JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 28 No. 7, pp. 85-93, July 2023
https://doi.org/10.9708/jksci.2023.28.07.085

Design and Implementation of Birthmark Technique for Unity Application

Heewan Park*

*Associate Professor, Dept. of IT Software, Halla University, Wonju, Korea

[Abstract]

Software birthmark refers to a unique feature inherent in software that can be extracted from program binaries even in the absence of the original source code of the program. Like human genetic information, the similarity between programs can be calculated numerically, so it can be used to determine whether software is stolen or copied. In this paper, we propose a new birthmark technique for Android applications developed using Unity. The source codes of Unity-based Android applications use C# language, and since the core logic of the program is included in the DLL module, it must be approached in a different way from normal Android applications. In this paper, a Unity birthmark extraction and comparison system was implemented, and reliability and resilience were evaluated. The use of the Unity birthmark technique proposed in this paper is expected to be effective in preventing illegal copy or code theft of the Unity-based Android applications.

▸ **Key words**: Unity, Android, Software birthmark, Code theft detection, Software protection

[요 약]

소프트웨어 버스마크란 프로그램의 소스 코드가 없는 상태에서도 바이너리 파일로부터 추출 가능한 소프트웨어에 내재된 고유한 특징을 의미한다. 사람의 유전자처럼 유사도를 수치로 계산할 수 있기 때문에 소프트웨어 도용과 복제 여부를 판단하는데 사용할 수 있다. 본 논문에서는 유니티를 이용하여 개발된 안드로이드 애플리케이션에 대한 새로운 버스마크 기법을 제안한다. 유니티 기반 안드로이드 애플리케이션은 C# 언어를 이용하여 코드를 작성하며 프로그램의 핵심 로직은 DLL 모듈에 포함되기 때문에 일반적인 안드로이드 애플리케이션과는 다른 방법으로 접근해야 한다. 본 논문에서 제안한 유니티 버스마크 추출 및 비교 시스템을 구현하여 신뢰도와 강인도를 평가하였다. 평가 결과에 의해서 유니티 버스마크 기법은 유니티 기반으로 제작된 안드로이드 애플리케이션의 코드 도용이나 불법 복제를 예방하는데 효과가 있을 것으로 기대한다.

▸ **주제어**: 유니티, 안드로이드, 소프트웨어 버스마크, 코드 도용 탐지, 소프트웨어 보호

• First Author: Heewan Park, Corresponding Author: Heewan Park
*Heewan Park (heewanpark@halla.ac.kr), Dept. of IT Software, Halla University
• Received: 2023. 06. 07, Revised: 2023. 06. 23, Accepted: 2023. 06. 28.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

유니티(Unity)는 유니티 테크놀로지스社에서 2004년에 개발한 게임 엔진이다. PC와 콘솔 게임기, 스마트폰 등 다양한 플랫폼에서 동작하는 2D 및 3D 게임 개발을 위해서 사용되며 자체 에셋 스토어(asset store)를 운영하여 다양한 리소스를 지원하기 때문에 개발자들이 선호하는 게임 엔진으로 꾸준히 성장하고 있다[1].

안드로이드 애플리케이션을 제작할 때 주로 사용되는 언어는 자바(Java)와 코틀린(Kotlin) 언어이다. 그러나 유니티 엔진을 이용하여 애플리케이션을 구현할 때는 C# 언어를 사용한다. C# 언어로 작성된 소스 코드는 컴파일되어 CIL(Common Intermediate Language)이라는 공통 중간 언어로 변환된다. CIL은 CLR(Common Language Runtime)이라는 공통 언어 런타임을 통해서 실행된다[2].

CIL 언어는 하드웨어에 의존적인 기계어가 아니라 자바 바이트코드와 유사한 중간 언어이기 때문에 디컴파일과 같은 역공학 분석에 취약하다는 단점이 있다. 만일 C# 전용 디컴파일러[3-5]를 사용한다면 원본에 가까운 소스 코드를 얻을 수 있기 때문에 코드 도용이나 불법 복제의 표적이 될 수 있다. 이를 예방하기 위해서 애플리케이션을 배포하기 전에 난독화 기법을 적용하여 역공학 분석을 어렵게 만들 수 있으나 분석 자체를 막을 수 있는 근본적인 해결책이 되지는 못한다.

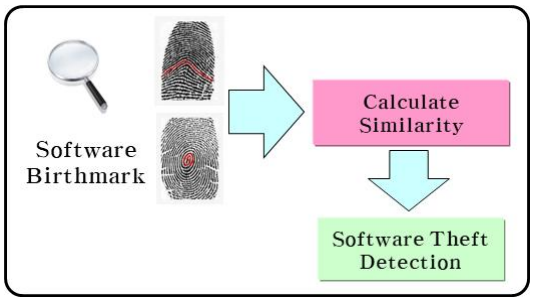


Fig. 1. Concept of the software birthmark

Fig. 1은 소프트웨어 버스마크(software birthmark)의 개념을 보여주는 그림이다. 소프트웨어 버스마크는 마치 사람의 지문과 같은 역할을 하며, 프로그램의 원본 소스를 구하기 어려운 경우에도 소프트웨어의 도용이나 복제를 판단하기 위해서 사용될 수 있는 프로그램 바이너리에 내재된 고유한 특징을 의미한다[6-12].

