Windows 프로그램 도용 탐지를 위한 기능 단위 동적 API 버스마크

(A Functional Unit Dynamic API Birthmark for Windows Programs Code Theft Detection)

최 석 우 † 조 우 영 †† 한 태 숙 †††
(Seokwoo Choi) (Wooyoung Cho) (Taisook Han)

요 약 소프트웨어 버스마크란 코드 도용 탐지를 위해 프로그램 자체에서 추출된 프로그램의 특징이다. 동적 API 버스마크는 실행 시간 API 호출 시퀀스로부터 추출된다. Tamada가 제안한 Windows 프로그램을 위한 동적 API 버스마크는 프로그램 실행 시작 부분의 API 시퀀스만을 추출하여 프로그램의 중요한 특성을 반영하지 못하였다. 이 논문에서는 프로그램의 핵심 기능을 실행할 때의 API 시퀀스에서 추출한 기능 단위 동적 API 버스마크를 제안한다. 기능 단위 동적 API 버스마크를 이용해 코드 도용을 탐지하기 위해서 먼저 두 프로그램을 실행하여 버스마크를 추출한다. 두 프로그램의 유사도는 프로그램에서 추출한 버스마크를 준전체 정렬 방식을 이용하여 비교하여 측정한다. 버스마크의 신뢰성을 평가하기 위하여 같은 기능을 가진 프로그램들을 대상으로 실험하였다. 강인성을 평가하기 위하여 동일한 소스 코드를 다양한 컴파일 방법으로 만들어 실험하였다. 실험 결과 본 논문에서 제안하는 기능 단위 동적 API 버스마크가 기존의 버스마크에서 탐지할 수 없었던 모듈 단위 도용을 탐지할 수 있음을 보였다.

키워드 : 소프트웨어 버스마크, 코드 도용 탐지, 동적 프로그램 분석, 데이터 마이닝

Abstract A software birthmark is a set of characteristics that are extracted from a program itself to detect code theft. A dynamic API birthmark is extracted from the run-time API call sequences of a program. The dynamic Windows API birthmarks of Tamada et al. are extracted from API call sequences during the startup period of a program. Therefore, the dynamic birthmarks cannot reflect characteristics of main functions of the program. In this paper, we propose a functional unit birthmark(FDAPI) that is defined as API call sequences recorded during the execution of essential functions of a program. To find out that some functional units of a program are copied from an original program, two FDAPIs are extracted by executing the programs with the same input. The FDAPIs are compared using the semi-global alignment algorithm to compute a similarity between two programs. Programs with the same functionality are compared to show credibility of our birthmark. Binary executables that are compiled differently from the same source code are compared to prove resilience of our birthmark. The experimental result shows that our birthmark can detect module theft of software, to which the existing birthmarks of Tamada et al. cannot be applied.

Key words : software birthmark, code theft detection, dynamic program analysis, data mining

· 본 연구는 지식경제부 및 정보통신연구진흥원의 대학 IT연구센터 지원사업(IITA-2009-C1090-0902-0020) 및 2009년도 정부(교육과학기술부)의 재원으로 한국과학재단의 지원(No. 2009-0060477)을 받아 수행된 연구임

† 학생회원 : KAIST 전산학과
choi.seokwoo@kaist.ac.kr
†† 비 회 원 : 티맥스코어 R&D Center Core본부
wooyoung_cho@tmax.co.kr
††† 종신회원 : KAIST 전산학과 교수
han@kaist.ac.kr

논문접수 : 2009년 4월 28일
심사완료 : 2009년 7월 17일

정보과학회논문지: 소프트웨어 및 응용 제36권 제9호(2009.9)

1. 서 론

코드 도용은 코드를 주요 자산으로 하고 있는 많은 회사에 직접적인 손해를 끼치고 있다. 코드 도용을 방지하는 기존 연구는 주로 소스 코드의 도용을 증명하는 것이 초점이다. 소스 코드 도용을 탐지하기 위한 기존 연구들은 코드를 도용하는 측에서 소스 코드를 배포하지 않고 실행 파일만 배포하는 경우 적용할 수 없게 된다. 소프트웨어 버스마크(software birthmark)는 소스 코드가 없고, 실행 파일만 있는 경우에 코드 도용을 탐지하기 위한 방법 중 하나이다.

소프트웨어 버스마크는 코드 도용 탐지를 위해 프로그램의 고유한 특징을 추출한 것이다. 소프트웨어 버스마크는 추출 방법에 따라서 정적 버스마크(static birthmark)와 동적 버스마크(dynamic birthmark)로 구분할 수 있다[1]. 정적 버스마크는 코드를 실행하지 않고 코드 자체를 분석한 결과이며, 동적 버스마크는 코드를 실행하며 나타나는 특징을 분석한 결과이다. 정적 버스마크는 프로그램 자체를 분석하여 프로그램을 실행할 때 나타나는 모든 상태를 예측할 수 있다. 그러나 상태를 요약하는 과정에서 정보가 손실되기 때문에 정확성이 떨어지기 쉽다. 동적 버스마크는 주어진 입력을 가지고 프로그램을 직접 실행하며 나타나는 외부적인 특징들을 기록한다. 실행되는 코드만 분석하기 때문에 코드 전체를 분석하지는 못하지만 더 정확한 분석이 가능하다.

