KISM Smart Media Journal
Vol. 4, No. 2 pp. 34~45 ISSN:2287-1322

API 기반 동적 버스마크를 이용한 윈도우용 소프트웨어의 효율적인 유사도 측정 기법

Efficient Similarity Measurement Technique of Windows Software using Dynamic Birthmark based on API

박대신*, 지현호*, 박영수*, 홍지만**
Daeshin Park(parkpmon@gmail.com), Hyunho Jie(hyunho0804@gmail.com),
Youngsu Park(ccod85@gmail.com), JiMan Hong(jiman@ssu.ac.kr)

요 약

윈도우는 국내에서 가장 많이 사용되는 운영체제이기 때문에 윈도우용 소프트웨어를 대상으로 불법 복제가 많이 이루어지고 있고 불법 복제로 인해 소프트웨어 저작권이 침해될 수 있다. 이를 보호하기 위해서 저작권 보호 방법 중 하나인 소프트웨어 버스마크를 사용한다. 소프트웨어 버스마크는 소프트웨어로부터 특징 정보들을 추출하여 소프트웨어간 도용 여부를 판별할 수 있는 기술이며 대상 소프트웨어로 부터 특징 정보를 추출하는 방법에 따라 정적 버스마크와 동적 버스마크로 구별된다. 정적 버스마크와 동적 버스마크는 서로 장단점을 가지고 있지만 본 논문에서는 API 기반 동적 버스마크를 이용한 유사도 측정 기법을 제안하고, 동적 버스마크의 추출 과정을 설명한다. 또한 실험을 통해 제안하는 동적 버스마크의 유사도 측정 기법이 신뢰성과 강인성을 만족하는 것을 확인할 수 있었으며 기존 동적 버스마크의 유사도 측정 기법보다 제안하는 동적 버스마크의 성능이 향상 된 것을 확인할 수 있었다.

■ 중심어 : 소프트웨어 버스마크 ; 유사도 측정 ; 소스 코드 도용 ;

Abstract

The illegal copy of Windows software is one of the problems, because Windows is the most popular operating system in the country. The illegal copy can be infringe a software copyright, and software birthmark is one of solutions which is protecting software copyright. Software birthmark is a technique to distinguish software piracy using feature information from software. The type of software birthmark can be differentiated between static birthmark and dynamic birthmark through an extraction method. Static birthmark and dynamic birthmark have strengths and weaknesses. In this paper, we propose similarity measurement technique using dynamic birthmark based on API, and we explain extraction process of dynamic birthmark. In addition, we have verified that the proposed similarity measurement technique meet resilience and credibility through experiment. Furthermore, we saw that proposed measurement technique better than existing measurement technique.

■ keywords : Software Birthmark ; Similarity Measurement ; Source Code Piracy ;

* 학생회원, 숭실대학교 컴퓨터학과
** 정회원, 숭실대학교 컴퓨터학부
이 논문은 문화체육관광부 및 한국저작권위원회의 2014년도 저작권 기술개발사업의 연구결과로 수행되었음.

접수일자 : 2014년 12월 23일
수정일자 : 2015년 06월 27일
게재확정일 : 2015년 06월 29일
교신저자 : 홍지만 e-mail : jiman@ssu.ac.kr

Ⅰ. 서 론

국내 운영체제 시장에서 마이크로소프트사의 윈도우가 가장 높은 점유율을 보유하고 있으며, 또한 소프트웨어 시장의 대부분을 윈도우용 소프트웨어가 차지하고 있다. 이 때문에 윈도우용 소프트웨어의 불법 복제는 소프트웨어 시장에서 심각한 문제로 부각 되고 있다. 소프트웨어 불법 복제[1]는 크게 두 가지로 나눠질 수 있다. 첫 번째는 정상적으로 구매하지 않은 비허가 사용자가 상용 소프트웨어를 불법적으로 사용하는 것이고, 두 번째는 기업에서 소프트웨어의 소스 코드를 도용하는 것이다. 이 중 소프트웨어 소스 코드의 도용은 소프트웨어 개발 회사에 금전적 피해와 이미지 훼손 등의 심각한 손실을 입힌다.

이러한 피해를 줄이고 소프트웨어 저작권을 보호하기 위해 다양한 분야에서 저작권 보호 기술들이 연구되어 왔다. 대표적인 저작권 보호 기술로 워터마크(Water mark)[2]와 핑거프린팅(Finger Printing)[3], 소프트웨어 버스마크(Software Birthmark)가 있다. 워터마크와 핑거프린팅은 소프트웨어에 저작권자 정보 또는 구매자 정보를 포함하는 특징 정보를 삽입하여 소프트웨어의 불법 유통을 검출하고 불법 복제 행위를 억제하는 저작권 보호 방법이다. 소프트웨어 버스마크는 임의의 변형이나 조작 없이 소프트웨어 자체의 특징을 추출하여 도용을 판별하는 저작권 보호 방법이다.

워터마크와 핑거프린팅은 특징 정보를 통해 소프트웨어의 저작권자나 소유자를 식별할 수 있기 때문에 소프트웨어의 불법적인 사용을 판단하고 검출하는데 효과적이지만 소프트웨어의 유사도를 측정하는데 사용하기에는 적합하지 않다. 소프트웨어 버스마크는 소프트웨어의 고유한 특징을 추출하여 특징 정보로 사용하기 때문에 두 소프트웨어간의 유사도를 측정하기에 적합하다. 소프트웨어 버스마크는 소스코드 자체에서 특징을 추출하는 정적 버스마크(Static Birthmark)와 소프트웨어의 실행하는 동안 발생하는 함수 호출이나, 메모리의 변화 등의 여러 가지의 특징들을 추출하는 동적버스마크(Dynamic Birthmark)가 있다.

소프트웨어의 소스코드 도용을 밝히기 위해선 유사도의 측정이 필요로 하지만, 소프트웨어 기업의 핵심 자산인 소스코드를 가져와 직접 비교할 수 있는 법제적 장치가 없기 때문에 소스코드 없이 유사도를 비교해야 한다. 그렇기 때문에 본 논문에서는 소스코드 없이 유사도 측정이 가능한 소프트웨어 동적 버스마크를 이용하여 소프트웨어간의 유사도를 측정한다.

2장에서는 버스마크와 관련된 기존 연구들과 유사도를 측정하기 위한 기법들에 대하여 소개하고, 3장에서는 API 기반 동적 버스마크의 추출 방법과 제안하는 유사도 측정 기법에 대해 설명한다. 4장에서는 제안하는 유사도 측정 기법에서 사용하는 그룹핑 값과 제안하는 유사도 측정 기법이 신뢰성과 강인성을 만족하는지 실험을 통해 평가한다. 마지막으로 5장에서는 결론 및 향후 연구로 끝을 맺는다.

Ⅱ. 관련 연구

1. 소프트웨어 버스마크

소프트웨어 버스마크는 유사도 비교를 위한 특징 정보로 사용된다. 특징 정보를 얻기 위해 외부적인 요소를 이용하는 것이 아니라 소프트웨어 자체를 분석한 정보들을 바탕으로 특징 정보를 추출한다. 이 특징 정보를 추출하는 방법과 시점에 따라 정적 버스마크와 동적 버스마크로 나뉘게 된다.