만일 복제가 의심되는 프로그램의 원본 소스를 구할 수 있다면 표절 검사 프로그램을 사용할 수 있다. 그러나 단순히 도용이나 복제가 의심된다고 프로그램의 소스 코드를 상대방에게 요청할 수는 없기 때문에 프로그램의 바이너리를 대상으로 유사도를 측정할 수 있는 도구가 필요하며 그것이 바로 소프트웨어 버스마크이다.

안드로이드 애플리케이션에 대한 소프트웨어 버스마크 기법은 최근까지 다양하게 연구되어 왔다[11-12]. 기존 안드로이드 버스마크 기법은 안드로이드 가상 머신 명령어가 포함된 DEX(Dalvik Executable) 파일을 분석하는 방법을 적용하였다. 그러나 유니티 기반으로 제작된 안드로이드 애플리케이션의 경우에는 기존 안드로이드에서 사용하는 방법을 적용할 수 없다. 그 이유는 C# 언어를 이용하여 작성된 핵심 로직이 DLL(Dynamic Link Library) 모듈로 안드로이드 assets 폴더에 포함되기 때문이다. 즉, 안드로이드 apk 파일에 포함된 DLL 모듈을 추출하여 C# 중간 언어를 분석하고 유사도를 계산하는 새로운 버스마크 기법이 필요하다.

본 논문에서는 유니티 엔진을 활용하여 제작된 안드로이드 애플리케이션의 코드 도용 여부를 탐지하기 위한 새로운 소프트웨어 버스마크 기법을 제안한다. 기존의 안드로이드 버스마크 기법이 apk 파일에 포함된 DEX 파일을 분석했던 것과 다르게 본 논문에서는 DLL 파일을 분석하여 유사도를 계산한다. 그리고 유니티 버스마크의 효용성을 평가하기 위해서 오픈 소스 프로그램에 대해서 신뢰도(credibility)와 강인도(resilience)를 평가한다.

본 논문의 구성은 다음과 같다. 2장에서 관련 연구를 살펴보고, 3장에서 본 논문에서 제안하는 유니티 버스마크 기법에 대해서 소개하며, 4장에서 구현 내용을 설명하고, 5장에서 실험 및 평가를 하고, 6장에서 결론과 향후 연구 과제에 대해서 소개한다.

II. Related works

프로그램 유사도 측정을 통한 코드 도용 및 불법 복제 탐지 방법은 비교 대상 소프트웨어의 소스 코드 확보 여부에 따라서 소스 코드 표절 검사와 소프트웨어 버스마크 두 가지로 나눌 수 있다.

만일 비교 대상 소프트웨어의 소스 코드를 구할 수 있다면 소스 코드 표절 검사 도구를 활용할 수 있다. 소스 표절 탐지에 사용되는 대표적인 도구로는 Moss[13]와 JPlag[14]가 있다. Moss는 C 언어와 자바 언어를 포함한 다양한 프로그래밍 언어를 지원하며, JPlag는 자바 언어로 작성된 소스를 비교하여 시각적으로 결과를 표현해 준다. 그러나 단순히 코드 도용이 의심된다는 이유로 다른 개발

사가 제작한 소스 코드를 요구할 수는 없기 때문에 불법복제가 의심되는 상황에서 소스 표절 검사 도구를 활용하는 것은 어려운 현실이다.

소스 코드가 확보되지 않은 상황에서 프로그램 바이너리 파일에 대한 유사도 계산 방법으로 제안된 개념이 소프트웨어 버스마크[6-12]이다. 소프트웨어 버스마크는 생물학적인 DNA 개념과 유사하다. 즉, 프로그램 바이너리 파일에 내재된 고유한 특징이기 때문에 소스 코드가 없는 상황에서도 유사한 프로그램을 식별하는데 사용될 수 있다.

소프트웨어 버스마크에 대한 연구는 Tamada 버스마크[9]와 Myles의 k-gram 버스마크[10]에서 시작되었다.

Tamada는 자바 클래스 파일에 적용이 가능한 소프트웨어 버스마크 기법을 처음 제안하였다. 이 기법은 자바 클래스 파일을 분석하여 (1) 자바 클래스에서 사용된 상수 값들의 순서, (2) 자바 메소드 호출 순서, (3) 클래스 상속 구조, (4) 사용된 클래스 정보의 네 가지 특징을 버스마크로 사용하여 자바 클래스 사이의 유사도를 계산하였다[9].

Myles는 자바 클래스에 포함된 JVM(Java Virtual Machine) 명령어들을 크기가 k인 조각으로 분리하여 유사도를 계산하는 k-gram 버스마크를 제안하였다. k-gram 버스마크는 JVM 명령어 단위로 유사도를 계산하기 때문에 다른 버스마크보다 신뢰도가 높다는 특징이 있다. 그러나 유사한 기능의 다른 명령어로 대체하거나 명령어의 순서를 변경할 경우에는 유사도가 낮아질 수 있다는 단점이 있다[10].

안드로이드는 비록 자바와 가상 머신의 종류는 다르지만 중간 언어를 사용하고 있다는 공통점 때문에 자바에서 사용하던 버스마크 기법과 유사한 방식을 적용할 수 있다. 안드로이드 앱을 대상으로 하는 API 호출 기반 버스마크[11]에서는 API 메소드 호출 순서를 분석하여 프로그램 유사도를 계산하는 방식을 제안하였고, 사용자 정의 메소드 기반 버스마크[12]에서는 사용자가 직접 정의 메소드를 분석하되 메소드명은 제외하고 매개변수 및 반환 값의 자료형만을 사용하여 난독화 도구에 의한 프로그램 변형에 대해서도 높은 유사도 값을 유지하였다.