소프트웨어 버스마크 기법은 특히 바이너리 실행 파일에서의 도용을 탐지하기 위해서 필요하다. 지금까지 소프트웨어 버스마크 기법은 자바 클래스 파일과 같은 중간 코드 실행 파일과 x86 PE(Portable Executable) 파일과 같은 바이너리 실행 파일에 적용되어 왔다. 중간 코드 실행 파일은 실행 파일로부터 코드를 정확히 구분해 낼 수 있기 때문에 비교적 정확한 정적 버스마크를 추출할 수 있다. 반면에 바이너리 실행 파일은 코드와 데이터의 정확한 구분이 불가능하기 때문에 코드를 정확하게 복구할 수 없다[2]. 또한 패킹이나 코드 난독화(obfuscation)를 통하여 코드 자체에 대한 복구를 어렵게 하기 때문에 자동화된 방법을 통하여 정적 버스마크를 추출하기 힘들다. 이런 이유로 기존의 정적 바이너리 버스마크에 대한 연구는 패킹이나 난독화가 되지 않았다는 가정 하에서 수행되었다[3]. 악의적으로 코드 도용을 하는 경우에는 코드 자체를 숨겨 정적 분석이 어렵게 만들기 때문에 도용 탐지를 위해서 바이너리 파일에 대한 동적 버스마크가 꼭 필요하다.

현재 Windows 바이너리 실행 파일의 도용 탐지를 위해 제안되어 있는 Tamada의 동적 버스마크는 프로그램을 실행하고 바로 종료시키는 과정에서 나타나는 API 함수 호출 시퀀스로부터 버스마크를 추출하기 때문에 프로그램의 핵심 부분의 특징을 반영하지 못한다[4]. 따라서 본 논문에서는 기존의 동적 바이너리 버스마크의 단점을 보완하여 프로그램의 핵심 부분의 특징을 반영하는 기능 단위 동적 API 버스마크(FDAPI: Functional unit Dynamic API birthmark)를 제안한다. FDAPI 버스마크는 프로그램의 실행 시점이 아닌 프로그램의 중요 기능 수행 시간의 API 함수 호출 시퀀스를 분석한 결과이다. 추출된 버스마크들은 도용 탐지를 위해서 준 전체 정렬(semi-global alignment) 알고리즘을 이용하여 비교된다.

제안한 버스마크를 평가하기 위하여 다양한 프로그램에 대한 버스마크를 추출하고 비교하였다. 신뢰성(credibility)을 증명하기 위해 같은 기능을 가지는 다양한 프로그램을 비교하였다. 강인성(resilience)을 평가하기 위해서는 같은 소스파일로부터 다양한 컴파일 방법을 통해 생성된 다른 실행 파일을 비교하였다. 또한, 도용 탐지 능력을 보이기 위하여 eMule 클론들을 비교하였다.

본 논문의 구성은 다음과 같다. 제2장에서는 기존의 연구에 대해서 설명한다. 제3장에서는 본 논문에서 제안하는 기능 단위 동적 API 버스마크와 구현된 시스템을 소개한다. 제4장에서는 프로그램 비교를 통해 버스마크를 평가한다. 제5장에서는 내용을 요약하고 향후 연구에 대해 설명한다.

2. 관련 연구

실행 파일을 이용하여 코드 도용을 탐지하는 대표적인 방법에는 소프트웨어 버스마크와 소프트웨어 워터마크(software watermark)가 있다[5]. 소프트웨어 워터마크는 실행 파일 안에 저작권 및 소유권과 같이 코드 도용을 탐지해낼 수 있는 정보를 워터마크 형태로 내장시킨다. 워터마크 시스템은 실행 파일로부터 워터마크를 추출하여 도용 여부를 판정하게 된다. 소프트웨어 버스마크는 소프트웨어 워터마크와 달리 추가적인 정보를 실행 파일에 포함시키지 않고 코드 자체를 분석하여 추출한다.

소프트웨어 버스마크는 코드에 대한 분석 방법에 따라 동적 버스마크와 정적 버스마크로 분류할 수 있다.

정적 버스마크는 코드 분석을 통해 추출한다. 프로그램을 직접 실행하지 않고 분석하기 때문에 분석 과정을 자동화 할 수 있으며, 코드 전체에 대한 특징을 분석할 수 있다는 장점이 있다. 그러나 코드 난독화가 적용된 경우 코드를 실행 파일로부터 분리하기 어렵고, 분석 매우 복잡해진다는 단점을 가지고 있다.

