정적 API 트레이스 버스마크를 이용한 자바 클래스 도용 탐지

(Detecting Java Class Theft using Static API Trace Birthmark)

박 희 완 † 최 석 우 ††
(Heewan Park) (Seokwoo Choi)
임 현 일 † 한 태 숙 †††
(Hyun-il Lim) (Taisook Han)

요 약 소프트웨어 버스마크는 프로그램을 식별하는데 사용될 수 있는 프로그램의 고유한 특징을 말한다. 본 논문에서는 정적 API 트레이스 정보를 이용하여 자바 클래스 도용을 탐지하는 방법을 제안한다. 정적 API 트레이스를 생성할 때 제어 흐름을 분석하여 버스마크의 강인성을 높였고, 트레이스를 비교할 때 준전체 정렬 방법을 사용하여 서로 다른 프로그램을 구별할 수 있는 신뢰성을 높였다. XML Parser 패키지에 대한 신뢰도와 강인도 실험 결과를 통하여 본 논문에서 제안하는 정적 API 트레이스 버스마크가 자바 클래스 도용을 탐지하는데 있어서 기존의 버스마크들보다 효과적임을 보였다.

키워드 : 코드 도용, 정적 분석, 소프트웨어 버스마크

Abstract Software birthmark is the inherent characteristics that can identify a program. In this paper, we propose a Java class theft detection technique based on static API traces of class files. We utilize control flow analysis to increase resilience, and we apply the semi-global alignment trace comparison algorithm to increase credibility. The credibility and resilience experiments for XML parsers show that our birthmark is more efficient than existing birthmarks.

Key words : Code Theft, Static Analysis, Software Birthmark

· 본 연구는 지식경제부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업의 연구결과로 수행되었음(IITA-2008-C1090-0801-0020)
· 이 논문은 2008 한국컴퓨터종합학술대회에서 '정적 API 트레이스 버스마크를 이용한 자바 클래스 도용 탐지'의 제목으로 발표된 논문을 확장한 것임

† 비 회 원 : KAIST 전산학전공
hwpark@pllab.kaist.ac.kr
hilim@pllab.kaist.ac.kr
†† 학생회원 : KAIST 전산학전공
swchoi@pllab.kaist.ac.kr
††† 종신회원 : KAIST 전산학전공 교수
han@pllab.kaist.ac.kr

논문접수 : 2008년 8월 27일
심사완료 : 2008년 10월 23일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제14권 제8호(2008.11)

1. 서 론

오늘날 세계적으로 수많은 소프트웨어들이 시시각각 작성되어 배포되고 있다. 최근 인터넷 환경의 발전으로 소프트웨어가 쉽게 복제 및 배포될 수 있어서 소프트웨어 도용 사례가 급격히 증가하고 있다[1].

프로그램 도용에 대한 연구는 소스 표절 검사 기법[2]에서 많이 다루어지고 있다. 그러나 프로그램은 컴파일되어 배포되기 때문에 항상 소스를 얻을 수 있는 것은 아니다. 결국 바이너리나 자바 클래스 파일을 대상으로 적용할 수 있는 도용 탐지 기법을 사용하여야 한다. 소프트웨어 버스마크[3,4]는 바이너리나 클래스 파일로부터 직접 고유한 특징을 추출하여 비교하는 방법이다.

본 논문에서는 자바의 정적 API 트레이스를 이용한 새로운 버스마크 기법을 제안한다. 여기서 API란 응용 프로그램이 운영체제가 제공하는 기본적인 기능들을 사용할 수 있도록 해주는 도구이다. 즉, API 함수 호출 명령어는 다른 것으로 대체하거나 제거하기 어렵다는 특징이 있다. 이 사실을 근거로하여 정적 API 트레이스 버스마크 기법은 프로그램 실행 중에 호출될 수 있는 자바 API 함수 시퀀스의 집합을 프로그램의 고유한 특징으로 정의한다.

