연산자 스택 정보를 이용한 자바 프로그램 버스마킹 기법*

박희완, 임현일, 최석우, 한태숙
한국과학기술원 전자전산학과 전산학전공
e-mail:{hwpark, hilim, swchoi}@pllab.kaist.ac.kr, han@cs.kaist.ac.kr

A Static Java Birthmark using Operand Stack Information

Heewan Park, Hyun-il Lim, Seokwoo Choi, and Taisook Han
Division of Computer Science, Dept. of EECS, KAIST

요 약

소프트웨어 버스마크는 프로그램을 식별하는데 사용될 수 있는 프로그램의 고유한 특징을 말한다. 본 논문에서는 자바의 연산자 스택 정보에 기반한 자바 프로그램 버스마킹 기법을 제안한다. 자바의 연산은 스택을 중심으로 이루어지기 때문에 스택 정보로부터 프로그램의 고유한 특징을 얻어 낼 수 있다. 본 논문에서 제안한 버스마킹 기법을 평가하기 위해서 서로 다른 프로그램을 구별할 수 있는 신뢰도와 프로그램 최적화나 난독화에 견딜 수 있는 강인도에 대한 실험을 하였다. 실험 결과로부터 본 논문에서 제안하는 버스마크가 강인도를 유지하면서 프로그램의 특성을 표현하고 있음을 확인할 수 있다.

1. 서론

최근 많은 프로그램들이 오픈 소스 프로젝트로 개발되고 있으나 소프트웨어 개발 업체들이 GPL 기반의 공개 소스를 사용함에도 불구하고 소스를 공개하지 않는 등 GPL 규정을 따르지 않는 것으로 알려지고 있다. 이런 경우 소프트웨어 버스마크(Birthmark)는 개발 후 배포된 소프트웨어에 대해서 도용을 확인하는데 사용될 수 있다.

소프트웨어 버스마크는 프로그램을 식별하는데 사용될 수 있는 고유한 특징을 말한다. 프로그램으로부터 추출된 문자열을 서로 비교하거나 체크섬을 비교하는 것 등이 버스마킹 기법으로 사용될 수 있고, 이 경우에는 추출된 문자열이나 체크섬이 버스마크가 된다.

본 논문에서는 자바의 연산자 스택(Operand Stack) 정보에 기반한 자바 프로그램 버스마킹 기법을 제안한다. 자바의 연산은 스택을 중심으로 이루어지기 때문에 스택 정보로부터 프로그램의 고유한 특징을 얻어 낼 수 있다. 연산자 스택의 높이가 0에서 시작하여 다시 0이 되는 과정에서 실행된 바이트코드를 모아서 패턴의 집합으로 만들어서 버스마크로 사용한다.

추출된 정적 버스마크를 사용하여 두 프로그램의 유사도를 비교할 때 먼저 두 프로그램의 바이트 코드 패턴을 비교하고 그 결과를 이용하여 프로그램의 유사도를 계산한다. 바이트코드 패턴 비교에는 최대 공통 부분문자열[9]을 이용하고, 프로그램 비교에는 패턴들의 최대 매칭값을 얻기 위해서 헝가리안 알고리즘[10]을 이용한다.

본 논문에서 제안한 버스마킹 기법을 평가하기 위해서 예제 프로그램에 대해서 실험을 한다. 서로 다른 프로그램을 구별할 수 있는 신뢰도(Credibility)와 프로그램 최적화나 난독화에도 견딜 수 있는 강인도(Resilience)를 평가한다. 실험 결과로부터 본 논문에서 제안하는 버스마크가 강인도를 유지하면서 프로그램의 특성을 표현하고 있음을 확인할 수 있다.

2. 관련 연구

버스마크는 기법에 따라서 정적인 것과 동적인 것으로 나눌 수 있고, 대상 프로그램에 따라서 자바 클래스 파일과 Windows PE 파일 등으로 나눌 수 있다. [2, 3]은 자바의 정적 버스마크 기법이다. [2]는 자바에서 필드 변수에 사용된 상수 값들, 메쏘드 호출의 시퀀스, 상속 구조, 사용된 클래스들을 버스마크로 사용하였고, [3]은 실행 프로그램에 사용되는 opcode의 k-gram들의 집합을 버스마크로 사용하였다. [4, 5]는 Windows PE 파일의 정적 버스마크이다. [4]는 함수별 API 호출 집합을 버스마크로 이용하였고, [5]는 임포트 테이블에서 추출한 API 정보를 이용하였다. 이 기법들은 API 사용빈도가 낮은 프로그램에 대해서는 신뢰도가 떨어지는 단점이 있다.