정적 버스마크는 대상 소프트웨어의 실행 없이 소스코드 혹은 바이너리 파일로부터 얻어지는 정보를 이용하여 버스마크를 추출하는 방법이다. 정적 버스마크를 추출하는 방법에는 소프트웨어 소스코드, 바이너리 파일이 갖는 문법적인 구조 그리고 바이너리의 구성을 분석하는 방법이 있다.

동적 버스마크는 소프트웨어의 실행을 통해 추출되는 정보를 이용하여 버스마크를 추출하는 방법이다. 실행환경이나 실행시의 입력 값에 따라 버스마크의 내용이 변할 수 있다는 단점을 갖고 있지만, 정적 버스마크가 갖는 단점인 코드 난독화에 대한 강인성과 분석 결과의 정확성이 상대적으로 높다는 장점을 갖는다. 동적 버스마크에서 특징 정보로 사용가능 한 요소에는 윈도우 API의 호출이나 메모리의 상태변화, 함수 호출 콜 그래프 등이 있다.

가. 동적 버스마크의 기존 연구

소프트웨어 동적 버스마크를 이용한 대표적인 연구에는 Tamada가 제안한 동적 윈도우 API 버스마크가 있다[4]. 이 연구에서는 윈도우용 소프트웨어들로부터 API 호출을 추출하여 호출 빈도와 호출 순서를 특징 정보로 하는 동적 버스마크로 제안하였다. Tamada가 정적 버

스마크를 이용하여 진행한 이전 연구와는 달리 동적 버스마크를 이용한다는 점에서 코드 난독화나 컴파일러 최적화에 영향을 받지 않는다[5]. 또한 연구에서 사용된 윈도우 API는 윈도우에서 동작하는 소프트웨어가 필수로 사용하는 함수들로 다른 함수로 대체하거나 함수 호출 순서를 바꾸기 어렵다는 특징이 있다. 이 연구는 환경적 요소에 따라 API 호출 순서가 달라지거나 노이즈가 발생할 수 있다는 것을 고려하지 못한 한계를 가지고 있다.

나. 동적 버스마크의 정의와 속성

동적 버스마크는 소프트웨어의 실행 중 추출되는 정보를 이용하여 특징 정보를 얻기 때문에 실행 환경과 입력 값에 큰 영향을 받는다. 동적 버스마크의 정의와 두 가지 속성인 강인성(Resilience)과 신뢰성(Credibility)은 다음과 같다.

정의 1 (버스마크) 소프트웨어 p에서 특징 정보를 추출하는 방법을 f라 하고, 특징 정보를 추출하기 위한 입력 값을 I라 할 때, 다음 두 가지 조건을 만족시키는 $f(p, I)$를 p의 동적 버스마크라 정의한다.

1. $f(p, I)$는 소프트웨어 p에 입력 값 I로만 얻어진다.
2. p가 q를 복제 했다면 $f(p, I) = f(q, I)$ 이다.

속성 1 (강인성) 소프트웨어 p'이 p의 변형으로 얻어진 소프트웨어라면, $f(p, I) = f(p', I)$가 성립한다.

속성 2 (신뢰성) 소프트웨어 p, q가 같은 기능을 갖더라도 서로 독립적으로 개발 되었다면, $f(p, I) \neq f(q, I)$가 성립한다.

강인성은 코드 난독화나 컴파일러 최적화 등의 기술들을 사용하여 소스코드를 변형하더라도 도용을 판별할 수 있어야 한다는 속성을 나타낸다. 신뢰성은 두 개의 소프트웨어가 동일한 기능을 하더라도 독립적으로 제작된 소프트웨어라면 다른 버스마크가 추출되어야 한다는 속성을 나타낸다[3].

2. 소프트웨어 버스마크의 유사도 측정 기법

소프트웨어 버스마크의 유사도를 측정하기 위한 방법들 중 대표적인 방법으로 K-gram 유사도 측정기법과 Cosine(Cosine) 유사도 측정기법, Diff 알고리즘을 이용한 유사도 측정 기법이 있다.

가. K-gram 유사도 측정기법

K-gram은 소프트웨어의 명령어를 k 개의 연속된 명령어 집합들로 추출하여 얻는 특징 정보이다. 두 소프트웨어로부터 얻은 K-gram의 동일한 정도를 계산하여 유사도를 측정한다.

소프트웨어 r과 s로부터 추출한 K-gram을 $f(r)$, $f(s)$라 할 때, 두 소프트웨어의 유사도 $Similarity(r, s)$는 다음과 같이 구할 수 있다.

$$Similarity(r, s) = \frac{|f(r) \cap f(s)|}{\min(|f(r)|, |f(s)|)} \quad \text{[수식 1]}$$

[수식]에서 $|f(r)|$은 $f(r)$에서 중복되지 않는 원소 개수를 의미한다. 두 소프트웨어 간의 유사성을 정량화하기 위해 전체 K-gram의 개수 중에서 동일한 K-gram의 개수가 차지하는 비율을 측정하면 그 비율이 유사도로 표현된다.

나. Cosine 유사도 측정 기법

Cosine 유사도 측정기법은 다양한 분야에 적용되는 기법으로 [수식 2-2]와 같이 내적공간의 두 벡터 간 사이각의 Cosine 값을 이용하여 유사도를 구한다[7].

$$\begin{aligned} Similarity(x, y) &= \cos(\theta) \\ &= \frac{x \cdot y}{\|x\| \times \|y\|} \\ &= \frac{\sum_{i=1}^{n} x_i \times y_i}{\sqrt{\sum_{i=1}^{n} (x_i)^2} \times \sqrt{\sum_{i=1}^{n} (y_i)^2}} \end{aligned} \quad \text{[수식 2]}$$

이때 $Similarity(x, y)$는 단위가 1인 벡터 x와 y 사이의 Cosine 값에 해당한다. 이렇게 계산된 유사도는 -1에서 1까지의 값을 가지며, x와 y가 유사한 경우 Cosine 값이 1에 가까워지고, 유사하지 않을 경우 Cosine 값이 0에 가까워진다.

3. 기존 연구의 한계점

기존 연구에서는 버스마크의 추출을 위해 함수의 호출 빈도나 호출 순서를 이용한다. 호출 빈도를 이용한 버스마크는 함수의 호출 횟수를 특징 정보로 사용한다. 비슷한 기능의 두 소프트웨어로부터 버스마크를 추출했을 때 두 소프트웨어는 비슷한 함수를 사용하기 때문에 높은 유사도가 나타날 수가 있다. 호출 순서를 이용

한 버스마크는 함수의 호출한 순서를 특징 정보로 사용하기 때문에 소프트웨어의 특징을 잘 표현한다. 하지만 노이즈나 순서 재배치에 취약하다는 문제점을 갖는다.