본 논문은 다음과 같이 구성되어 있다. 2장에서는 버스마크에 대한 선행 연구들을 살펴본다. 3장에서는 본 논문에서 제시하는 정적 API 트레이스 버스마크를 정의한다. 4장과 5장에서는 각각 정적 API 트레이스의 생성 방법과 유사도 비교 방법을 설명한다. 6장에서는 실험 결과를 기반으로 본 논문의 버스마크를 평가한다. 끝으로 7장에서는 결론을 맺고, 향후 연구 과제를 소개한다.

2. 관련 연구

Tamada는 자바 클래스 파일에 대한 정적 소프트웨어 버스마크 기법[3]을 처음 제안했다. 이 기법은 자바 클래스의 필드 변수에 사용된 상수 값들의 시퀀스 정보, 자바 메소드 호출 시퀀스 정보, 클래스 상속 구조 정보, 사용된 클래스 정보의 4가지 버스마크로 구성된다. 자바 클래스의 구조적인 특징 위주로 버스마크를 구성하였기 때문에 난독화와 같은 프로그램 변환 기법에도 버스마

크가 쉽게 손상되지는 않지만, 클래스 구조는 동일하면서 서로 다른 알고리즘을 구현한 두 프로그램은 구별할 수 없다는 단점이 있다.

Myles는 k-gram 버스마크[4]를 제안하였다. 여기서 k-gram이란 자바 클래스에서 연속된 k개의 JVM 명령어 시퀀스 집합을 의미한다. 즉, 자바 클래스를 이루고 있는 JVM 명령어를 길이가 k인 조각으로 잘라서 중복된 것을 제외하고 버스마크로 사용한다. k-gram 버스마크는 JVM 명령어 레벨에서 유사도를 비교하기 때문에 서로 다른 프로그램을 구별하는 능력은 뛰어나지만 난독화나 컴파일러에 따라서 JVM 명령어가 변경되거나 시퀀스가 바뀌는 경우가 발생할 수 있기 때문에 버스마크가 쉽게 손상된다는 단점이 있다.

3. 정적 API 트레이스 버스마크

본 논문에서 제안하는 정적 API 트레이스 버스마크에 대해서 설명하기 전에 먼저 일반적인 소프트웨어 버스마크의 정의와 속성에 대해서 알아본다.

3.1 소프트웨어 버스마크

Myles는 소프트웨어 버스마크를 다음과 같이 정의했다[4].

정의 1 (버스마크)

프로그램 p 와 q 에 대한 함수 f가 다음 조건을 만족할 때, $f(p)$ 를 프로그램 p 의 버스마크라고 한다.

조건 1. $f(p)$ 는 부가적인 코드의 삽입 없이 p 자신으로부터 얻는다.

조건 2. 프로그램 p 와 q가 서로 코드 도용 관계에 있다면 $f(p) = f(q)$ 이다.

다음 속성은 버스마크가 만족시켜야 하는 성질이다.

속성 1 (신뢰성: Credibility)

같은 기능을 하는 두 프로그램 p 와 q 에 대해서, p 와 q가 서로 소스 코드를 공유하지 않고 개발되었을 때, $f(p) \neq f(q)$를 만족해야 한다.

속성 2 (강인성: Resilience)

프로그램 p' 이 프로그램 p로부터 프로그램 최적화나 난독화 기법과 같이 프로그램의 의미를 바꾸지 않는 변환 기법에 의해서 변환되었다고 할 때, $f(p) = f(p')$을 만족해야 한다.

3.2 정적 API 트레이스 버스마크

정의 2 (API 흐름 그래프)

자바 메소드의 제어 흐름 그래프 $G = (V, E, Vs, Ve)$에 대해서 G의 시작 블록을 Vs, 종료 블록을 Ve라고 할 때 다음 조건을 만족하는 $G' = (V', E', Vs, Ve)$ 를 자바 메소드의 API 흐름 그래프라고 정의한다.