[6, 7]은 자바의 동적 버스마크 기법이다. [6]은 전체 프로그램 패스(whole program path)로부터 추출된 명령어 시퀀스를 요약한 정보를 사용하였고, [7]은 자바 프로그램 실행 중의 호출된 API들을 기록하여 버스마크로 사용

* 본 연구는 정보통신부 및 정보통신연구진흥원의 대학 IT 연구센터 육성지원사업의 연구결과로 수행되었음(IITA-2007-C1090-0701-0020)

하였다. [8]은 Windows PE 파일의 동적 버스마크이다. 프로그램이 동작하는 동안 호출된 API 함수를 후킹하여 버스마크로 사용하였다.
동적인 버스마크는 정적인 버스마크와 비교했을때 신뢰도와 강인도가 높다는 장점이 있지만 사용자와의 상호 작용 없이 동작하는 프로그램에만 적용할 수 있다는 한계가 있다.

3. 연산자 스택에 기반한 정적 버스마크

3.1 소프트웨어 버스마크

Tamada와 Myles는 copy 관계를 이용하여 소프트웨어 버스마크를 정의했다. 다음은 버스마크에 대한 Myles의 정의[3]이다.

정의 1 (버스마크)
프로그램 p와 q에 대한 함수 f가 다음 조건을 만족할 때, f(p)를 프로그램 p의 버스마크라고 한다.
조건 1. f(p)는 부가적인 정보 없이 p 자신으로부터 얻는다.
조건 2. 프로그램 p와 q가 서로 copy 관계에 있다면 f(p) = f(q) 이다.

조건 1은 워터마킹과 버스마킹이 구분되는 특징이다. 버스마킹은 워터마킹과 달리 프로그램이 원래 가지고 있던 고유한 특징을 추출하는 방법이다.
조건 2는 copy된 두 프로그램의 경우에 같은 버스마크가 추출된다는 것을 나타낸다. 동일한 프로그램은 최적화나 난독화와 같은 프로그램 변환을 거치더라도 같은 버스마크가 추출되어야 한다.
다음 두 가지 속성은 Tamada와 Myles가 제시한 버스마크가 만족시켜야 하는 평가 기준을 나타낸다.

속성 1 (신뢰도)
같은 기능을 하는 두 프로그램 p와 q에 대해서, p와 q가 독립적으로 개발되었을 때, f(p)≠ f(q)을 만족해야 한다.
속성 2 (강인도)
프로그램 p'이 프로그램 p로부터 변환되었다고 할 때, f(p) = f(p')을 만족해야 한다.

속성 1은 버스마킹 기법의 신뢰도를 나타낸다. 두 프로그램이 비록 동일한 기능을 하는 프로그램이라고 하더라도 독립적으로 개발된 프로그램이라면 두 프로그램에서 추출된 버스마크는 같지 않아야 한다.
속성 2는 버스마킹 기법의 강인도를 나타낸다. 컴파일러 최적화나 난독화와 같은 프로그램 변환 기법을 적용하기 전과 후의 버스마크는 같아야 한다.

3.2 자바 연산자 스택 기반 버스마크

그림 1은 두 정수 x, y의 합과 곱을 구하는 자바 예제 프로그램이다.

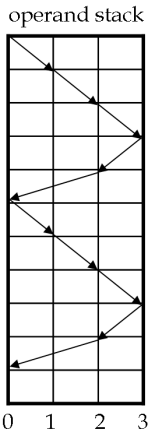
```
public class Test {
    int s;
    int p;

    public void f(int x, int y)
    {
        s = x + y;
        p = x * y;
    }
}
```

(그림 1) 자바 예제 프로그램

그림 2는 이 예제 프로그램을 javac로 컴파일 했을 때 생성된 클래스 파일에서 함수 f의 바이트코드를 나타낸 것이다. 바이트코드 실행에 따르는 연산자 스택의 변화는 정적으로 예측할 수 있기 때문에 각각의 바이트코드 실행에 따르는 스택의 변화를 기록할 수 있다. 여기서 스택의 높이가 0인 상태에서 다시 0이 되는 동안 실행되는 바이트코드를 묶어서 하나의 패턴으로 생각할 수 있는데 이 패턴은 자바언어의 명령문(statement)과 근사하게 매핑될 수 있다. 예를 들어, 그림1의 f함수를 이루고 있던 두 명령문은 그림2의 스택 패턴 2개와 매핑된다.

```
public void f(int, int);
  Code:
    0:   aload_0
    1:   iload_1
    2:   iload_2
    3:   iadd
    4:   putfield #2;//s
    7:   aload_0
    8:   iload_1
    9:   iload_2
    10:  imul
    11:  putfield #3;//p
    14:  return
```



(그림 2) 바이트코드에 의한 스택의 변화

본 논문에서 제안하는 스택 기반 버스마크는 자바 클래스 파일로부터 스택 패턴들을 추출해내고 이 패턴들을 버스마크로 사용한다.