동적 버스마크는 운영체제의 스케줄링, 멀티 스레드, GUI 노이즈 등으로 인해 같은 소프트웨어라도 다른 버스마크가 생성되는 특징이 있다. 이러한 특징은 호출 빈도를 이용한 버스마크에는 큰 영향을 주지 않지만, 호출 순서를 이용한 버스마크에서는 큰 차이를 나타낸다. 하지만 기존 연구에서는 대부분 이러한 특징을 고려하지 않아 동일한 소프트웨어도 다른 소프트웨어로 오판하는 경우가 발생한다.

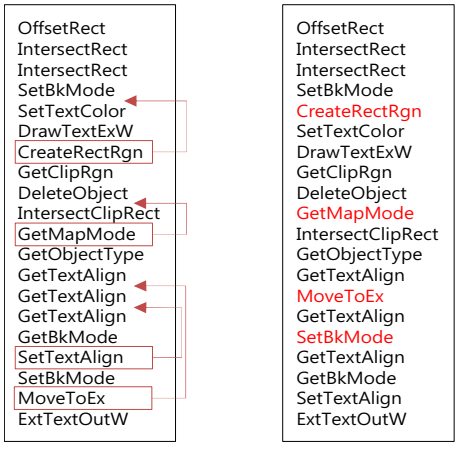

그림 1 원본 데이터와 변형된 데이터

[그림 1]은 동일한 소프트웨어를 실행 할 때 같은 입력 값을 사용하더라도 운영체제의 스케줄링 또는 멀티 스레드로 인해 순서의 재배치가 일어날 수 있음을 보여준다. 4개 함수의 호출 순서가 바뀌었을 뿐이지만 K-gram 유사도 측정기법으로 측정 시 낮은 유사도가 측정되어 다른 소프트웨어로 오판할 수 있다.

표 1. 동일 소프트웨어 및 다른 소프트웨어의 k-gram과 Cosine 유사도

소프트웨어	Cosine 유사도	K-gram 유사도
Note++6.5.2	99.84%	67.64%
Filezilla 3.6.0	99.94%	58.63%
7Zip 4.65 vs BreadZip 4.0	87.64%	50.82%
Akelpad 4.8.0 vs Notepad 6.1	99.84%	39.28%
Pot Player 1.5.35491 vs Window Media Player 12.0.7601.17514	99.53%	32.50%

[표 1]은 동적으로 추출한 버스마크를 K-gram 유사도 측정 기법과 Cosine 유사도 측정 기법을 이용하여 유사도를 비교한 표이다. K-gram을 이용하여 유사도를 측정한 경우 전체적으로 낮은 유사도가 나타났으며 동일한 소프트웨어에서도 낮은 유사도 값을 나타낸다. 또한 Cosine을 이용한 경우에는 전체적으로 높은 유사도가 보이며 유사한 기능을 제공하지만 다른 소프트웨어에서도 높은 유사도 값이 측정된다.

이러한 실험 결과에서 보이듯이 기존의 호출 빈도나 호출 순서 하나만을 이용하여 동적버스마크에서 유사도를 측정할 경우 유사도 값이 상이하게 측정 될 수 있다. 본 논문에서는 호출 빈도와 호출 순서를 동시에 사용하여 동적 버스마크를 사용할 때 발생할 수 있는 문제점을 보완한 API 기반 버스마크를 제안하고자 한다.

Ⅲ. API 기반 동적 버스마크

본 절에서는 API 기반 동적 버스마크의 추출 방법과 기존 연구의 한계를 보완하기 위한 유사도 비교 기법에 대해 설명한다.

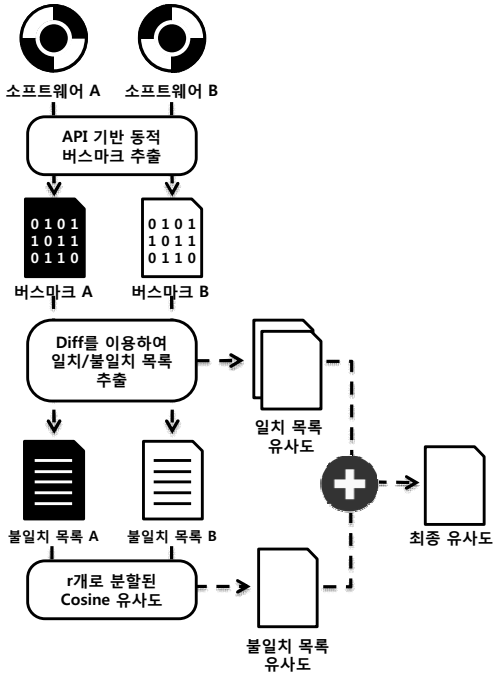

그림 2 API 기반 동적 버스마크 추출 및 유사도 비교

[그림 2]은 API 기반 동적 버스마크 추출 방법과 본 논문에서 제안하는 API 기반 동적 버스마크의 유사도

비교 기법을 도식화 한것이다. 윈도우용 소프트웨어 A와 B를 동일한 기능만 실행하는 동안 API 기반 동적 버스마크를 추출한다. 추출된 API 기반 동적 버스마크를 Diff를 이용하여 일치 목록과 불일치 목록으로 구별한다. 그후 일치 목록과 불일치 목록의 유사도를 측정한다. 마지막으로 일치 목록의 유사도와 불일치 목록의 유사도를 더하여 최종 유사도를 구한다.

1. API 기반 동적 버스마크 추출

본 논문에서 사용하는 동적 버스마크는 윈도우용 소프트웨어를 실행한 상태에서 추출한 정보를 이용하여 동적 버스마크를 추출한다. 윈도우 API는 윈도우에서 동작하는 소프트웨어들이 필수로 사용해야 하는 함수들로써, API 함수를 다른 함수로 대체하거나 함수의 호출 순서를 바꾸는 것은 어려운 작업이다. 그러므로 본 논문에서는 윈도우 API를 소프트웨어의 중요한 특징 정보로 판단하고, 이 특징 정보를 사용하여 버스마크를 추출한다. 다음은 입력 값 I와 함수 w의 정의이다.

정의 (입력 값 I) 소프트웨어가 사용자의 원하는 기능을 수행하기 위하여 소프트웨어에 입력되는 숫자, 문자 또는 마우스 움직임 등의 모든 입력 값을 뜻한다.

정의 (함수 w) 함수는 여러 명령어들로 구성되어 있고, 소프트웨어를 구성한다.

MS Detours Library는 Microsoft사에서 제공하는 Hooking Library이며, MS Detours Library를 사용하여 실행 중인 Windows 바이너리 파일에서 호출되는 API 정보들을 추출 할 수 있다. 다음은 API 추출 집합 W의 정의이다.

정의 (API 추출 집합 W) Windows OS에서 제공하는 전체 API 중 MS Detours library에서 추출 가능한 1,662개의 API 집합이다[8].