정적 버스마크는 프로그램의 문법적인 구조만을 분석하는 버스마크와 프로그램의 의미까지 분석하는 버스마

크로 분류할 수 있다. 문법 구조만을 구조하는 버스마크로는 Tamada의 jbirth[6], Myles의 k-gram 버스마크[7], 그리고 Lim의 Java Stack Birthmark[8]가 있다. 프로그램의 의미까지 고려하는 버스마크로는 Park의 Java Trace 버스마크[9], Lim의 Control Flow Edge 버스마크, 그리고 Choi의 정적 Windows API 버스마크[3]를 들 수 있다.

동적 버스마크는 프로그램을 직접 실행하며 버스마크의 내용을 추출하기 때문에 자동화된 버스마크 추출이 어렵다. 또한 실행할 때의 환경이나 입력 값에 따라서 버스마크의 내용이 변하게 된다. 동적 버스마크의 장점은 코드 난독화에 대한 강인성, 분석 결과의 정확성이 정적 버스마크에 비해 뛰어나다는 것이다. 코드 난독화는 프로그램의 구조는 변화시키지만 의미는 변화시키지 못하기 때문에 관찰할 수 있는 결과는 쉽게 바꿀 수 없다. 따라서 관찰할 수 있는 결과를 이용하는 동적 버스마크도 쉽게 변하지 않게 된다. 또 실제 실행 결과를 이용하기 때문에 분석 내용이 정적 버스마크보다 정확하게 된다. 대표적인 동적 버스마크에는 Tamada의 동적 Windows API 버스마크[4], Myles의 Whole Program Path 버스마크[10], Schuler의 동적 Java API 버스마크[11]가 있다. 기존의 동적 버스마크의 단점은 초기화 과정에 대한 특징만 추출하거나[4], 중요한 기능에 대한 특징을 추출하지만 대화형 프로그램에 적용할 수 없고, 배치 모드 프로그램에 대해서만 동작한다는 것이다[10,11].

3. 기능 단위 동적 API 버스마크

이 장에서는 기능 단위 동적 API 버스마크를 정의하고, 버스마크의 추출 및 비교 방법을 제시하며, 실제 구현된 내용을 소개한다.

3.1 버스마크의 정의 및 추출

본 논문에서 제안하는 버스마크는 Windows 프로그램을 위한 기능 단위 동적 API 버스마크(FDAPI: Functional unit Dynamic API birthmark)이다. FDAPI는 Windows 응용 프로그램을 특정한 입력을 가지고 직접 실행하면서 관찰할 수 있는 Windows API 함수에 대한 호출을 기록하고, 그 가운데서 특정한 기능을 호출할 때의 API 함수의 호출 시퀀스를 추출한다. 논문에서 사용되는 기능은 다음과 같이 정의된다.

정의 1 : [기능]

프로그램의 기능은 프로그램에 입력 장치를 통한 사용자의 입력이나 운영체제나 네트워크와 같은 외부 환경을 통한 특정한 명령에 대해 프로그램이 수행하는 일련의 작업을 의미한다.

정의 1에 의하면 GUI 방식 프로그램에서 메뉴를 통해 선택하여 실행하는 작업들은 기능이라고 볼 수 있다. 예를 들어 워드 프로세서에서 문서를 열기 위한 파일-열기, 파일-저장 등의 작업을 기능이라고 볼 수 있다. 다른 예로 웹 서버에서 네트워크를 통해 특정한 웹 페이지를 요청할 때 해당되는 웹 페이지를 전송하는 작업을 기능이라고 볼 수 있다.

FDAPI는 프로그램의 기능을 실행하면서 Windows API 함수에 대한 호출 시퀀스로부터 버스마크를 추출한다. Windows API(Application Programming Interface)는 Windows 응용프로그램이 운영체제가 제공하는 메모리, 디스크, 네트워크, 그래픽, 사운드 등 시스템 자원에 접근할 수 있게 하는 유일한 통로이다. 대부분의 Windows 응용프로그램은 Windows API 함수를 빈번하게 호출하며, Windows API 함수 호출은 다른 것으로 대치하기가 매우 힘들기 때문에 기존의 연구들에서도 API 함수에 대한 호출을 버스마크로 이용하였다[3,4,6,9,11]. 따라서 Windows API 함수에 대한 호출 시퀀스에 대한 분석은 해당되는 응용 프로그램이 운영체제 자원에 대한 사용 방법에 대한 특징으로 그 프로그램 자체의 특징을 잘 나타낸다고 볼 수 있다.