정의 2 (패턴의 유사도)
두 패턴 A, B에 대해서, LCS[9]는 최대 공통 부분문자열(Longest Common Substring)이고, SCS는 최소 공통 합문자열(Shortest Common Superstring)이라고 할때 두 패턴간 유사도는 다음과 같이 정의한다.

$$\text{Similarity}(A,B) = \frac{|\text{LCS}(A,B)|}{|\text{SCS}(A,B)|}$$

(여기서 |SCS(A,B)| = |A|+|B| - |LCS(A,B)|이므로)

$$\text{Similarity}(A,B) = \frac{|\text{LCS}(A,B)|}{|A|+|B|-|\text{LCS}(A,B)|}$$

<표 1> 예제 패턴 A, B에 대한 LCS 계산

line	pattern A	abstract bytecode	stack	pattern B	abstract bytecode	stack	LCS
1	iload_0	**L**oad	*	iload_0	**L**oad	*	
2	iload_0	**L**oad	**	iconst_1	**C**onst	**	
3	iconst_1	**C**onst	***	isub	**S**ub	*	L
4	isub	**S**ub	**	invoke	in**V**oke	*	L
5	invoke	in**V**oke	**	iload_0	**L**oad	**	C
6	imul	**M**ul	*	iconst_2	**C**onst	***	S
7	ireturn	**R**eturn		isub	**S**ub	**	V
8				invoke	in**V**oke	**	R
9				iadd	**A**dd	*	
10				ireturn	**R**eturn		

표 1의 예제 패턴 A, B에 대해서 유사도와 포함정도를 정의 1과와 정의 2에 의해서 계산해보면 다음과 같다. 두 패턴 A, B의 LCS는 "LLCSVR"임을 알 수 있다. 정의2에 의해서 유사도를 구하면 다음과 같다.

유사도 = 6 / (7 + 10 - 6) = 6 / 11 = 0.5455

<표 2> 패턴간의 유사도 행렬

Q의 패턴 / P의 패턴	Q1	Q2	Q3	Q4
P1	0.2000	0.2000	0.3333	0.1000
P2	0.7500	0.7500	0.6666	0.2000
P3	0.3000	0.3000	0.4000	0.1500
P4	0.2000	0.2000	1.0000	0.2000

프로그램의 유사도는 프로그램을 구성하고 있는 패턴의 유사도를 이용하여 최대 매칭값을 얻는 방식으로 구한다. 예를 들면, 프로그램 P와 Q가 각각 4개의 패턴으로 이루어져 있고 각각의 패턴간 유사도를 구한 행렬을 표2라고 하자. 이 행렬에 대해서 헝가리안 알고리즘[10]을 이용하면 패턴의 개수를 N이라고 할때 $O(N^3)$시간에 매칭값의 총 합을 최대로 하는 매칭을 찾아낼 수 있다. <표2>로부터 얻어낸 최대 매칭은 P1-Q4, P2-Q2, P3-Q1, P4-Q3 이고, 이때 매칭값의 합은 0.10 + 0.75 + 0.30 + 1.00 = 2.15이다. 이 값을 패턴의 크기인 4로 나눈 값인 2.15/4 = 0.5375가 프로그램의 유사도 값이다.

정의 3 (프로그램의 유사도)

두 프로그램 P와 Q에 대해서 S(P)와 S(Q)를 각각 프로그램 P와 Q의 패턴들의 집합이라고 할 때 두 프로그램의 유사도는 정의 2의 패턴의 유사도를 이용하여 다음과 같이 정의한다.

$$\frac{\sum_{(A,B)\in \mathrm{match}(S(P),S(Q))} \mathrm{Similarity}(A,B)}{\mathrm{MAX}(|S(P)|,|S(Q)|)}$$

여기서, match(S(P),S(Q))는 헝가리안 알고리즘에 의해서 찾아진 패턴 쌍들의 집합을 의미한다. 즉, 매칭된 패턴들 사이의 유사도 합을 구하고 큰 프로그램의 패턴 갯수로 나눈다.

정의 4 (프로그램 포함정도)

두 프로그램 P와 Q에 대해서 P의 Q에 대한 포함정도는 다음과 같이 정의한다.

$$\frac{\sum_{(A,B)\in \mathrm{match}(S(P),S(Q))} \mathrm{Similarity}(A,B)}{|S(P)|}$$

프로그램의 포함정도를 계산할 때는 분모를 S(P)의 크기로 하여 P가 Q에 포함된 정도를 측정한다.