최근까지 진행된 안드로이드를 대상으로 하는 소프트웨어 버스마크 연구는 주로 자바 언어로 작성된 애플리케이션의 중간 언어를 분석하는 방법을 사용해왔으며 유니티 개발 도구를 이용해서 제작된 안드로이드 앱에 대한 연구는 이루어지지 않았다. 본 논문에서는 유니티 엔진을 기반으로 C# 언어를 이용하여 제작된 안드로이드 애플리케이션에 대한 새로운 소프트웨어 버스마크 기법을 제안하고 성능 평가를 수행한다.

III. Unity-based Software Birthmark

1. Software Birthmark

소프트웨어 버스마크에 대한 정의는 다음과 같다. 소프트웨어 버스마크는 어떠한 추가적인 정보가 없는 상태에서 프로그램 P 자신으로부터 추출할 수 있어야 한다. 그리고 두 프로그램 P와 Q가 서로 코드 도용 관계에 있다면 두 프로그램에서 추출한 소프트웨어 버스마크가 서로 같아야 한다[9].

물론 완벽하게 동일한 프로그램으로부터 추출한 소프트웨어 버스마크는 당연히 동일하겠지만 일반적으로 코드를 도용하고자 하는 의도가 있을 때에도 전체가 아닌 일부분의 코드를 도용하거나 일부분을 원본과 다르게 수정하는 것도 고려해야 한다. 따라서 두 프로그램으로부터 추출한 버스마크의 유사도를 고려해서 코드 도용 여부를 판단해야 한다.

소프트웨어 버스마크를 활용할 때 주의해야 할 점은 두 프로그램으로부터 추출된 버스마크가 같다고 해서 반드시 도용했다고 단정할 수는 없다는 것이다. 프로그램의 크기가 매우 작거나 관용적으로 많이 사용되는 코드인 경우에는 도용하지 않고 코드를 작성하더라도 소프트웨어 버스마크는 같을 수 있기 때문이다. 즉, 소프트웨어 버스마크는 코드 도용을 법적으로 입증하는 증거가 될 수는 없으며 도용을 의심할 수 있는 근거 자료로 사용될 수 있다[9].

2. Unity birthmark system

유니티 버스마크 시스템은 유니티 프로그램으로부터 핵심 로직을 추출하여 유사도를 비교해야 한다. 다음은 유니티로 제작한 애플리케이션으로부터 버스마크를 생성하고 유사도를 비교하는 방법에 대한 예시이다.

```
if (Input.GetMouseButtonDown(0))
{
    this.startPos = Input.mousePosition;
}
else if (Input.GetMouseButtonUp(0))
{
    Vector2 endPos = Input.mousePosition;
    float swipeLength = endPos.x - this.startPos.x;
    this.speed = swipeLength / 500.0f;
}
```

Fig. 2. Sample C# codes of Unity application

Fig. 2는 유니티 애플리케이션 개발을 위해서 C# 언어로 작성한 소스코드의 샘플이다. 해당 소스코드는 마우스 버튼을 누르고 뗀 좌표를 인식하고 그 거리를 계산하여 변

수에 저장하도록 작성되었다.

유니티 개발 환경에서 안드로이드용 apk 파일을 빌드하면 C# 언어로 작성한 핵심 모듈은 apk 파일에 포함된 assets 폴더 내부의 Assembly-CSharp.dll 파일에 저장된다. 따라서 이 DLL 파일을 dnSpy[5]와 같은 역공학 프로그램으로 분석하여 공통 중간 언어로 구성된 CIL 파일을 생성할 수 있다.

```
01: IL_0000: ldc.i4.0
02: IL_0001: call bool ... Input::GetMouseButtonDown(int32)
03: IL_0006: brfalse.s IL_001A
04: IL_0008: ldarg.0
05: IL_0009: call ... Input::get_mousePosition()
06: IL_000E: call ... Vector2::op_Implicit(...)
07: IL_0013: stfld ... CarController::startPos
08: IL_0018: br.s IL_0056
09: IL_001A: ldc.i4.0
10: IL_001B: call bool ... Input::GetMouseButtonUp(int32)
11: IL_0020: brfalse.s IL_0056
12: IL_0022: call ... Input::get_mousePosition()
13: IL_0027: call ... Vector2::op_Implicit(...)
14: IL_002C: ldfld float32 ... Vector2::x
......
```

Fig. 3. Sample CIL codes of Unity application

Fig. 3은 Fig. 2의 샘플 소스를 빌드하여 생성한 DLL 파일을 dnSpy로 디스어셈블한 결과를 보여준다. 비록 완벽하지는 않지만 디컴파일 기능을 사용한다면 원본 소스 코드와 유사한 C# 소스 코드를 얻을 수도 있다. 본 논문에서 제안하는 유니티 버스마크 시스템에서는 중간 언어 형태인 CIL 파일에 포함된 명령어들을 k개의 조각으로 나누어서 버스마크로 사용한다.

```
01: IL_0000: ldc.i4.0
02: IL_0001: call bool ... Input::GetMouseButtonDown(int32)
03: IL_0006: brfalse.s IL_001A

02: IL_0001: call bool ... Input::GetMouseButtonDown(int32)
03: IL_0006: brfalse.s IL_001A
04: IL_0008: ldarg.0

03: IL_0006: brfalse.s IL_001A
04: IL_0008: ldarg.0
05: IL_0009: call ... Input::get_mousePosition()
.....
```