본 논문에서는 Windows API 함수 호출 시퀀스로부터 버스마크를 추출할 때 함수 호출에 사용하는 매개변수(parameter) 정보를 포함하지 않고 함수 자체만 고려하였다. 각 함수는 구별을 위해서 알파벳 순서로 번호를 붙인다. Windows API 함수의 목록[1]은 MSDN에 정의되어 있으며 본 논문에서 사용한 Windows API 함수는 Windows Vista에서 사용하는 대부분의 API 함수로 2331개이다. 따라서 각 Windows API 함수는 1부터 2331까지 식별 번호가 붙여져 있어서 2 바이트 한 글자(character)로 인코딩 된다.

다음은 기능 단위 동적 API 버스마크에 대한 정의이다.

정의 2 : [기능 단위 동적 API 버스마크]

프로그램을 p, 프로그램에 대한 입력을 I, 비교하려는 기능을 m이라 하자. 기능 단위 동적 API 버스마크 $FDAPI(p,I,m)$은 I를 입력으로 p를 실행하는 과정에서 m이 동작될 때 호출되는 API 함수 식별 번호의 시퀀스이다.

정의 2에서 p는 일반적인 Windows 응용 프로그램, I는 프로그램 p를 실행하기 위해 필요한 모든 입력, m은 추출하고자 하는 프로그램의 특정한 기능을 나타낸다. 예를 들면, 오디오 재생 프로그램에서 mp3 파일 한 개를 재생할 때의 FDAPI를 추출할 때, p는 오디오 재생 프로그램, I는 재생하고자 하는 mp3 파일, 그리고 m은 한 곡을 재생하는 기능을 의미한다.

1) Windows API Reference. http://msdn.microsoft.com/en-us/library/aa383749(VS.85).aspx, last accessed 2009.4.12

3.2 버스마크의 비교

기능 단위 동적 API 버스마크는 API 함수 식별 번호의 시퀀스이다. 두 시퀀스를 비교하기 위하여 문자열 비교 알고리즘, 또는 생물 정보학에서 유전자 비교를 위해 이용하는 시퀀스 정렬 알고리즘을 사용할 수 있다. 대표적인 시퀀스 비교 알고리즘으로 최대 공통 부분 문자열(longest common substring), 최대 공통 부분 시퀀스(longest common subsequence), 전체 정렬(global alignment), 지역 정렬(local alignment), 준 전체 정렬(semi-global alignment) 등을 들 수 있다. 시퀀스 비교 알고리즘들은 두 개의 시퀀스에서 각각 부분 시퀀스(subsequence)를 선택하고 그 두 부분 시퀀스의 각 문자들 사이의 일치(match), 불일치(mismatch), 그리고 간격(gap)에 따라서 일치 점수(match score)를 계산한다. 시퀀스 비교의 결과는 일치 점수가 최대가 되는 부분 문자열과 그 일치 점수를 이용한 두 시퀀스 사이의 유사도(similarity)이다. 본 논문에서는 시퀀스 비교 알고리즘으로 전체 정렬 방법(global alignment)을 개량한 준 전체 정렬(semi-global alignment)을 사용한다.

전체 정렬 방법은 두 시퀀스를 비교하기 위해 시퀀스를 처음부터 끝까지 정렬시켜 유사도를 구하는 방법이다. 시퀀스 전체에 대한 일치를 시도하기 때문에 두 시퀀스 전체가 유사할 때의 비교에 적합하다. 구현을 위해서 일반적으로 다이나믹 프로그래밍(dynamic programming) 방법을 이용하는 Needleman-Wunsch 알고리즘을 이용한다.

준 전체 정렬은 전체 정렬을 개선한 방법으로 한 시퀀스가 다른 시퀀스에 포함되어 있는지를 확인할 때 유용하다. 준 전체 정렬은 기본적으로 전체 정렬과 같은 방법을 사용하지만 전단부와 후반부의 불일치에 대해 감점을 하지 않는다는 차이점이 있다. 일반적으로 도용이 일어나는 경우 원본이 되는 코드를 그대로 사용하며 그 코드 실행 전후에 추가적인 작업을 하는 코드를 추가하게 된다. 즉, 원본이 되는 코드가 도용하고 있는 코드에 포함되었다고 볼 수 있기 때문에 준 전체 정렬이 코드 도용 탐지에 적합한 알고리즘이라고 볼 수 있다.

알고리즘 1은 준 전체 정렬 방법을 이용하여 일치 점수를 구하는 방법을 기술하고 있다.