조건 1. $V' = V - \{v \in V \mid v$에서 API를 호출하지 않는다.$\}$

조건 2. $E' = \{(vf, vt) \mid (vf, v) \in E \wedge (v, vt) \in E \wedge v \notin V'\}$

그림 1은 자바 바이트코드의 제어 흐름 그래프를 정의 2의 API 흐름 그래프로 변경한 예이다.

정의 3 (정적 API 트레이스, API 트레이스 집합)

자바 메소드의 API 흐름 그래프 $G = (V', E', Vs, Ve)$에 대해서 Vs에서 $e \in E$를 따라서 Ve에 이르는 경로에서 사용되는 API 함수 시퀀스를 정적 API 트레이스라고 정의한다. 그리고 메소드에 포함되는 모든 정적 API 트레이스들의 집합을 메소드의 API 트레이스 집합이라고 정의한다.

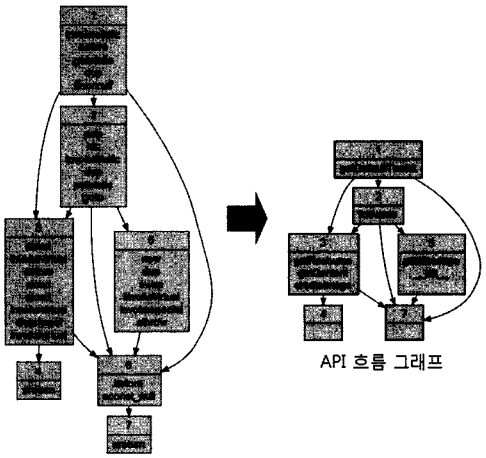

그림 1 자바 바이트코드의 제어 흐름 그래프와 API 흐름 그래프

정의 4 (정적 API 트레이스 버스마크)

정적 API 트레이스 버스마크는 트레이스를 구성하고 있는 API 시퀀스 중에서 서로 인접하여 중복 호출되는 API를 1개로 축약시킨 정적 API 트레이스 집합이라고 정의한다.

4. 정적 API 트레이스 생성

4.1 바이트코드의 제어 흐름 그래프 생성

자바 클래스 파일의 제어 흐름 그래프(이하 CFG)를 생성하기 위해서 먼저 클래스 파일을 디스어셈블하여 JVM 명령어로 변경하고 분기와 관련된 명령어를 해석하여 CFG를 생성한다. CFG 생성 방법은 컴파일러 이론에서 일반적으로 언급되는 알고리즘[5]을 따른다.

JVM 명령어로부터 CFG를 생성할 때 특별하게 고려해야 할 부분은 서브루틴 호출(jsr)과 예외 상황(exception)에 대한 처리이다. `jsr` 같은 서브루틴 호출 명령어의 경우는 `ret` 문을 만나면 `jsr`과 인접한 다음 명령어로 제어가 넘어가기 때문에 `jsr` 목표 지점에 있는 `ret` 문과, `jsr` 문에서 인접한 기본 블록을 연결해야

한다.

예외 처리는 원칙적으로 예외 범위에 포함된 모든 JVM 명령어를 예외 처리 루틴의 시작 지점과 연결해야 한다. 이 경우에 예외 범위 사이의 모든 명령어들이 각각 기본 블록이 되는 상황이 발생하기 때문에 CFG의 에지가 급격히 증가하는 문제가 생긴다. 따라서 본 논문에서는 예외 범위에 포함된 기본 블록들을 예외 처리 루틴과 연결하는 방법으로 예외 처리에 대한 에지 추가를 최소화했다.

4.2 정적 API 트레이스 생성

CFG로부터 API 트레이스를 생성하기 위해서는 먼저 CFG를 API 흐름 그래프로 변경해야 한다. CFG의 노드 중에서 API를 포함하지 않는 노드를 삭제하면서 삭제된 노드로 진입하는 에지를 삭제된 노드에서 나가는 에지에 연결시키는 작업을 한다. 일단 API 흐름 그래프가 완성되면 시작 지점에서 DFS 탐색을 시작한다. 그리고 종료 지점에 도달할 때마다 API 트레이스를 1개씩 생성한다. 이 작업을 모든 경로를 방문하는 동안 반복하면 API 트레이스 집합을 생성할 수 있다. 이때 문제가 되는 것이 반복문에 대한 처리이다. 반복문은 정적인 분석으로는 반복 회수를 알 수 없는 경우가 많다. 본 논문에서는 모든 반복문에 대해서 역방향 에지를 한번까지만 허용하도록 처리했다.