Fig. 4. Example of 3-size code fragments of Fig. 2

Fig. 4는 Fig. 3의 중간 언어로부터 생성된 크기가 3인 코드 조각들을 보여준다. 조각의 크기가 3인 경우에는 시작 포인트를 슬라이딩 시키면서 일정하게 3개의 중간 언어 명령어를 모아서 명령어 시퀀스 집합을 생성한다.

```
ldc.i4.0
call bool ... Input::GetMouseButtonDown(int32)
brfalse.s

call bool ... Input::GetMouseButtonDown(int32)
brfalse.s
ldarg.0

brfalse.s
ldarg.0
call ... Input::get_mousePosition()
.....
```

Fig. 5. Example of 3-gram of Fig. 3

Fig. 5는 Fig. 4의 중간 언어 조각으로부터 생성된 3-gram 버스마크를 보여준다. 코드 조각으로부터 버스마크를 생성할 때는 쉽게 변경이 가능한 변수 이름이나 레이블은 제거한다. 즉, 크기가 일정한 명령어들의 시퀀스들의 집합이 바로 k-gram 버스마크이며 크기가 3인 경우에 3-gram 버스마크라고 부른다.

버스마크를 생성할 때 조각의 크기가 너무 크면 프로그램의 일부분이 변경될 경우에도 버스마크가 쉽게 손상될 수 있기 때문에 버스마크의 강인도가 낮아질 수 있고, 반대로 조각의 크기가 너무 작으면 프로그램의 고유한 특성을 충분히 반영할 수 없기 때문에 버스마크의 신뢰도가 낮아질 수 있다. 따라서 적절한 k 값을 정하는 것이 중요하며 관련 연구에서는 일반적으로 신뢰도와 강인도를 모두 높일 수 있는 값인 3을 주로 사용하고 있다[10,11].

비교 대상 프로그램으로부터 버스마크가 추출되었으면 두 버스마크의 유사도를 계산해야 한다. 프로그램 P와 Q로부터 추출된 k-gram 버스마크를 각각 k-gram(P)와 k-gram(Q)라고 할 때, 버스마크의 유사도 Similarity (P,Q)는 다음과 같이 구한다[10,11].

$$\text{Similarity}(P,Q) = \frac{|\text{k-gram}(P) \cap \text{k-gram}(Q)|}{\min(|\text{k-gram}(P)|,|\text{k-gram}(Q)|)}$$

버스마크 유사도 계산식에서 |k-gram(P)|는 k-gram(P)의 원소의 개수를 의미한다. 즉, 프로그램 P와 Q에서 추출한 k-gram 중에서 작은 값을 기준 값으로 정하고, 두 프로그램에 공통으로 포함되어있는 k-gram의 비율을 유사도로 계산한다. 분모에 min 함수를 사용하였기 때문에 원본을 복제한 이후에 코드를 추가한 경우에도 유사도가 높게 계산될 수 있도록 하였다. 즉, 원본의 소스 코드를 그대로 활용하고 추가로 많은 코드를 덧붙인다고 하더라도 유사도는 원본을 기준으로 계산되기 때문에 높은 유사도 값이 계산될 수 있다.

IV. Implementation

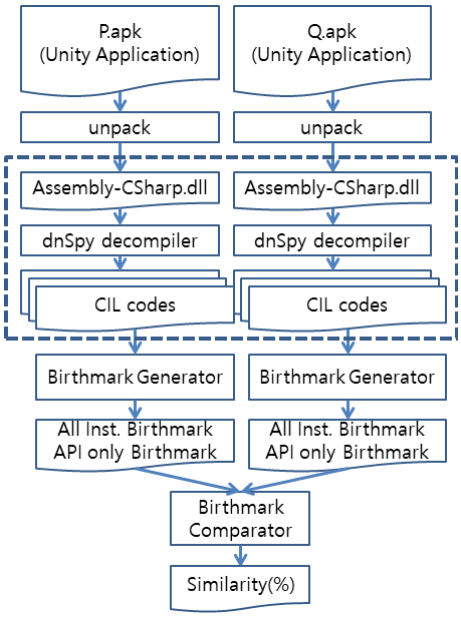


Fig. 6. Structure of Unity based Android application birthmark system

Fig. 6은 유니티 기반으로 제작된 안드로이드 애플리케이션의 버스마크 시스템 구조도이다. 시스템의 동작 절차는 다음과 같다. 시스템의 입력은 안드로이드 환경에서 동작하는 두 개의 유니티 애플리케이션이다. apk 형식인 애플리케이션의 압축을 해제한 후 assets 폴더에 포함되어 있는 Assembly-CSharp.dll 파일을 추출한다. 그리고 dnSpy[5] 역공학 도구를 활용하여 중간 언어인 CIL 코드를 추출한다. CIL 중간 언어를 입력으로 받아서 버스마크 생성 모듈은 유니티 버스마크를 생성하고, 버스마크 비교 모듈은 두 유니티 버스마크 사이의 유사도 값을 계산한다.