3.3 버스마크 공격에 대한 방어

일단 버스마킹 기법이 공개되면 버스마크를 추출하지 못하도록 다양한 공격이 시도될 수 있다. 예를 들면, 최적화 도구나 난독화 도구에 의해서 바이트코드의 순서가 바뀌거나, 기존의 바이트코드가 유사한 다른 바이트코드로 교체되거나, 부가적으로 새로운 바이트 코드가 추가될 수 있다.

바이트 코드의 순서가 바뀌는 것은 헝가리안 알고리즘의 최대 매칭 계산으로 보완될 수 있다. int 연산을 long 연산으로 수정하는 등의 공격을 방어하기 위해서 202개의 바이트 코드를 타입정보가 생략된 44개의 abtract 코드로 변경하여 유사도를 비교하였다. 부가적으로 바이트코드를 프로그램에 추가하여 유사도를 떨어뜨리는 공격을 보완하기 위해서 프로그램의 유사도와 함께 프로그램의 포함정도(Contatinment)를 계산하였다.

4. 실험 및 평가

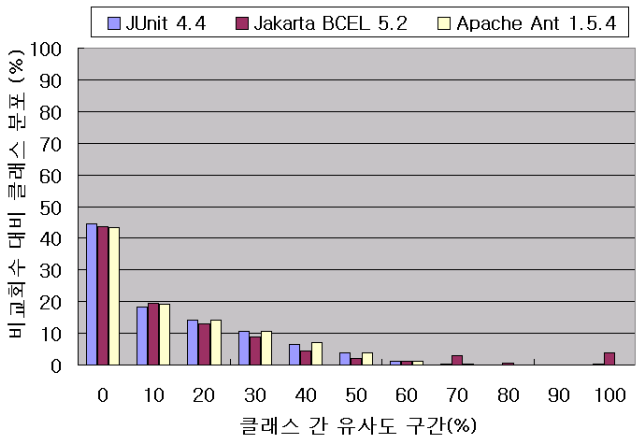
본 논문에서 제안한 연산자 스택 정보를 이용한 버스마킹 시스템은 Windows 환경에서 C++ 프로그래밍 언어로 구현하였다. 프로그램의 동작은 다음과 같다. 먼저 두 개의 자바 클래스 파일을 입력으로 받아서 프로그램의 스택 패턴을 생성한다. 생성된 패턴을 구성하는 바이트코드 명령어 집합으로부터 LCS값과 SCS값을 계산하여 패턴간의 유사도 행렬을 만든다. 이 행렬을 헝가리안 알고리즘의 입력으로 넣어서 유사도의 합을 최대로 하는 매칭을 찾고 유사도와 포함정도를 계산하여 출력한다.

신뢰도와 강인도에 대한 실험을 위해서 사용된 예제 프로그램은 표 3과 같다.

<표 3> 신뢰도와 강인도 실험을 위한 예제 프로그램

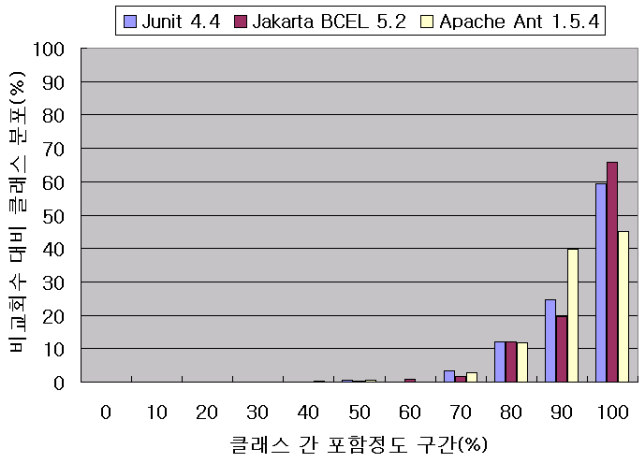
예제 프로그램	JUnit 4.4[11]	BCEL 5.2[12]	Ant 1.5.4[13]
클래스 개수	150	378	406
신뢰도 비교회수	11175	71253	82215
강인도 비교회수	150	378	406

신뢰도는 각 예제 프로그램 패키지에 포함된 클래스에서 추출가능한 2개의 클래스 조합에 대한 유사도를 비교하였고, 강인도는 자바 난독화 도구인 Smokescreen[14]를 사용하여 난독화 전과 후의 클래스 포함정도를 비교하였다.