서브루틴에 대한 트레이스 생성을 위해서는 서브루틴 스택을 사용해야 한다. `jsr` 문에 의해서 분기가 될 때 복귀 지점을 스택에 저장하고 서브루틴으로 분기한다. 서브루틴이 `ret` 문을 만나서 종료되면 서브루틴 스택에 저장된 복귀 지점으로 분기한다. 본 논문에서는 서브루틴 스택을 이용하여 재귀적으로 서브루틴이 호출되는 경우도 고려하여 트레이스를 생성하였다.

예외상황에 대한 트레이스 생성은 CFG 생성과정에서 예외상황 에지가 이미 추가되었기 때문에 일반적인 분기문과 동일하게 예외상황 에지를 탐색하여 트레이스를 생성한다.

생성된 트레이스 집합은 JVM 명령어의 시퀀스로 이루어져있다. 이중에서 자바6 표준 패키지[1]에 포함된 API를 호출하는 `invoke` 명령어만을 선택하여 API 트레이스로 변환한다.

5. 준(準)전체 정렬을 이용한 유사도 계산

두 프로그램으로부터 추출된 트레이스를 비교하는 방법은 생물 정보학 분야의 유전자 비교에서 주로 사용되는 시퀀스 정렬(sequence alignment)을 사용하였다. 대표적인 시퀀스 정렬에는 다음과 같이 3가지가 있다. 첫째, 시퀀스 전체를 비교 대상으로 하여 유사도를 구하는 전체 정렬(global alignment), 둘째, 부분 일치에 초점을 두는 지역 정렬(local alignment), 셋째, 전체 정렬과 비슷하지만 길이가 서로 다른 시퀀스 비교를 고려하여 전단부(prefix)와 후단부(suffix)에 대해서 감점을 주지 않는 방식으로 유사도를 계산하는 준전체 정렬(semi-global alignment)이 있다. 본 논문에서는 트레이스 유사도 계산을 위하여 준전체 정렬[6]을 사용하였다. 본 논문에서 적용한 준전체 정렬은 전체 정렬의 정의를 이용하여 아래와 같이 정의할 수 있다.

정의 5 (전체 정렬)

원본 트레이스 T_1과 도용 트레이스 T_2에 대하여 $\sigma(i, j)$를 $T_1[i]$와 $T_2[j]$의 일치 여부에 대한 점수라고 하고, gap은 공백 삽입에 의한 감점이라고 했을 때, $T_1[1..i]$와 $T_2[1..j]$의 정렬 점수를 최대로 하는 전체 정렬 $G(i, j)$는 다음과 같이 정의한다.

$$\text{전체 정렬 } G(i, j) = max \begin{cases} G(i-1, j-1) + \sigma(i, j) \\ G(i-1, j) + gap \\ G(i, j-1) + gap \end{cases}$$

$$\sigma(i, j) = \begin{cases} 1 : T_1[i] = T_2[j] \\ -1 : T_1[i] \neq T_2[j] \end{cases}, \quad gap = -1$$

정의 6 (준전체 정렬)

준전체 정렬 $SG(i, j)$는 전체 정렬 $G(i, j)$를 이용하여 다음과 같이 정의한다.

$$SG(i, j) = max\{G(i, 1), G(i, 2), ..., G(i, j), 0\} \quad (1)$$

$$\text{단, } G(0, 0) = G(0, 1) = G(0, 2), ..., G(0, j) = 0 \quad (2)$$

정의 7 (트레이스 유사도)

트레이스 T_1과 T_2 중 어느 것이 원본이고 어느 것이 도용된 것인지 알 수 없는 상황에서 준전체 정렬을 이용하여 T_1과 T_2의 트레이스 유사도 T_{sim}을 다음과 같이 정의한다.

$$T_{sim}(T_1, T_2) = max\left(\frac{SG(i, j)}{|T_1|}, \frac{SG(j, i)}{|T_2|}\right)$$

위 정의는 T_1을 원본이라고 가정한 유사도와 T_2를 원본이라고 가정 한 유사도 값 중에서 최대값을 취한 형태이다.