본 논문에서는 버스마크를 생성할 때 두 가지 유형의 버스마크를 생성하였다. 첫 번째, 모든 CIL 중간 언어 명령어를 대상으로 3개의 명령어 조각들을 연결하여 CIL 명령어 기반 버스마크를 생성하였다. 두 번째, CIL 중간 언어 명령어 중에서 API 메소드를 호출하는 명령어만을 추출하여 3개의 명령어 조각들을 연결하여 API 메소드 기반 버스마크를 생성하였다. 프로그램을 구성하는 모든 명령어들을 버스마크 생성에 사용하는 것이 일반적이지만 프로그램의 크기가 크거나 비교 대상이 많은 경우에는 중요한 명령어만을 추출하여 생성되는 버스마크의 크기를 줄일 수 있으며 대표적으로 선택 가능한 것이 API 메소드 호출이다. 그 이유는 API 메소드 호출 명령어는 다른 명령어로 대체하기 어려운 특징이 있기 때문이다. 그러나 API 호출이 많지 않은 프로그램에서는 버스마크 생성 자체가 어려울 수도 있기 때문에 상황에 따라서 적절한 방법을 선택할 필요가 있다.

본 논문에서는 두 가지 유형의 버스마크를 모두 추출하고 두 결과를 비교 분석하여 신뢰도와 강인도를 평가하였다. 유니티 기반 안드로이드 버스마크 생성 및 비교 시스템은 Visual Studio 2022 버전을 활용하여 C++ 언어로 구현하였고 Windows 10 환경에서 실험하였다.

V. Experiment and Evaluation

실험 대상 프로그램으로는 오픈 소스로 인터넷에 공개된 SwipeCar, CatEscape, ClimbCloud, Bamsongi, AppleCatch 5개의 유니티 게임 앱[15]을 선정하였다. 그리고 모든 명령어 기반 CIL 버스마크와 API 메소드 호출 명령어 기반 버스마크의 신뢰도를 평가하였고, Dotfuscator 난독화 도구[16]를 사용하여 강인도를 평가하였다. 추가로 원본 게임 앱의 기능을 일부 수정하였을 때 버스마크의 유사도 변화를 확인하기 위한 실험을 진행하였다.

버스마크 유사도 계산 실험에서 어느 정도 수치의 유사도가 계산될 경우에 코드 도용으로 판단할 것인가에 대해서는 기존 관련 연구 결과를 참고하였다[17]. 실험에 사용된 버스마크 기법에 따라서 차이가 있을 수 있으나, 일반적으로 60% 이상의 유사도가 계산될 경우에 도용으로 의심할 수 있으며 80% 이상인 경우에는 도용이 거의 확실하다고 판단할 수 있다.

1. Evaluation of Unity Birthmark Credibility

소프트웨어 버스마크 시스템에서 버스마크의 신뢰도(credibility)란 서로 다른 프로그램을 구별할 수 있는 특성을 의미한다. 소프트웨어 버스마크가 마치 유전자 정보인 DNA처럼 소프트웨어의 고유한 특징으로서 사용되기 위해서는 신뢰도는 매우 중요하며 서로 다른 프로그램들에 대해서 낮은 유사도 측정값을 가질수록 신뢰도가 높다고 평가할 수 있다.

1.1 CIL instruction based birthmark credibility

Table 1. Credibility result of Unity CIL Birthmark (Name of applications : A=SwipeCar, B=CatEscape, C=ClimbCloud, D=Bamsongi, E=AppleCatch)

	original app	A	B	C	D	E
	# of k-gram	67	95	126	36	194
A	# of equal k-gram	-	15	17	5	18
	similarity(%)	-	22.4	25.4	13.9	26.9
B	# of equal k-gram	-	-	27	4	40
	similarity(%)	-	-	28.4	11.1	42.1
C	# of equal k-gram	-	-	-	7	29
	similarity(%)	-	-	-	19.4	23.0
D	# of equal k-gram	-	-	-	-	7
	similarity(%)	-	-	-	-	19.4

실험 대상 앱으로부터 추출한 모든 중간언어 명령어를 대상으로 k-gram을 생성하여 유사도를 비교하는 CIL 명령어 기반 신뢰도 평가 결과는 Table 1과 같다. 5개의 앱을 대상으로 서로 다른 2개의 앱을 선택하여 유사도를 계산하기 때문에 ${}_5C_2$ = 10번의 비교를 하였다. 소프트웨어 버스마크에서 신뢰도는 서로 다른 프로그램과의 유사도이며 수치가 낮을수록 우수하다. 신뢰도 평가 결과로 최소 11.1%에서 최대 42.1%에 이르는 값이 측정되었으며 서로 다른 앱을 구분할 수 있을 정도로 낮은 값이므로 본 논문에서 제안한 버스마크는 신뢰도가 우수함을 알 수 있다.

1.2 API call instruction based credibility

Table 2. Credibility result of Unity API Birthmark

	original app	A	B	C	D	E
	# of k-gram	21	34	41	16	56
A	# of equal k-gram	-	1	1	0	2
	similarity(%)	-	4.8	4.8	0	9.5
B	# of equal k-gram	-	-	0	0	9
	similarity(%)	-	-	0	0	26.5
C	# of equal k-gram	-	-	-	0	0
	similarity(%)	-	-	-	0	0
D	# of equal k-gram	-	-	-	-	1
	similarity(%)	-	-	-	-	6.3

Table 2는 실험 대상 앱으로부터 추출한 API 메소드 호출 명령어를 이용하여 k-gram을 생성하여 유사도를 측정하는 API 기반 신뢰도 평가 결과이다. 유사도 평가 결과로 최소 0%에서 최대 26.5%에 이르는 값이 측정되었다. 10개의 유사도 계산 값이 CIL 명령어를 대상으로 k-gram을 생성하는 Table 1의 실험 결과보다 모두 낮은 값으로 계산되었다. 따라서 API 명령어 기반 버스마크가 모든 CIL 명령어 기반 버스마크보다 신뢰도가 우수하다고 평가할 수 있다.