알고리즘 1 : [준 전체 정렬 일치 점수]

알고리즘 1의 입력은 길이가 각각 m, n인 시퀀스 S_1, S2이다. 문자가 일치될 때의 점수인 *match_score*를 1로 하고 문자가 불일치될 때의 점수인 *unmatch_score*를 -1로, 그리고 간격이 생길 때의 점수인 *gap_score*를 -1로 한다. 이 값들은 비교하려는 시퀀스의 특성에 따라서 정할 수 있다. 일치 점수 *match_score*를 1로 할 때 두 시퀀스의 비교에서 최대 점수는 두 시퀀스 중 더 길이가 짧은 시퀀스의 길이가 된다.

```
function semi-global-alignment-score(S1[1..m], S2[1..n])
    G = integer array of [0..m][0..n]
    match_score = 1, unmatch_score = -1, gap_score
        = -1
    for j = 0 to n
        G[0][j] = 0
    for i = 1 to m
        G[i][0] = gap_score * i
        for j = 1 to n
            if (S1[i] == S2[j])
                then Choice1 = G[i-1][j-1] +
                    match_score
                else Choice1 = G[i-1][j-1] +
                    unmatch_score
            Choice2 = G[i-1][j] + gap_score
            Choice3 = G[i][j-1] + gap_score
            G[i][j] = max(Choice1, Choice2, Choice3)
    return max(G[m][1], G[m][2], ... ,G[m][n], 0)
```

준 전체 정렬을 이용한 시퀀스의 유사도는 포함 관계를 이용하여 구할 수 있다.

정의 3 : [시퀀스 유사도]

각각 길이가 m, n인 두 시퀀스 S_1, S_2가 주어졌을 때 두 시퀀스의 유사도 $sim(S_1, S_2)$는 알고리즘 1을 이용하여 다음과 같이 정의된다.

$$sim(S_1, S_2) = \frac{semi\text{-}global\text{-}alignment\text{-}score(S_1, S_2)}{min(m,n)}$$

준 전체 정렬 점수가 최소 0에서 최대 $min(m,n)$이므로 시퀀스의 유사도의 값은 0에서 1 사이의 실수가 나온다.

도용하고자 하는 각 프로그램에서 기능 단위 동적 API 버스마크를 추출하고 각 쌍에 대해 시퀀스 유사도를 구하면 도용 여부를 확인할 수 있다. 시퀀스 유사도가 어느 정도가 될 때 두 프로그램이 도용 관계에 있는지를 결정하는 것은 프로그램의 특성에 따라 달라지기 때문에 비교하려는 프로그램 도메인에 따라 실험적으로 결정되어야 한다.

3.3 기능 단위 동적 API 버스마크 시스템

기능 단위 동적 API 버스마크를 평가하기 위하여 기능 단위 동적 API 버스마크 시스템을 구현하였다. 그림 1은 버스마크를 추출하고 비교하여 유사도를 계산하는 과정을 개략적으로 보여준다. 이러한 몇 단계의 과정을 거쳐서 버스마크를 추출하고 유사도를 구한다. 프로그램이 실행되는 동안 사용되는 API를 추출하고 기록하기

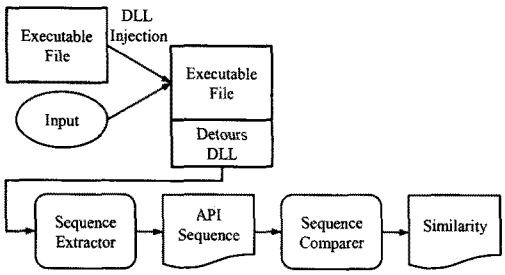

그림 1 기능 단위 동적 API 버스마크 시스템

위해 Detours 라이브러리를 사용하였다[12]. Detours는 Win32 API 함수들의 실행을 가로채 API 함수의 로그를 기록할 수 있도록 하는 기능을 제공한다. 본 논문에서는 Detours를 이용하여 필요한 부분의 API만을 기록할 수 있도록 구현하였다.

API 호출 시퀀스를 추출하기 위해서는 먼저 프로그램을 실행하고, API를 추출하고 싶은 기능을 실행하기 직전부터 기록을 시작하고 기능을 실행하고 나서 기록을 중지한다. 그림 2는 이러한 방식으로 추출된 API 시퀀스의 일부분을 나타낸 예이다. 그림에서 들여쓰기 되어 있는 함수는 API 함수에 의해 다시 호출된 API 함수를 의미한다. 들여쓰기 되어 있지 않은 첫 단계의 API 함수는 실행 파일에서 직접 호출한 함수이다. 함수들 중에 다른 함수를 간접적으로 호출하는데 사용되는 함수들은 그 함수 자체의 의미보다 간접적으로 호출되는 함수가 더 큰 의미를 가진다. 예를 들면 SendMessage 함수의 경우 다른 창 객체에 메시지를 전달하는데 그 메시지에 따라서 호출되는 API 함수의 종류가 달라진다. 이런 함수들은 실행 파일 자체에 해당 API 함수 호출이 나타나 있지 않지만 간접적으로 호출되는 API 함수들이 프로그램의 특성을 더 잘 나타내고 있기 때문에 버스마크에 그 내용이 포함되어야 한다. 따라서 본 논문에서는 첫 단계만 이용하지 않고 2, 3단계를 같이 이용한다.

```
GetParent
TranslateMessage
DispatchMessageW
  GetWindowLongW
  BeginPaint
  IsWindowVisible
  GetClientRect
    OffsetRect
  CreateCompatibleDC
  CreateCompatibleBitmap
  SelectObject
  OffsetWindowOrgEx
  IsWindowEnabled
  IsWindow
  GetParent
  GetClientRect
    OffsetRect
  MapWindowPoints
  SetPropW
  GetViewportOrgEx
  SetViewportOrgEx
  SendMessageW
    GetClassLongW
    SendMessageW
    GetClientRect
GetTickCount
  CreateRectRgnIndirect
WaitForSingleObject
  WaitForSingleObjectEx
    IsWindowVisible
GetTickCount
  GetWindowRect
```