API 추출 집합 W는 MSDN에 명시된 2667개의 API 함수들을 전부 포함하지 않는다. Detours Library에서 추출할 수 있는 API 함수의 개수는 1662개 이지만, Windows 소프트웨어가 필수로 사용하는 네이티브 API와 주로 사용하는 User32.dll, GDI32.dll, KERNEL32.dll, WS2_32.dll 등의 라이브러리의 API도 포함되어 있다. 따라서 MSDN에서 명시하는 모든 API를 포함하지 않더라도 유효한 동적 버스마크를 추출 할 수 있다.

```
CoInitialize
 CoInitializeEx
    CoCreateGuid
    GetCurrentThreadId
    GetCurrentThreadId
    CoCreateGuid
    CallNextHookEx
    IsWindow
    IsWindow
    CallNextHookEx
    IsWindow
    CallNextHookEx
    GetCurrentThreadId
    GetCurrentThreadId
GetParent
```

그림 3 API 추출 데이터 예

[그림 3]은 MS Detours Library를 통해 추출된 API 호출 순서의 일부분이며, 그림에서 들여쓰기 된 API 함수는 다른 API 함수에 의해 다시 호출된 API 함수를 뜻한다. 예를 들어, CoCreateGuid의 호출 과정을 설명하면 CoCreateGuid 함수는 CoInitializeEx 함수 내부에서 호출되었으며, CoInitializeEx 함수는 CoInitialize 함수 내부에서 호출된 함수이다. 이와 같이 함수 내부에서 다른 함수를 호출하는 과정까지 모두 추출하여 특징 정보로 사용하며, 어떤 함수에 의해 함수가 호출 되었는지에 따라 함수가 호출되는 의미가 달라지는 경우가 발생할 수 있다. 다음은 API 기반 동적 버스마크의 정의 이다.

정의 (API 기반 동적 버스마크) 소프트웨어 P에 I를 입력하여 추출된 함수 호출 순서가 $(P, I) = (w_1, w_2, ..., w_n)$ 일 때, W에 포함되지 않는 함수 w들을 제거하고 남은 데이터를 API 기반 동적 버스마크 $DYSEQ(P, I)$라 한다.

2. API 기반 동적 버스마크의 유사도 비교

본 논문에서 제안하는 유사도 비교 과정은 크게 네 가지로 나눠진다. 첫 번째 단계는 Diff 알고리즘을 이용하여 동적 버스마크를 일치 목록과 불일치 목록으로 나눈다. 두 번째는 일치목록의 유사도를 계산하고, 세 번째는 불일치 목록의 유사도를 계산한다. 마지막으로 두 목록을 통해 계산된 유사도 값을 합하여 최종 유사도 값을 구한다.

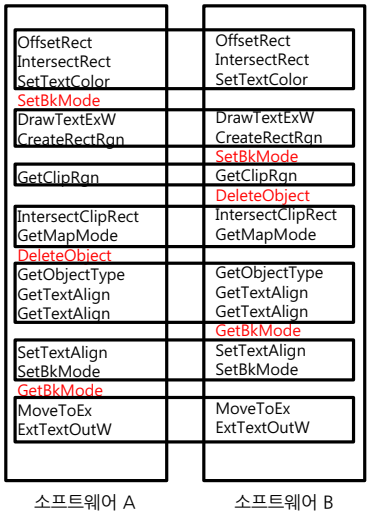

그림 4 API 기반 버스마크에 Diff 알고리즘 적용 예

[그림 4]와 같이 소프트웨어에서 추출한 API 기반 동적 버스마크를 Diff 알고리즘을 사용하여 일치하는 목록과 불일치하는 목록으로 구별한다. 다음은 일치성분 s와 일치목록 S 정의이다.

정의 (일치성분 s) 소프트웨어 P와 P'에서 추출한 동적 버스마크를 Diff 알고리즘으로 비교하여 일치하지 않는 부분을 제거한 나머지 원소를 일치성분 s라 정의한다.

정의 (일치목록 S) 소프트웨어 P와 P'에 대해 일치성분들로만 구성된 목록을 일치목록 $S_{P'}^{P} = (s_1, s_2, ..., s_x)$로 정의한다.

[그림 4]에서 일치목록은 $S_{P'}^{P}$ =(OffsetRect, IntersectRect, SetTextColor, ... , MoveToEx, ExtTextOutW)가 된다. 다음은 불일치성분 d와 불일치목록D의 정의이다.

정의 (불일치성분 d) 소프트웨어 P와 P'에서 추출한 동적 버스마크를 Diff 알고리즘으로 비교하여 일치하지 않는 부분들의 원소 중 소프트웨어 P에 대한 불일치 성분을 d, P'에 대한 불일치 성분을 d'라 정의 한다.

정의 (불일치 목록 D) 소프트웨어 P와 P'에 대해 불일치성분들로만 구성된 목록 $D(P) = (d_1, d_2, ..., d_x)$와 $D(P') = (d'_1, d'_2, ..., d'_y)$를 불일치 목록이라 정의한다.

[그림 4] 에서 불일치 목록은 $D(P)$ =(SetBkMode, DeleteObject, GetBkMode)와 $D(P')$ =(SetBkMode, DeleteObject, GetBkMode)이다. 다음은 그룹핑된 불일치 목록 D_i의 정의이다.

정의 (그룹핑된 불일치목록 D_i) 소프트웨어 P의 불일치 목록 $D(P) = (d_1, d_2, ..., d_x)$과 소프트웨어 P'의 불일치 목록 $D(P') = (d'_1, d'_2, ..., d'_y)$이 있고, 그룹핑 값 r이 임의의 값일 때 $i = x \div r$이며 $r' = y \div i$이다. 이렇게 구해진 i값으로 그룹핑된 불일치 목록은 $D_i(P) = ((d_1, d_2, ..., d_r)_1,$ $..., (d_{r+1}, d_{r+2}, ..., d_x)_i)$와 $D_i(P') = ((d'_1, d'_2, ..., d'_{r'})_1, ..., (d'_{r'+1}, d'_{r'+2}, ..., d'_y)_i)$로 정의된다. (단 $y > x$)

불일치 목록을 그룹핑 하는 이유는 스레드나 스케줄러에 의해 함수의 호출 순서가 변경될 경우 위치가 변경된 함수의 호출은 비교군의 불일치목록에서 비슷한 위치에서 호출될 가능성이 높기 때문이다. 본 논문에서는 그룹핑 값 r의 값을 실험을 통해 정의하고, 비교 횟수를 동일하게 만들기 위하여 비교군의 r'값을 $r' = y \div i$으로 구한다. 또한, x또는 y의 값이 r값 보다 작을 경우 불일치 목록을 그룹핑을 하지 않고 유사도를 측정한다.

일치 목록은 두 소프트웨어를 비교해서 정확히 함수 호출이 일치 하는 부분을 추출한 것이다. 그래서 일치목록의 유사도는 1이 되고 실제 유사도 값은 두 소프트웨어의 API 호출 순서를 이용한 동적 버스마크의 원소 개수에서 일치목록이 차지하는 비율이 두 소프트웨어의 일치목록 유사도 값이 된다. 다음과 같은 방식으로 일치 목록의 유사도 값이 측정된다. 소프트웨어 P와 P'의 일치 목록이 $S_{P'}^{P}$일 때, 일치목록 유사도는

$$SIM(S_{P'}^{P}) = \frac{n(S_{P'}^{P}) \times 2}{n(P) + n(P')}$$

과 같은 수식을 통해 측정된다. 여기서 $n()$은 원소의 개수이다.