API 버스마크가 CIL 버스마크보다 신뢰도가 우수한 이유는 버스마크 생성 대상이 프로그램을 구성하는 모든 명령어가 아니라 API 호출 명령어만을 대상으로 하기 때문이다. API 호출 명령어는 다른 명령어로 대체하기 어려우며 프로그램의 핵심 기능을 수행하는 역할을 하므로 서로 다른 프로그램을 구별할 수 있는 변별력이 높아서 결과적으로 버스마크의 신뢰도가 높아질 수 있다. 그러나 API 기반 버스마크는 API 메소드 호출 명령어가 다수 존재하는 프로그램에서만 적용 가능하다는 단점이 있기 때문에 자료구조나 알고리즘이 서로 다른 프로그램을 구별하는 용도로는 사용하기 어렵다는 한계가 있다.

2. Evaluation of Unity Birthmark Resilience

소프트웨어 버스마크 시스템에서 버스마크의 강인도(resilience)란 프로그램의 최적화나 난독화와 같은 프로그램 변환에 대해서 버스마크가 손상되지 않고 유지되는 특성을 의미한다. 프로그램을 도용한 사람은 도용 사실을 숨기기 위해서 다양한 변환 기법을 적용할 수 있기 때문에 이러한 프로그램 변환에 대해서도 버스마크가 손상되지 않고 유지되는 것은 매우 중요하다.

본 논문에서 제안하는 버스마크 시스템의 난독화에 대한 강인도를 평가하기 위해서 .NET 난독화 도구인 Dotfuscator를 사용하였다. Dotfuscator는 PreEmptive 社에서 개발한 도구로서 Visual Studio 정식 버전에 포함되어 배포되는 대표적인 난독화 도구이다.

난독화 옵션 중에서 가장 일반적인 것은 식별자 이름의 난독화(renaming obfuscation)이다. 프로그래머가 직접 정의한 변수나 함수의 이름을 a, b, c 등 본래의 의미를 파악할 수 없는 임의의 문자로 변경하기 때문에 역공학 분석을 어렵게 만드는 효과가 있다. 그러나 본 논문에서 제안하는 버스마크를 생성할 때는 식별자 이름을 사용하지 않기 때문에 의미가 없는 난독화 옵션이다.

CIL 명령어 기반 버스마크와 API 메소드 기반 버스마크 시스템은 명령어의 시퀀스를 이용하여 k-gram을 생성하기 때문에 이를 방해하기 위한 난독화는 제어 흐름 난독화(control flow obfuscation)이다. 프로그램의 제어 흐름이 변경되면 명령어 시퀀스가 변경될 수 있기 때문에 버스마크 시스템에 의해서 생성되는 k-gram도 변경될 수 있다. Dotfuscator는 제어 흐름 난독화로 Low, Medium, High의 3 단계 옵션을 제공하며 본 논문에서는 난독화 강도를 High로 설정하여 가장 강도 높은 제어 흐름 난독화를 수행한 후 버스마크의 강인도를 평가하였다.

2.1 CIL instruction based resilience

Table 3. Resilience result of Unity CIL Birthmark

original app	A	B	C	D	E
# of k-gram	67	95	126	36	194
obfuscated app	A'	B'	C'	D'	E'
# of k-gram	137	187	226	97	318
# of equal k-gram	54	68	78	25	140
similarity(%)	80.6	71.6	61.9	69.4	72.2

Table 3은 CIL 명령어 기반 버스마크의 난독화에 대한 강인도 평가 결과이다. 강인도는 다양한 변환 기법에 대해서도 버스마크가 손상되지 않고 유지되는 특성을 의미하기 때문에 원본 프로그램과 난독화 이후의 프로그램의 유사도가 높을수록 우수한 버스마크이다. 실험 결과 최소 61.9%, 최대 80.6%에 이르는 유사도 값을 얻었으므로 강인도가 우수하다고 평가할 수 있다.

2.2 API call instruction based resilience

Table 4. Resilience result of Unity API Birthmark

original app	A	B	C	D	E
# of k-gram	21	34	41	16	56
obfuscated app	A'	B'	C'	D'	E'
# of k-gram	21	34	41	16	56
# of equal k-gram	17	30	26	16	43
similarity(%)	81.0	88.2	63.4	100	76.8

Table 4는 API 명령어 기반 버스마크의 난독화에 대한 강인도 평가 결과이다. 실험 결과 최소 63.4%, 최대 100%에 이르는 유사도 값을 얻었다.