그림 2 기능 단위 동적 API 시퀀스의 예

4. 평 가

이 장에서는 버스마크 평가를 위한 여러 가지 실험 결과에 대해 보여 준다. 실험은 네 가지 종류로 이루어진다. 먼저, 기존의 버스마크 평가 기준인 신뢰성(credibility)과 강인성(resilience)을 위한 두 가지 실험을 제시한다. 또한, 기존의 Tamada 버스마크와 비교를 위한 실험 결과를 보여 준다. 마지막으로 도용 탐지의 확인을 위한 실험과 그 결과에 대해 기술한다.

4.1 신뢰성

신뢰성은 기능이 같지만 독립적으로 구현된 프로그램에 대해 버스마크가 두 프로그램은 다르다고 판정해야 한다는 것이다. 두 프로그램을 p, q라 하고 동일한 기능을 m이라고 했을 때 입력 I가 주어지면 FDAPI를 이용한 두 프로그램 사이의 유사도는 정의 2, 정의 3에 의해

$$sim(FDAPI(p,I,m),\ FDAPI(q,I,m))$$

으로 정의된다. 즉, 같은 기능을 나타내는 두 프로그램이 있을 때, 두 프로그램이 실제로 같은 코드를 이용하여 구현되었다면 유사도가 높게 나타나야 하며, 다른 코드를 이용하여 구현되었다면 유사도가 낮게 나타나야 한다는 것이다.

제안한 버스마크의 신뢰성 평가를 위해서 이미지 뷰어와 FTP 클라이언트 프로그램들을 대상으로 선택하였다. 표 1은 신뢰성 평가를 위한 프로그램들이다. 이미지 뷰어의 기능에는 이미지 보기 기능, 파일 형식 변환, 이미지 편집, 이미지 보정, 이미지 파일 관리 등이 있다. FTP 클라이언트의 기능에는 FTP 서버 접속, 파일 업로드, 파일 다운로드, 사이트 관리 등이 있다. 이런 기능들에 대해 각각 실험을 통해 프로그램을 비교할 수 있다. 본 논문에서는 이미지 뷰어에서는 이미지 보기 기능을 이용하였고, FTP 클라이언트에서는 파일 업로드 기능을 이용하였다. 이미지 보기와 파일 업로드 기능을 선택한 이유는 이 기능들이 이미지 뷰어와 FTP 클라이언

트에서 필수적인 기능들이기 때문이다. 또한 이 기능들은 다른 기능들에 비하여 사용자 인터페이스에 의한 영향이 적기 때문에 API 호출 시퀀스에 대한 노이즈가 줄어들어 정확한 버스마크를 추출하기에 유리하다.

이미지 뷰어는 프로그램을 실행하고 이미지 목록 창이 보이는 상태에서 API 호출 기록을 시작하고 실험을 위해 마련한 작은 이미지[2]를 읽고 화면에 출력하고 나서 기록을 종료하였다. FTP 클라이언트는 프로그램을 실행하고 서버에 접속한 상태에서 API 호출 기록을 시작하고 작은 파일 하나를 업로드한 뒤 기록을 종료하였다.

신뢰성 실험은 독립적으로 개발되었지만 같은 기능을 가진 프로그램들 사이의 비교와, 같은 코드를 공유하지만 약간의 버전 차이가 있는 같은 프로그램들 사이의 비교의 두 가지로 이루어진다.

약간 업그레이드 된 프로그램은 원래 프로그램과 코드를 대부분 공유하기 때문에 신뢰성 있는 버스마크를 이용하여 비교했을 때 유사도가 높게 나타나야 한다. 표 2는 각 프로그램의 서로 다른 버전 사이의 비교 결과이다. 표 2에서 동일한 실행 파일을 수행한 결과의 비교에서도 유사도가 1보다 낮게 측정된 이유는 실행되는 API 시퀀스가 실행 환경에 따라서 수행 순서에서 차이가 날 수 있기 때문이다. 이런 경우는 UI와 관련된 컴포넌트들을 사용할 경우에 자주 발생하는데, 예를 들어 창이 다른 창과 겹치는 경우 repaint가 호출되면서 API가 중간에 많이 들어오게 된다. 실험 결과 서로 같은 프로그램간의 비교나 약간의 업그레이드가 일어난 버전들 사이의 비교에서 모두 최소 0.769 이상의 유사도를 보이고 있다. 이 결과를 통해 볼 때 기능 단위 동적 API 버스마크가 같은 코드를 사용하고 있는 프로그램들의 유사도를 비교적 정확하게 측정하는 것을 볼 수 있다.