불일치목록의 유사도 측정 방법은 두 소프트웨어의 불일치 목록을 그룹핑하고, 그룹핑 된 불일치 목록끼리 Cosine 유사도를 이용하여 측정한다. 두 소프트웨어의 불일치 목록이 r값으로 나누어 떨어지지 않을 수가 있으며, 그럴 경우 남는 원소들을 하나의 그룹으로 만들어 유사도를 측정하게 된다. 각각의 그룹핑된 불일치 목록의 유사도 값을 모두 합산한 값이 전체 불일치 목록의 유사도 값이 된다. 다음과 같은 방식으로 불일치 목록의 유사도 값이 측정된다. 소프트웨어 P의 불일치목록 $D_i(P) = ((d_1, d_2, ..., d_r)_1, ..., (d_{r+1}, d_{r+2}, ..., d_x)_i)$

, $D_i(P)_1 = (d_1, d_2, ..., d_r)$일 때, $D_i(P)_1$의 벡터집합은 $\overrightarrow{D_i(P)_1} = ((d_1, k), (d_2, j), ..., (d_r, l))$이다. 여기서 k, j, l은 $D_i(P)_1$에서 각 원소의 출현 빈도이다. 그리고 소프트웨어 P와 P'의 그룹핑된 불일치 목록이 각각 $D_i(P)$, $D_i(P')$일 때, 불일치 목록의 유사도는 다음과 같은 식을 사용하여 측정된다. (단, $n(D(P')) > n(D(P))$)

$$SIM(D(P), D(P')) = \sum_{x=1}^{i} \left(\frac{r+r'}{n(P)+n(P')} \times \cos(\overrightarrow{D_i(P)_x}, \overrightarrow{D_i(P')_x})\right) + \left(\frac{(n(D(P)) - ((i-1)\times r)) + (n(D(P')) - ((i-1)\times r'))}{n(P)+n(P')} \times \cos(\overrightarrow{D_i(P)_i}, \overrightarrow{D_i(P')_i})\right)$$

마지막으로 일치목록의 유사도 값과 불일치 목록의 유사도 값을 합산한 결과가 최종적으로 두 소프트웨어의 유사도 값이 된다.

Ⅳ. 실험 및 성능 평가

1. 실험 방법 및 구성

API를 기반으로 추출한 동적 버스마크를 본 논문에서 제안하는 유사도 측정 기법으로 평가하기 위해 여러 가지 실험을 하고 실험 결과를 평가한다. 실험은 총 네 가지로 구성했으며, 첫번째 실험은 제안하는 유사도 측정 기법의 과정 중 불일치 목록의 유사도를 측정할 때 사용되는 최적의 그룹핑 값을 찾는 실험이다. 두번째 및 세번째 실험은 제안하는 버스마크의 유사도 측정 기법이 버스마크의 속성인 신뢰성과 강인성을 만족하는지를 평가한다. 마지막으로 제안하는 유사도 측정 기법과 기존 유사도 측정 기법을 비교하여 제안하는 유사도 측정 기법을 평가한다.

2. 최적의 그룹핑 값 찾기

본 실험은 불일치 목록을 그룹핑할 때 사용되는 r의 최적의 값을 찾기 위한 실험이다. 불일치 목록을 그룹핑하는 이유는 함수의 호출 순서가 스레드나 스케줄러에 의해 변경 되는 경우가 발생하기 때문이며, 함수 호출 순서가 변경되었지만 비슷한 위치에서 함수가 호출될 것 가능성이 높기 때문에 r값을 이용하여 불일치 목록을 그룹핑한다. 최적의 그룹핑 값 r을 찾기 위하여 세가지 값 500, 1000, 2000을 r값으로 선정했다. 실험 과정은 r값을 변경하면서 소프트웨어 간 유사도를 측정했다.

최적의 그룹핑값을 평가하기 위하여 선정된 윈도우용 소프트웨어로는 압축 프로그램, 문서편집기, FTP, 음악 재생 프로그램, 동영상 플레이어이다. 각 분류별로 동일한 소프트웨어의 다른 버전과 같은 기능을 하는 다른 소프트웨어를 선정하여 총 15개의 소프트웨어에서 동적 버스마크를 추출하였다. [표 2]는 최적의 그루핑값을 평가하기 위해 선정된 소프트웨어 이름이다.

표 2. 실험을 위해 선정된 소프트웨어 분류, 이름 및 약어

분류	소프트웨어 이름	약어
동영상 플레이어 (VPL)	MPC 1.7.4.8	VPL1
	MPC 1.7.3.94	VPL2
	Pot Player 1.6	VPL3
음악 재생 프로그램 (MPL)	Foobar 1.2.6	MPL1
	Foobar 1.2.5	MPL2
	AIMP 3.55	MPL3
FTP (FTP)	Filezilla 3.5.3	FTP1
	Filezilla 3.5.2	FTP2
	Aldrive 1.21	FTP3
문서 편집기 (EDI)	Akelpad 4.8.8	EDI1
	Akelpad 4.7.7	EDI2
	Note++ 6.5.2	EDI3
압축 프로그램 (CMP)	7zip 9.32	CMP1
	7zip 9.22	CMP2
	Breadzip 4.0	CMP3

동적 버스마크는 추출 할 때 수행하는 기능에 따라 추출되는 버스마크가 달라지기 때문에 각 소프트웨어의 분류에 맞춰 추출 과정 중 수행할 기능을 정의하고 동일한 기능만을 수행하여 버스마크를 추출하였다. 또한 각 분류별 소프트웨어 수행 과정에서 필요한 입력 파일은 동일한 파일을 사용했다.

표 3. 동적 버스마크를 추출과정에서 수행한 기능

분류	수행한 기능
압축 프로그램	실행 -> 파일 열기 -> 파일 추가 압축 -> 종료
문서편집기	실행 -> 문서 파일 열기 -> 내용 수정 -> 파일 저장 -> 종료
FTP	실행 -> FTP 서버 접속 -> 파일 업로드 -> 종료
음악 재생 프로그램	실행 -> 음악 파일 열기 -> 3초 재생 -> 정지 -> 종료
동영상 플레이어	실행 -> 동영상 파일 열기 -> 3초 재생 -> 정지 -> 종료

[표 3]은 각 분류별 동적 버스마크를 추출하는 과정에서 수행한 기능들을 정리한 표이다.