정의 8 (메소드 유사도)

두 메소드 M과 N으로부터 생성된 트레이스를 $TS_1[1..m]$과 $TS_2[1...n]$이라고 할 때, TS_1과 TS_2의 모든 트레이스 조합에 대해서 $m * n$ 크기의 SG 행렬을 생성하면 두 메소드 M과 N의 유사도 M_{sim}은 다음과 같이 정의한다.

1) http://java.sun.com/javase/6/docs/api/ 문서에 기술된 패키지를 추출 대상 패키지로 선정하였다. java.applet부터 org.xml.sax.helpers에 이르기까지 총 202개의 패키지가 포함된다.

$$row_{max}(i) = max(SG(i,0), ..., SG(i,n))$$
$$col_{max}(j) = max(SG(0,j), ..., SG(m,j))$$

$$sum_{row}(M,N) = \sum_{i=1}^{m} row_{max}(i),$$

$$sum_{col}(M,N) = \sum_{j=1}^{n} col_{max}(j)$$

$$sum_{trace}(M) = \sum_{i=1}^{m} |TS_1[i]|,$$

$$sum_{trace}(N) = \sum_{j=1}^{n} |TS_2[j]|$$

$$M_{sim}(M,N) = max\left(\frac{sum_{row}(M,N)}{sum_{trace}(M)}, \frac{sum_{col}(M,N)}{sum_{trace}(N)}\right)$$

정의 9 (클래스 유사도)

두 클래스 C와 D에 포함된 메소드가 각각 $M[1..m]$과 $N[1..n]$일 때, 두 클래스의 유사도 C_{sim}은 다음과 같이 정의한다.

$$map(M[i]) = max(sum_{row}(M[i], N[1]), ..., sum_{row}(M[i], N[n]))$$
$$map(N[j]) = max(sum_{col}(M[1], N[j]), ..., sum_{col}(M[m], N[j]))$$

$$C_{sim}(C,D) = max\left(\frac{\sum_{i=1}^{m} map(M[i])}{\sum_{i=1}^{m} sum_{trace}(M[i])}, \frac{\sum_{j=1}^{n} map(N[j])}{\sum_{j=1}^{n} sum_{trace}(N[j])}\right)$$

정의 10 (패키지 유사도)

두 패키지 P와 Q에 포함된 클래스를 $C[1..m]$과 $D[1..n]$라고 하고 $C[i]$의 메소드를 $M_i[1..c_i]$라고 하고 $D[j]$의 메소드를 $N_j[1..d_j]$라고 할 때, 두 패키지의 유사도 P_{sim}은 다음과 같이 정의한다.

$$P_{sim}(P,Q) = max\left(\frac{\sum_i \sum_k map(M_i[k])}{\sum_i \sum_k sum_{trace}(M_i[k])}, \frac{\sum_j \sum_k map(N_j[k])}{\sum_j \sum_k sum_{trace}(N_j[k])}\right)$$

6. 실험 및 평가

정적 API 트레이스 버스마크에 대한 실험을 할 때, 추출된 API 트레이스 집합에서 트레이스 길이가 3이상인 것을 실험 대상으로 선정하였다. 그 이유는 작은 트레이스는 메소드의 특성이 포함되기 어렵다고 판단했기 때문이다. 실험 대상으로는 표 1과 같이 XML Parser 패키지 4개를 선택하였다. 난독화 도구에 대한 강인도를 측정하기 위해서 대표적인 난독화 도구인 Smokescreen[7]을 사용하였다.