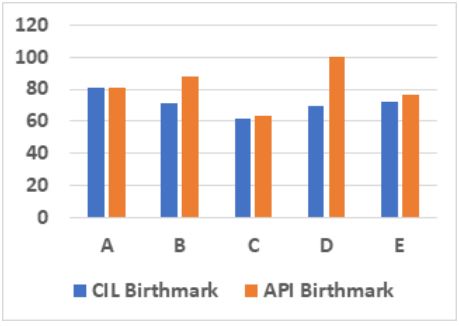


Fig. 7. Comparison of resilience between CIL birthmark and API birthmark

Fig. 7은 CIL 버스마크와 API 버스마크를 비교한 결과이다. 실험 대상으로 사용된 5개의 앱에서 모두 CIL 버스마크보다 API 버스마크가 유사도가 높게 계산되었다. API 메소드 호출은 프로그램의 핵심 기능을 담당하기 때문에 순서를 변경하거나 수정하기 어렵기 때문에 모든 명령어를 대상으로 하는 CIL 버스마크보다 강인도가 높게 평가되었다고 해석할 수 있다.

3. Case Study of Unity Birthmark

소프트웨어 버스마크는 원본 프로그램을 수정하여 업데이트 하는 경우에도 손상되지 않고 유지되는 강인도가 중요하다. 따라서 일반적으로 난독화 도구를 이용해서 강인도를 평가하다. 본 논문에서는 실험에 사용된 프로그램이 유니티 게임이라는 특성을 감안하여 원본 게임을 도용한 사람이 게임의 기능을 일부 변경하는 상황을 가정하여 강인도 평가를 추가로 수행하였다.

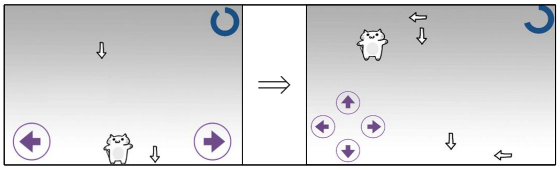


Fig. 8. The capture image of original CatEscape Unity application and it's upgraded version

Fig. 8의 왼쪽 그림은 CatEscape[15]라는 게임의 원본이고, 오른쪽 그림은 원본 게임의 수정본이다. 원본 게임은 캐릭터가 좌, 우로만 이동 가능하고 화살이 화면 상단에서 내려오도록 구현되었으나 수정본에서는 캐릭터가 전, 후, 좌, 우 4 방향으로 이동 가능하고 화살도 상단 및 우측에서 동시에 날아오도록 수정하였다.

Table 5. Resilience result of original and modified version of CatEscape Unity application

	CIL Birthmark	API Birthmark
original app # of k-gram	95	34
modified app # of k-gram	113	41
# of equal k-gram	94	34
similarity(%)	99.0	100

Table 5는 CatEscape 원본 게임과 수정된 게임에 대한 강인도 평가 결과이다. CIL 기반 버스마크에서는 유사도가 99.0%로 계산되었으며 API 기반 버스마크에서는 100%로 계산되었다. 게임을 수정하는 과정에서 일부 C# 명령어가 추가되었으나 간단한 수정만으로는 버스마크가 쉽게 손상되지 않고 유지됨을 알 수 있다.

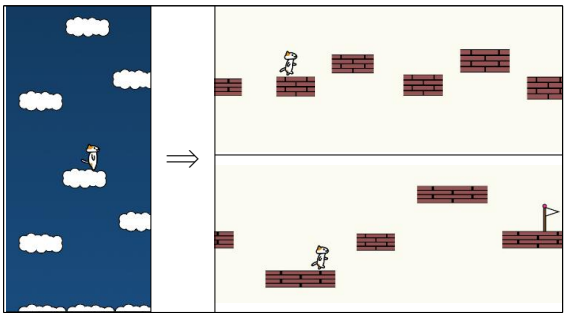


Fig. 9. The capture image of original ClimbCloud Unity application and it's upgraded version

Fig. 9의 왼쪽 그림은 ClimbCloud[15]라는 게임의 원본이고, 오른쪽 그림은 원본 게임의 수정본이다. 원본 게임은 고양이 캐릭터가 구름을 밟고 위로 올라가야 하는 종스크롤 방식으로 동작한다. 수정본에서는 벽돌이 상/하, 좌/우로 움직이는 동작이 추가되었고 고양이 캐릭터는 움직이는 벽돌을 밟고 오른쪽으로 이동해야 하는 횡스크롤 방식의 게임으로 수정하였다.

Table 6. Resilience result of original and modified version of ClimbCloud Unity application

	CIL Birthmark	API Birthmark
original app # of k-gram	126	41
upgraded app # of k-gram	225	72
# of equal k-gram	92	34
similarity(%)	73.0	82.9

Table 6은 ClimbCloud 원본 게임과 수정된 게임에 대한 강인도 평가 결과이다. CIL 기반 버스마크에서는 유사도가 73.0%로 계산되었으며 API 기반 버스마크에서는 82.9%로 계산되었다. 구름에서 벽돌로 바뀐 디자인은 버스마크 유사도 계산에 영향을 끼치지 않았지만 벽돌이 상/하, 좌/우로 움직이는 동작이 추가되고 게임의 스크롤 방식이 바뀌면서 앞선 Table 5의 실험보다 유사도가 낮아지는 현상이 발생하였다. 그러나 73.0%와 82.9%라는 유사도 값은 충분히 도용을 의심할 수 있을 정도로 높은 유사도이므로 본 논문에서 제안하는 버스마크는 앱의 부분적인 수정이 있더라도 도용 탐지를 위해서 사용될 수 있음을 보여준다.