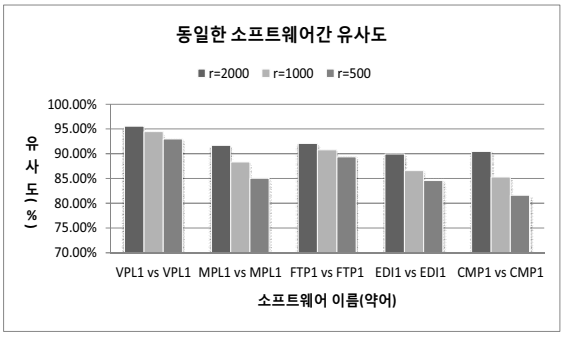

그림 5 r값에 따른 동일한 이름과 버전의 소프트웨어간 유사도

[그림 5]은 r값을 500, 1000, 2000으로 변경하고 동일한 버전의 소프트웨어를 비교한 유사도 측정 그래프이다. 이 실험은 동일한 버전의 소프트웨어를 비교하여 유사도를 측정하였기 때문에 유사도 값이 높게 나올수록 좋은 결과로 볼 수 있다. 실험 결과를 보면 전체적으로 r값이 클수록 유사도 측정 값이 높아지는 것을 볼 수 있으며, r값이 2000일 때 가장 높은 유사도 값이 측정 되었다.

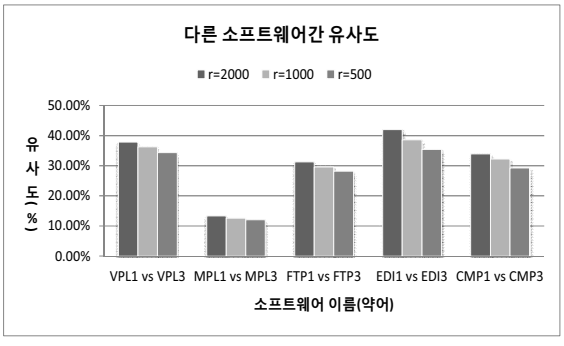

그림 6 각 r값에 따른 동일한 분류에서 다른 이름의 소프트웨어간 유사도

[그림 6]은 r값에 따라 같은 기능을 수행하지만 다른 소프트웨어를 비교한 유사도 측정 그래프이다. 이 실험에서는 동일한 기능을 수행하지만 다른 소프트웨어를 비교하여 유사도를 측정했기 때문에, 유사도 값이 낮을수록 좋은 결과로 볼 수 있다. 실험 결과를 보면 [그림 5]과 같이 r값이 클수록 유사도 측정 값이 높아지는 것을 볼 수 있다.

위 두 실험을 보면 r값이 2000일 때 동일한 소프트웨어의 유사도 측정값은 평균적으로 4.91% 상승하였으며, 다른 소프트웨어의 유사도 측정값 또한 3.06% 상승하였다. 또한 r이 2000일 때 동일한 소프트웨어의 유사도 측정값과 다른 소프트웨어 유사도 측정값이 모두 상승하였지만, 동일한 스프트웨어의 유사도 상승폭이 다른 소프트웨어의 유사도 상승폭보다 높기 때문에 최적의 그룹핑 값은 2000이다.

3. 신뢰성 평가

본 실험은 제안하는 버스마크의 유사도 측정 기법이 신뢰성을 만족하는지를 평가한다. 신뢰성은 두 소프트웨어가 같은 기능을 수행하지만 독립적으로 개발된 소프트웨어에 대해 버스마크가 두 소프트웨어를 다르다고 판단해야 한다는 것이다. 즉, 같은 기능을 하는 소프트웨어 두 개가 있을 때, 두 소프트웨어가 동일한 소스코드를 사용한다면, 두 소프트웨어의 유사도는 높아야 된다. 하지만 두 소프트웨어가 다른 소스코드를 사용한다면 두 소프트웨어의 유사도는 낮게 측정되어야 한다.

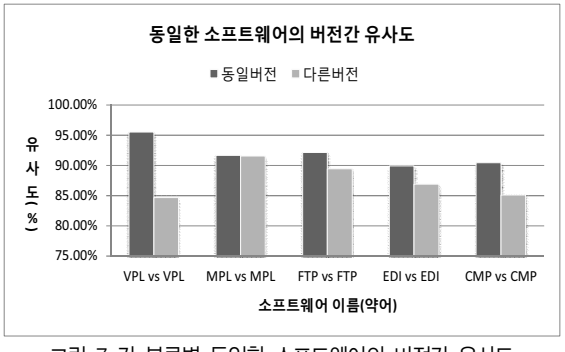

그림 7 각 분류별 동일한 소프트웨어의 버전간 유사도

[그림 7]은 각 분류별 동일한 소프트웨어의 같은 버전과 다른 버전의 유사도 측정 결과이다. 다른 버전은 동일한 소프트웨어에서 약간 업그레이드 된 버전으로 기존 소스코드의 대부분을 동일하게 사용할 가능성이 높다. 그 때문에 신뢰성이 높은 유사도 비교 기법을 사용했다면, 동일 버전의 유사도는 높게 측정되어야 하며, 다른 버전의 유사도는 동일 버전의 유사도보다 낮게 측정되어야 한다. 실험 결과는 다른 버전의 소프트웨어 유사도 측정 값은 동일 버전의 소프트웨어 유사도보다 낮은 유사도가 측정되었으며, 평균적으로 84%이상의 유사도가 측정되었다.

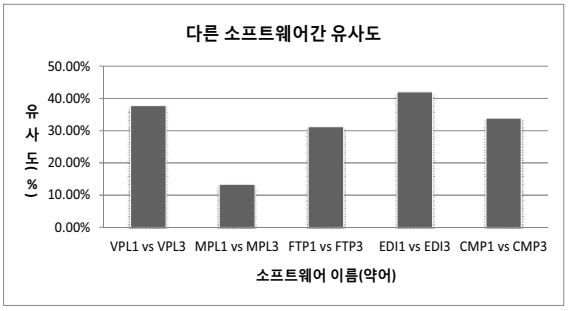

그림 8 각 분류별 다른 이름의 소프트웨어간 유사도

[그림 8]은 각 분류별 같은 기능을 수행하지만 독립적으로 구현된 소프트웨어간의 유사도 측정 결과이다. 신뢰성이 높은 유사도 측정 기법이라면 본 실험에서는 낮은 유사도가 측정되어야 한다. 실험 결과는 평균 30.13%의 유사도가 측정 되었으며, 가장 높은 유사도 값이 측정된 분류 중 하나는 동영상 플레이어이다. 그 이유는 동적 버스마크를 추출하는 과정에서 같은 동영상 파일을 짧은 시간 동안 재생하여 비슷한 함수가 호출 되었기 때문으로 분석된다.

신뢰성 실험 결과를 분석해 보면, 동일한 소프트웨어의 같은 버전 유사도는 평균 91.95%이고, 다른 버전 유사도는 평균 87.53%으로 두 가지 실험 모두 높은 유사도를 나타낸다. 다른 소프트웨어간 유사도 값은 평균 30.13%의 유사도를 나타냈다. 위 실험 결과를 분석해 볼 때 제안하는 유사도 측정 기법은 독립적으로 개발된 소프트웨어를 대한 구별을 할 수 있으며, 신뢰성을 갖고 있다는 것을 알 수 있다.

4. 강인성 평가

본 실험은 제안한 유사도 측정 기법이 강인성을 만족하는지를 평가한다. 강인성은 코드 도용 탐지를 회피하기 위해 난독화 기법이나 컴파일러 최적화등을 이용하여 원본 소스코드에 변형을 가한 경우라도 같은 소프트웨어를 판별 할 수 있어야 함을 의미한다.