이미지 뷰어들은 이미지 보기 기능을 공통적으로 가지고 있지만 독립적으로 구현되었다. 따라서 신뢰성이 높은 버스마크를 통해서 이미지 뷰어들을 비교했을 때 유사도가 낮아야 한다. FTP 클라이언트들도 파일 업로드 기능을 공통적으로 가지고 있지만 독립적으로 구현되었다. 따라서 신뢰성이 있는 버스마크를 통해 FTP 클라이언트들을 비교한다면 유사도가 낮아야 한다. 표 3은 기능이 같은 프로그램들을 비교한 결과이다. 표 3에서 독립적으로 개발된 모든 프로그램에 대한 유사도는 0이다. 여기서 볼 수 있는 것처럼 기능 단위 동적 API 버스마크는 서로 다른 프로그램들을 정확하게 구별하는 것을 알 수 있다. 표 2와 표 3의 두 가지 실험 결과는 FDAPI가 신뢰성을 가지고 있다는 것을 보여 준다.

2) http://en.wikipedia.org/wiki/Lenna

표 1 신뢰성 평가를 위한 프로그램

Image Viewer		FTP Client	
Name	Version	Name	Version
ACDSEE (acdsee)	4.0.1 4.0.2	FileZilla (fz)	3.10 3.15
FastStone Image Viewer(fsview)	3.4 3.5	Leapftp (lftp)	2.71 2.76
XnView(xnview)	1.21 1.25	Freshftp (freshftp)	4.95 5.0

표 2 같은 프로그램의 두 버전간의 유사도

Image Viewer (I = 'lena.jpg', m = 이미지보기)		FTP Client (I = 'readme.txt', m = 파일업로드)	
Program 1 (P_1) / Program 2 (P_2)	sim.	Program 1 (P_1) / Program 2 (P_2)	sim.
acdsee 4.0.1 / 4.0.1	0.852	freshftp 4.95 / 4.95	0.833
acdsee 4.0.1 / 4.0.2	0.871	freshftp 4.95 / 5.0	0.843
acdsee 4.0.2 / 4.0.2	0.910	freshftp 5.0 / 5.0	0.848
fsview 3.4 / 3.4	0.784	fz 3.10 / 3.10	0.962
fsview 3.4 / 3.5	0.769	fz 3.10 / 3.15	0.957
fsview 3.5 / 3.5	0.802	fz 3.151 / 3.15	0.974
xnview 1.21 / 1.21	0.977	lftp2.71 / 271	0.730
xnview 1.21 / 1.25	0.981	lftp2.71 / 276	0.780
xnview 1.25 / 1.25	0.990	lftp2.76 / 276	0.875

표 3 같은 기능을 가지는 프로그램 유사도

Image Viewer (I = 'lena.jpg', m = 이미지보기)		FTP Client (I = 'readme.txt', m = 파일업로드)	
Program 1 (P_1) / Program 2 (P_2)	sim.	Program 1 (P_1) / Program 2 (P_2)	sim.
acdsee4.0.1/ fsview3.4	0.0	freshftp4.95/ fz3.10	0.0
acdsee4.0.1/ fsview3.5	0.0	freshftp4.95/ fz3.15	0.0
acdsee4.0.1/ xnview1.21	0.0	freshftp4.95/ lftp2.71	0.0
acdsee4.0.1/ xnview1.25	0.0	freshftp4.95/ lftp2.76	0.0
acdsee4.0.2/ fsview3.4	0.0	freshftp5.00/ fz3.10	0.0
acdsee4.0.2/ fsview3.5	0.0	freshftp5.00/ fz3.15	0.0
acdsee4.0.2/ xnview1.21	0.0	freshftp5.00/ lftp2.71	0.0
acdsee4.0.2/ xnview1.25	0.0	freshftp5.00/ lftp2.76	0.0
fsview3.4/ xnview1.21	0.0	fz3.10/ lftp 2.71	0.0
fsview3.4/ xnview1.25	0.0	fz3.10/ lftp 2.76	0.0
fsview3.5/ xnview1.21	0.0	fz3.15/ lftp 2.71	0.0
fsview3.5/ xnview1.25	0.0	fz3.15/ lftp 2.76	0.0

4.2 강인성

코드를 도용하는 사람들은 일반적으로 도용 탐지를 회피하기 위해 소스 코드를 다른 방식으로 컴파일하거나 코드 난독화(code obfuscation)와 같은 프로그램 변환(program transformation)을 수행한다. 강인성은 버스마크의 도용 탐지 능력이 프로그램이 변환되었을 때