표 1 실험 대상 프로그램

	원본 클래스 개수	원본 패키지크기 (byte)	실험대상 클래스 개수	실험대상 패키지 크기 (byte)	실험대상과 원본의 크기 비율
Aelfred 7.0	7	60,608	2	55,036	91%
Crimson 1.1.3	145	355,230	57	274,735	77%
Piccolo 1.04	87	323,018	38	227,729	71%
XP 0.5	88	150,562	11	83,683	56%

표 2 난독화 및 컴파일러 변경에 대한 강인도 실험

	Smokescreen			Jikes		
	Tamada	k-gram	API 트레이스	Tamada	k-gram	API 트레이스
Aelfred	0.758	0.671	0.980	0.857	0.892	0.975
Crimson	0.689	0.563	0.996	0.839	0.871	0.998
Piccolo	0.733	0.614	0.998	0.832	0.891	0.998
XP	0.720	0.589	1.000	0.920	0.894	0.984
평균	0.725	0.609	0.994	0.862	0.887	0.989

API 트레이스 버스마크와 비교를 하기 위해서 대표적인 정적 버스마크 기법인 Tamada 버스마크와 k-gram 버스마크를 선택하였고 k-gram에서 k값은 3을 선택하였다.

표 2는 강인성 실험 결과이다. 예제 프로그램을 Smokescreen으로 변환 후 원본 프로그램과 비교하였고, 프로그램 소스를 Javac 컴파일러와 Jikes 컴파일러[8]로 컴파일한 결과를 비교하였다. 그 결과 두가지 경우 모두 API 트레이스 버스마크가 가장 강인도가 높게 평가되었다.

Smokescreen이나 Jikes는 메소드의 제어 흐름을 변경하는데 Tamada 버스마크는 제어 흐름을 반영한 메소드 시퀀스를 추출하지 않고 단순히 바이트 코드 상에서 인접한 메소드 시퀀스를 찾기 때문에 변형에 취약하다. k-gram의 경우도 바이트 코드 상에서 인접한 k개의 JVM명령어 집합을 추출하기 때문에 제어 흐름 변경에 취약하다. API 트레이스는 제어 흐름을 반영하기 때문에 높은 강인도 값을 얻었다.

표 3 Tamada 신뢰도 실험 결과(평균:0.392)

	Crimson	Piccolo	XP
Aelfred	0.281	0.445	0.363
Crimson	-	0.575	0.306
Piccolo	-	-	0.380

표 4 k-gram (k=3) 신뢰도 실험 결과(평균:0.230)

	Crimson	Piccolo	XP
Aelfred	0.261	0.224	0.184
Crimson	-	0.456	0.131
Piccolo	-	-	0.126

표 5 API 트레이스 신뢰도 실험 결과(평균:0.111)

	Crimson	Piccolo	XP
Aelfred	0.018	0.013	0.110
Crimson	-	0.341	0.115
Piccolo	-	-	0.068

표 3,4,5는 신뢰도 실험 결과이다. Tamada, k-gram, API 트레이스 버스마크와 상관없이 Crimson과 Piccolo의 유사도 값이 가장 높은 값으로 계산되었다. 위 결과로부터 Crimson과 Piccolo는 서로 도용 관계가 있다는 것을 어느 정도 예측할 수 있다. 실제로 두 패키지 모두 오픈 소스 라이브러리인 xml.parsers와 xml.sax를 포함하고 있었다.

버스마크의 성능을 더 엄밀하게 평가하기 위해서 Crimson을 Smokescreen으로 난독화 변환을 시킨 Crimson-smoke 패키지를 Piccolo 패키지와 비교하였다.