표 4. 강인성 실험을 위한 실험군의 정보

약어	소프트웨어 이름	기존 컴파일러	새로운 컴파일러	컴파일 옵션
Dif_Comp 1	Akelpad 4.7.7	Microsoft Visual C++ 6.0	Microsoft Visual C++ 10.0	없음
Dif_Comp 2	Akelpad 4.7.7	Microsoft Visual C++ 6.0	Intel C++ Compiler XE 14.0	없음
Dif_Comp 3	Filezilla 3.5.2	Microsoft Visual C++ 12.0	MinGW	없음
Dif_Opt 1	Akelpad 4.7.7	Microsoft Visual C++ 6.0	Microsoft Visual C++ 10.0	코드 크기 최적화
Dif_Opt 2	Akelpad 4.7.7	Microsoft Visual C++ 6.0	Microsoft Visual C++ 10.0	코드 속도 최적화

[표 4]는 강인성 실험을 위해 사용한 컴파일러 및 컴파일러 옵션 정보를 정리한 표이다. 실험은 원본 소프트웨어를 컴파일러 또는 컴파일 옵션을 변경하여 컴파일된 소프트웨어와 컴파일러에 의해 변형되지 않은 원본 소프트웨어를 비교하여 유사도를 측정했다. 강인성이 있는 유사도 측정 기법이라면, 컴파일러에 의해 소프트웨어가 변경되었더라도 같은 소프트웨어이기 때문에 유사도가 높게 측정되어야 한다.

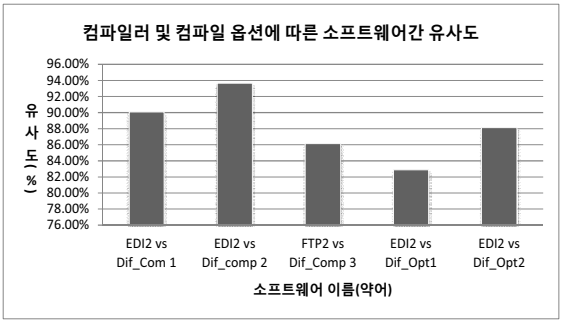

그림 9 컴파일러 및 컴파일로 옵션 변경에 따른 소프트웨어간 유사도

강인성 실험은 신뢰도 실험과 동일한 기능들을 수행하여 동적 버스마크를 추출했다. [그림 9]은 강인성 실험 결과를 정리한 그래프이다. 컴파일러나 컴파일러 옵션을 변경한 소프트웨어와 원본 소프트웨어를 비교했을 때, 컴파일러나 컴파일러 옵션에 따라 유사도가 차이가 있지만 82% 이상의 유사도가 측정되었다.

컴파일러 옵션에 따라 최적화된 코드는 기존의 함수가 다른 함수로 변경되었거나 여러 개의 함수를 하나의 함수로 합쳐지게 된다. 그 때문에 컴파일러 최적화 옵션이 적용된 소프트웨어에서 가장 낮은 유사도가 측정됨을 확인할 수 있다. 하지만 82%이상의 유사도가 측정되었기 때문에 두 소프트웨어는 유사하다고 할 수 있다. 본 실험을 통해 제안한 유사도 측정 기법이 컴파일러 및 컴파일 옵션이 변경되더라도 평균적으로 88.16%의

유사도를 측정되기 때문에, 소스 코드 변경에 대한 강인성을 만족하는 것을 알 수 있다.

5. 기존 유사도 측정 기법과의 성능 비교

본 실험은 기존 버스마크보다 제안하는 유사도 측정 기법이 사용하기 적합한지를 실험을 통해 확인한다. 실험은 [표 2]에서 명시된 소프트웨어를 대상으로 하였으며, 핵심 기능도 [표 3]과 동일하게 수행하였다.

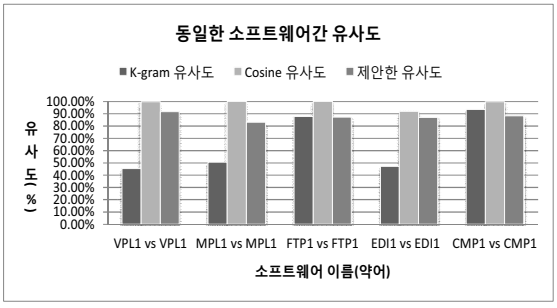

그림 10 기존 유사도 측정 기법 간의 동일한 소프트웨어 유사도

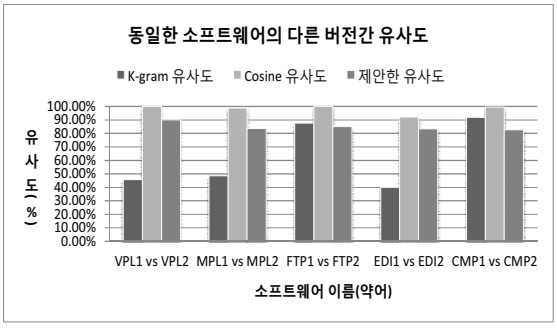

그림 11 기존 유사도 측정 기법 간의 다른 버전 소프트웨어 유사도

[그림 10]과 [그림 11]은 3가지 유사도 측정 기법을 이용하여 같은 소프트웨어 간의 유사도를 측정한 그래프이다. 실험 결과 Cosine 유사도는 2가지 실험에서 대부분 90%이상의 유사도가 나타났다. 하지만 K-gram 유사도는 동일한 소프트웨어 임에도 불구하고 FTP와 압축 프로그램을 제외한 나머지 분류에서는 유사도가 낮게 측정됐다. 그 이유는 동적 버스마크를 추출할 때 멀티스레드나 GUI 노이즈에 의한 영향을 받아 K-gram 유사도 측정 기법을 이용하여 유사도를 측정하면 유사도가 낮게 측정되는 것으로 분석된다. 제안한 유사도 측정 기법은 Cosine 유사도에 비해 유사도 측정 값이 낮지만 K-gram 유사도 기법 보다는 높은 82% 이상의 유사도를 보여준다.

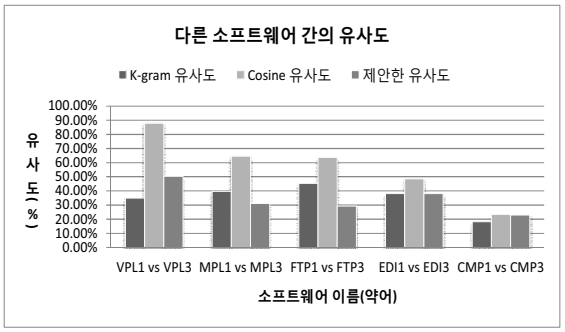

그림 12 기존 유사도 측정 기법 간의 동일 분류에서 다른 소프트웨어 유사도

[그림 12]는 기존 유사도 측정 기법으로 같은 기능을 제공하지만 다른 소프트웨어를 비교하여 유사도를 측정한 그래프이다. 이 실험에서는 Cosine 유사도가 K-gram 유사도와 제안한 유사도 기법에 비해 유사도가 높게 측정되었으며, 동영상 재생 프로그램은 87%의 높은 유사도가 측정되어 동일한 소프트웨어로 오판했다. K-gram 유사도는 Cosine 유사도에 비해 다른 소프트웨어를 잘 구별했지만, 동일한 소프트웨어는 잘 구별하지 못했다. [그림 10]과 [그림 11]에서 멀티스레드에 의하여 호출 순서가 변경되거나 GUI 노이즈가 동적 버스마크에 삽입될 때 K-gram 유사도가 취약한 것을 볼 수 있다.