VI. Conclusions and Future Work

소프트웨어 버스마크는 프로그램에 내재된 고유한 특징으로서 프로그램 도용 탐지 여부를 확인하는데 사용될 수 있는 기술이다. 자바와 안드로이드 등 다양한 분야에서 소프트웨어 버스마크가 도입되고 연구 결과가 발표되었다. 그러나 게임 개발을 위해서 주로 사용되는 유니티 엔진을 기반으로 제작된 프로그램에 대한 버스마크 기법은 현재까지 제안되지 않았다.

본 논문에서는 기존 연구에서 다뤄지지 않았던 유니티 앱을 대상으로 C# 바이트코드 분석 결과를 활용하는 새로운 버스마크 기법을 제안하고 시스템을 구현하였다. 또한 실험을 통해서 버스마크의 신뢰도와 강인도를 평가하고 버스마크로서의 유용성을 입증하였다. 본 논문에서 제안한 유니티 버스마크가 널리 활용된다면 유니티 엔진을 사용하여 제작된 안드로이드 애플리케이션에 대한 코드 도용이나 불법 복제 시도를 예방할 수 있을 것으로 기대한다.

향후 과제로는 유니티 중간언어를 IL2CPP[18] 방식으로 변환하여 생성된 바이너리에 대한 소프트웨어 버스마크 기법의 연구를 고려하고 있다. IL2CPP는 기존 C# 중간언어를 C++ 언어로 변환하여 바이너리를 생성하는 방식을 의미한다. 따라서 C++ 언어를 컴파일하여 생성된 바이너리 파일에 대한 분석 작업과 버스마크 생성 및 비교 시스템 구현이 필요하다.

REFERENCES

[1] “Unity,” *https://www.unity.com*

[2] “Microsoft .NET Document,” *https://docs.microsoft.com/dotnet/*

[3] “ILdasm: IL Disassembler,” *https://docs.microsoft.com/dotnet/framework/tools/ildasm-exe-il-disassembler*

[4] “ILSpy: .NET Decompiler with support for PDB generation, ReadyToRun, Metadata,” *https://github.com/icsharpcode/ILSpy*

[5] “dnSpy: .NET debugger and assembly editor,” *https://github.com/dnSpy/dnSpy*

[6] Nazir, S., Khan, H.U., “An Overview on the Identification of Software Birthmarks for Software Protection,” Lecture Notes in Networks and Systems, Vol 614, pp. 323-330, Springer, Singapore, May, 2023. DOI: 10.1007/978-981-19-9331-2_27

[7] Keqing G., Shah N., Xianli K., Sadaqat ur R., “Software Birthmark Usability for Source Code Transformation Using Machine Learning Algorithms,” Scientific Programming, Vol. 2021, Article ID 5547766, 7 pages, 2021. DOI: 10.1155/2021/5547766

[8] C. K. Chung and P. C. Wang, “Version-Wide Software Birthmark via Machine Learning,” IEEE Access, Vol. 9, pp. 110811-110825, 2021. DOI: 10.1109/ACCESS.2021.3103186

[9] H. Tamada, M. Nakamura, A. Monden, K. Matsumoto, “Java birthmark - Detecting the software theft,” IEICE Transactions on Information and Systems, Vol.E88-D, No.9, pp.2148-2158, Sep.,

2005. DOI: 10.1093/ietisy/e88-d.9.2148

[10] Ginger Myles and Christian Collberg. “k-gram Based Software Birthmarks,” In Pro. of the 2005 ACM Symposium on Applied Computing, pp. 314-318, 2005. DOI: 10.1145/1066677.1066753

[11] H. Park. “An Android Birthmark based on API k-gram,” KIPS Trans. on Computer and Communication Systems, Vol. 2, No. 4, pp. 177-180. Apr. 2013. DOI: 10.3745/KTCCS.2013.2.4.177

[12] D. Kim, S. Cho, Y. Chung, J. Woo, J. Ko and S. Yang. “Android App Birthmarking Technique Resilient to Code Obfuscation,” The Journal of Korean Institute of Communications and Information Sciences, Vol. 40 No. 04, pp. 700-708. Apr. 2015. DOI: 10.7840/kics.2015.40.4.700

[13] “Moss - A System for Detecting Software Similarity,” *https://theory.stanford.edu/~aiken/moss/*

[14] “JPlag - Detecting Software Plagiarism,” *https://github.com/jplag/JPlag*

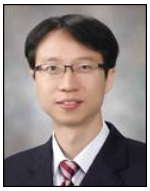
[15] “Unity textbook 5th edition sample files,” *https://github.com/gilbutITbook/080302*

[16] “Dotfuscator - .Net obfuscator,” *https://www.preemptive.com/products/dotfuscator/*

[17] H. Park, H. Lim, S. Choi and T. Han, “Detecting Common Modules in Java Packages Based on Static Object Trace Birthmark,” The Computer Journal, Vol. 54, No. 1, pp. 108-124, Jan. 2011. DOI: 10.1093/comjnl/bxp095

[18] “IL2CPP Overview,” *https://docs.unity3d.com/2021.3/Documentation/Manual/IL2CPP.html*

Authors



Heewan Park received the B.S. degree in Dept. of Computer Engineering from Dongguk University, Korea in 1997 and received the M.S. and Ph.D. degrees in Dept. of Computer Science from KAIST, Korea in 1999 and 2010, respectively. Dr. Park joined the faculty of the Dept. of IT Software at Halla University, Wonju, Korea in 2011. He is currently a associate professor in the Dept. of IT Software, Halla University. He is interested in code obfuscation, reverse engineering, software birthmark and software watermark.