프로그램 확충
  - 점수 계산법 개선: 항목별 Weight 튜닝

Programming Research Laboratory
SEOUL NATIONAL UNIVERSITY
ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 NRF ERC