표 4. 기존 유사도 측정 기법과 비교 결과

	K-gram 유사도	Cosine 유사도	제안한 유사도
동일 소프트웨어 (같은 버전)	65.73%	98.67%	87.35%
동일 소프트웨어 (다른 버전)	62.98%	98.43%	85.69%
다른 소프트웨어	35.21%	58.24%	34.81%

[표 4]는 기존 유사도 측정 기법과 비교한 실험 결과를 정리한 표이다. 두 가지 기존 유사도 측정 기법과는 다르게 제안한 유사도 측정 기법은 동일한 소프트웨어 간의 유사도는 높게 측정되었으며, 다른 소프트웨어 간의 유사도는 낮게 측정 되는 것을 확인 할 수 있다. 이는 제안한 유사도 측정 기법이 멀티 스레드에 의한 호출 순서 변경이나 GUI 노이즈에 영향을 잘 받지 않는 것을 알 수 있다. 또한 동일한 소프트웨어의 유사도를 측정할 때 K-gram 유사도 보다는 높은 유사도가 측정되었지만 Cosine 유사도 보다는 낮은 유사도가 측정되었다. 그러나 다른 소프트웨어의 유사도를 측정 할 때

Cosine 유사도 보다는 낮은 유사도가 측정되었고 K-gram 유사도와는 비슷하거나 더 낮은 유사도가 측정되었다.

Ⅴ. 결론 및 향후 연구

이 논문에서는 멀티스레드와 GUI 노이즈 삽입 등으로 인하여 API 호출 순서가 변경되는 동적 버스마크를 호출 순서가 완벽하게 일치하는 부분과 일치하지 않는 부분으로 구별한 후 유사도를 측정하는 유사도 측정 기법을 제안했다. 또한 실험을 통하여 최적의 r값을 찾았으며, 제안하는 유사도 측정 기법이 신뢰성과 강인성을 만족하는지를 평가하였다. 그리고 기존 유사도 측정 기법인 Cosine 유사도와 K-gram 유사도와의 비교 실험을 수행하였다.

실험 결과 제안한 유사도 측정 기법에서 사용되는 r 값이 2000일 때 동일한 소프트웨어간 유사도와 다른 소프트웨어간 유사도가 모두 상승하였지만, 동일한 소프트웨어의 유사도 상승폭이 다른 소프트웨어의 유사도 상승폭보다 높기 때문에 최적의 그룹핑 값이 2000임을 실험을 통해 확인했다. 또한 제안한 유사도 측정 기법은 신뢰성 실험에서 동일한 소프트웨어간 유사도는 높게 측정되었으며, 동일하지만 다른 버전의 소프트웨어간 유사도는 동일 소프트웨어의 유사도 측정값 보다 조금 낮게 측정되어 신뢰성을 만족하는 것을 알 수 있다. 강인성 실험에서는 컴파일러와 컴파일러 옵션이 변경하여 소프트웨어의 유사도를 비교하더라도 높은 유사도가 측정되어 강인성을 만족하는 것을 알 수 있다. 기존 유사도 비교 기법과 제안한 유사도 비교기법을 비교한 결과 동일한 소프트웨어와 다른 소프트웨어의 판별이 잘 되는 것을 실험을 통해 확인했다.

본 논문에서 제안한 유사도 측정 기법을 이용하여 다른 소프트웨어간 유사도를 측정했을 때 평균 34.33%의 유사도가 측정되었다. 이는 독립적으로 개발된 소프트웨어라고 할지라도 윈도우용 소프트웨어에 필수적으로 사용되는 API 들이 동적 버스마크에 포함되어 있기 때문이다. 향후 연구로는 윈도우용 소프트웨어간 유사도 측정 결과를 향상 시키기 위하여 필수적인 API를 제거한 동적 버스마크 추출 방법의 연구가 필요하다.

REFERENCES

[1] Ginger Marie Myles, “Software Theft Detection Through Program Identification”, Ph.D. Thesis, Department of Computer Science, The University of Arizona, 2006.

[2] Collberg C. S., and Thomborson C., "Watermarking, tamper- proofing, and obfuscation-tools for software protection", IEEE Transactions on software engineering, Vol. 28, pp. 735-746, August. 2002.

[3] N.R Wanger, "Fingerprinting", IEEE Symposium on Security and Privacy, pp. 18-22, 1983.

[4] Tamada, H., Okamoto, K., Nakamura, M., Monden, A., and Matsumoto, K. I., "Dynamic software birthmarks to detect the theft of windows applications", International Symposium on Future Software Technology, Vol. 2004, October. 2004,

[5] Tamada, H., Nakamura, M., Monden, A., and Matsumoto, K. I., “Java Birthmarks--Detecting the Software Theft--”, IEICE transactions on information and systems, Vol. 88, No. 9, pp. 2148-2158, 2005.

[6] Ginger Myles and Christian Collberg, “k-Gram Based Software Birthmarks”, In Proceedings of the 2005 ACM Symposium on Applied Computing, pp. 314-318, 2005.

[7] Ricardo B. Y. and Berthier R. N., “Modern Information Retrieval” ACM Press, Vol. 463, 1999.

[8] Galen H. and Doug B., "Detours : binary interception of win32 functions", In Proceedings of the Third USENIX Windows NT Symposium, pp. 135-143, 1999.

저 자 소 개

박대신(학생회원)

2013년 : 목포대학교 정보보호학과 학사 졸업.
2014년 : 숭실대학교 컴퓨터학과 석사 과정.

〈주관심분야 : 시스템소프트웨어, 시스템 보안 〉

지현호(학생회원)

2013년 : 숭실대학교 컴퓨터학부 학사 졸업.
2014년 : 숭실대학교 컴퓨터학과 석박통합과정.

〈주관심분야 : 시스템소프트웨어, 시스템 보안 〉

박영수(학생회원)

2009년 : 강남대학교 컴퓨터공학과 학사 졸업.
2014년 : 숭실대학교 컴퓨터학과 석사 졸업.

〈주관심분야 : 시스템소프트웨어, 임베디드시스템 , 시스템 보안〉

홍지만(정회원)

2003년 : 서울대학교 컴퓨터공학과 박사 졸업.
2003년 ~ 2007년 : 광운대학교 컴퓨터공학부 조교수
2007년 ~ 현재 : 숭실대학교 컴퓨터학과 교수
2013년 ~ 현재 : ACM SIGAPP 회장

〈주관심분야 : 운영체제, 시스템소프트웨어, 임베디드 시스템〉