(그림 3) 예제 프로그램에 대한 신뢰도 비교

그림 3은 예제 프로그램에 대한 신뢰도 비교 결과이다. 서로 다른 클래스에 대한 비교이므로 낮은 값을 가질수록 신뢰도가 높다.



(그림 4) 예제 프로그램에 대한 강인도 비교

그림 4는 예제 프로그램에 대한 강인도 비교 결과이다. 이 값은 높을 수록 강인도가 높다.

신뢰도와 강인도 실험 결과를 평가해보면 헝가리안 알고리즘과 abstract 바이트코드의 적용으로 인해서 강인도가 높으나 신뢰도가 낮아지는 trade-off가 있음을 확인하였다.

5. 결론 및 향후 연구 과제

소프트웨어 버스마크는 프로그램을 식별하는데 사용될 수 있는 고유한 특징을 말한다. 본 논문에서는 자바의 연산자 스택 정보에 기반한 자바 프로그램 버스마킹 기법을 제안하였다. 자바의 연산은 스택을 중심으로 이루어지기 때문에 스택 정보로부터 프로그램의 고유한 특징을 얻어 낼 수 있다. 추출된 버스마크를 사용하여 두 프로그램의 유사도를 비교할 때 바이트코드 패턴 비교에는 최대 공통 부분문자열을 이용하였고, 프로그램 비교에는 패턴들의 최대 매칭값을 얻기 위해서 헝가리안 알고리즘을 이용하였다.

본 논문에서 제안한 버스마킹 기법을 평가하기 위해서 서로 다른 프로그램을 구별할 수 있는 신뢰도와 프로그램 최적화나 난독화에도 견딜 수 있는 강인도에 대한 실험을 하였다. 실험 결과로부터 본 논문에서 제안하는 버스마크가 강인도를 유지하면서 프로그램의 특성을 표현하고 있음을 확인할 수 있다.

향후 연구 과제는 다음과 같다. 강인도를 높이기 위해서 매칭 방법으로서 선택한 헝가리안 알고리즘이 명령어 순서 재배치(Instruction Reordering) 범위를 벗어난 문장들의 매칭을 가능하게 만들기 때문에 이 문제점을 보완해야 한다. 그리고 자바 명령어(opcode)에 가중치를 주어서 신뢰도를 높이는 방법을 고려하고 있으며, 기존의 버스마킹 기법과의 비교 평가 및 다양한 자바 컴파일러와 난독화 도구에 대한 강인도 실험을 계획하고 있다.

참고 문헌

[1] T. Lindholm and F. Yellin, The Java Virtual Machine Specification Second Edition, Addison-Wesley, April 1999.

[2] Tamada, H., Nakamura, M., Monden, A., Matsumoto, K. Java birthmark Detecting the software theft. IEICE Transactions on Information and Systems, E88-D, 9 (Sept. 2005), 2148-2158

[3] Ginger Myles and Christian Collberg. k-gram Based Software Birthmarks. In Proceeding of the 2005 ACM Symposium on Applied Computing, pp. 314-318. Santa Fe, New Mexico, USA, 2005.

[4] Seokwoo Choi, Heewan Park, Hyun-il Lim, and Taisook Han. A Static Birthmark of Binary Executables Based on API call Structure. ASIAN 2007, Carnegie Mellon University Qatar Campus, Doha, Qatar, 2007.

[5] 박희완, 임현일, 최석우, 한태숙. Windows PE 파일의 임포트 테이블에 기반한 소프트웨어 버스마킹 기법. 정보과학회 제34회 추계학술대회, 2007년 10월.

[6] Ginger Myles and Christian Collberg. Detecting Software Theft via Whole Program Path Birthmarks. ISC 2004, LNCS 3225, pp. 404-415, 2004.

[7] David Schuler, Valentin Dallmeier, and Christian Lindig. A Dynamic Birthmark for Java. ASE'2007, Atlanta, Georgia, USA, 2007.

[8] Haruaki Tamada and Keiji Okamoto. Dynamic software birthmarks to detect the theft of windows applications. In Proc. Int. Symposium on Future Software Technology 2004, Xi-an, China, 2004.

[9] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein (2001). Introduction to Algorithms, second edition, MIT Press and McGraw-Hill

[10] Harold W. Kuhn, The Hungarian Method for the assignment problem, Naval Research Logistic Quarterly, 2:83-97, 1955.

[11] "JUnit" http://www.junit.org

[12] "Jakarta BCEL" http://jakarta.apache.org/bcel/

[13] "Apache Ant" http://ant.apache.org

[14] "Smokescreen" http://www.leesw.com/smokescreen/