표 6 Crimson-smoke와 Piccolo의 유사도 분포

클래스 유사도 구간	해당 유사도 구간에 포함된 클래스 개수					
	Tamada		k-gram		API 트레이스	
	원본	난독화	원본	난독화	원본	난독화
$0.0 \leq C_{sim} < 0.1$	0	0	4	10	7	10
$0.1 \leq C_{sim} < 0.2$	0	0	12	12	4	3
$0.2 \leq C_{sim} < 0.3$	11	12	6	1	0	1
$0.3 \leq C_{sim} < 0.4$	4	9	0	1	3	2
$0.4 \leq C_{sim} < 0.5$	7	3	1	3	1	1
$0.5 \leq C_{sim} < 0.6$	0	2	0	7	5	3
$0.6 \leq C_{sim} < 0.7$	2	3	1	2	2	2
$0.7 \leq C_{sim} < 0.8$	2	9	3	1	2	2
$0.8 \leq C_{sim} < 0.9$	2	0	2	1	2	2
$0.9 \leq C_{sim} < 1.0$	1	0	3	0	1	1
$C_{sim} = 1.0$	9	0	6	0	11	11

표 6은 난독화된 Crimson-smoke와 Piccolo의 유사도 분포 결과이다. 난독화 이후에는 Tamada와 k-gram 버스마크는 100% 일치하는 클래스를 하나도 찾지 못했다. 그러나 API 트레이스는 난독화 전과 마찬가지로 100%일치하는 클래스를 11개 찾았다. 실제로 Crimson과 Piccolo에서 사용한 xml 공통 라이브러리의 개수는 총 16개인데 두 패키지가 서로 다른 버전의 라이브러리를 사용하였기 때문에 API 트레이스 버스마크가 16개를 모두 찾아낼 수는 없었다.

7. 결론 및 향후 연구 과제

소프트웨어 버스마크는 프로그램을 식별하는데 사용될 수 있는 프로그램의 고유한 특징을 말한다. 본 논문에서는 정적 API 트레이스 정보에 기반을 둔 자바 버스마킹 기법을 제안하였다. 정적 API 트레이스 기반 버스마크의 특징은 다음과 같이 요약할 수 있다. API 트레이스를 생성할 때 메소드의 제어 흐름을 분석하기 때문에 난독화나 컴파일러 변경에도 강인성이 보장된다. API 트레이스를 비교할 때 시퀀스 정렬 기법 중 준전체 정렬 방법을 사용했기 때문에 서로 다른 프로그램을 구별하는 신뢰도를 높이면서 트레이스의 전단부와 후단부에 코드를 추가하는 것에 대한 고려를 하였다.

본 논문에서 제안한 버스마킹 기법을 평가하기 위해서 XML Parser 패키지 4개를 비교한 결과 2개가 서로 같은 공개 라이브러리를 포함하고 있는 것을 알아낼 수 있었다. 패키지를 난독화하여 실험해본 결과 다른 정적 버스마크 기법들이 탐지하지 못하였으나 정적 API 트레이스는 이를 탐지하였다.

향후 연구 과제는 다음과 같다. 자바 API 함수에 가중치를 주어서 신뢰도를 더욱 높이는 방법을 고려하고 있으며, 기존의 동적 버스마킹 기법과의 비교 평가 및 다양한 자바 컴파일러와 난독화 도구에 대한 강인도 실험을 계획하고 있다.

참 고 문 헌

[1] Business Software Allience, "Fourth Annual BSA and IDC Global Software Piracy Study," 2006. http://www.bsa.org/globalstudy.

[2] L. Prechelft, G. Malpohl, and M. Philippsen, "Finding plagiarisms among a set of programs with jPlag," Journal of Universal Computer Science, vol. 8, no. 11, pp. 1016-1038, 2002.

[3] 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.

[4] 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.

[5] Aho, Sethi, Ullman, Compilers: Principles, Techniques, and Tools, Addison-Wesley, 1986.

[6] Brudno M, Malde S, Poliakov A, Do CB, Couronne O, Dubchak I, Batzoglou S. Glocal alignment: finding rearrangements during alignment. Bioinformatics Vol.19 Suppl.1:i54-62, 2003.

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

[8] "Jikes Java Compiler" http://jikes.sourceforge.net/.