얼마나 잘 유지되는지를 나타낸다. 원본 프로그램을 p, p에 프로그램 변환 T를 수행하여 컴파일 한 코드를 p'이라 하자. 비교하려는 동일한 기능 m과 입력 I에 대해 FDAPI를 이용한 두 프로그램 사이의 유사도는 정의 2, 정의 3에 의해

$$sim(FDAPI(p,I,m),\ FDAPI(T(p),I,m))$$

이 된다. 강인성이 있는 버스마크는 원본 프로그램과 변환된 프로그램을 비교했을 때 높은 유사도를 얻을 수 있어야 한다.

본 논문에서는 강인성 평가를 위해 현실적으로 쉽게 적용할 수 있는 컴파일러 변경과 컴파일 옵션 변경을 이용하였다. 실험 대상으로 다루고 있는 프로그램은 C/C++로 작성된 Windows 프로그램이다. 현재 C/C++로 작성된 Windows 프로그램을 난독화할 수 있는 유일한 도구로 CloakWare Security Suite[3]가 있다. CloakWare Security Suite는 난독화 과정에서 제어 흐름 난독화를 수행한다[13]. 프로그램에 제어 흐름 난독화를 적용한다 하더라도 API 함수 호출 순서는 유지되기 때문에 FDAPI에는 영향을 줄 수 없다. 따라서 본 실험에서는 API 함수의 사용과 순서에 영향을 줄 수 있는 컴파일러 변경과 옵션 변경에 대하여만 실험한다.

강인성 실험을 위해서 오픈 소스 이미지 뷰어인 JPEGView를 사용하였다. 컴파일러는 Microsoft Visual C++ 8.0과 9.0 버전을 사용하였으며, 컴파일 옵션은 Release 모드와 Debug 모드를 사용하였다. API 시퀀스 수집은 이미지를 화면에 띄우기 직전부터 기록을 시작하고 이미지를 띄운 직후에 기록을 종료해서 그 사이에 호출된 API 시퀀스를 저장하여 각 시퀀스 조합에 대해 비교 실험을 진행하였다.

표 4는 강인성 평가를 위한 실험 결과이다. 실험 결과 FDAPI는 제시한 컴파일러와 옵션 변경을 통한 프로그램 변환에 대해 강인성을 가지고 있음을 보여 준다.

표 4 컴파일러와 옵션 변경에 따른 JPEGView 유사도 (I = 'lena.jpg', m = 이미지보기)

Program 1 (P_1)	Program 2 (P_2)	similarity
jv.vc8.debug	jv.vc8.debug	1.0
jv.vc8.debug	jv.vc8.release	1.0
jv.vc8.debug	jv.vc9.debug	1.0
jv.vc8.debug	jv.vc9.release	1.0
jv.vc8.release	jv.vc8.release	1.0
jv.vc8.release	jv.vc9.debug	1.0
jv.vc8.release	jv.vc9.release	1.0
jv.vc9.debug	jv.vc9.debug	1.0
jv.vc9.debug	jv.vc9.release	1.0

3) Cloakware Security Suite. http://security.cloakware.com/products/software-security.php, last accessed 2009.4.16

4.3 기존의 버스마크와의 비교

대표적인 기존의 동적 버스마크에는 Myles의 Whole Program Path 버스마크, Schuler의 자바 API 버스마크, 그리고 Tamada의 Windows 버스마크가 있다.

Myles의 Whole Program Path 버스마크는 자바 프로그램을 특정한 입력을 가지고 실행하며 제어 흐름 그래프(control flow graph)의 기본 블록(basic block) 단위의 트레이스(trace)를 기록하고 SEQUITUR 알고리즘을 적용하여 DAG(direct acyclic graph) 형태인 문맥 자유 문법(context free grammar)으로 변환시킨다. 그리고 프로그램들을 비교할 때는 각 프로그램에 해당되는 DAG를 MCS(maximal common subgraph) 알고리즘으로 비교한다. MCS의 시간 복잡도가 각 DAG 노드 개수를 $|v_p|$와 $|v_q|$라 했을 때 $O(|v_p|\ |v_q|^3)$가 된다. 노드 개수가 같다고 했을 때 $O(n^4)$ 알고리즘이라고 볼 수 있다. 따라서 WPP 버스마크는 시간 복잡도의 문제 때문에 작은 프로그램 에만 적용할 수 있다는 단점이 있다. 실제로 Myles의 실험에 사용된 가장 큰 클래스 파일은 6KB 정도이므로 실제 응용 프로그램에 적용할 수 없다.

Schuler의 자바 API 버스마크는 프로그램을 특정한 입력을 가지고 실행하면서 각 클래스별로 호출되는 API 메소드의 시퀀스를 기록한다. 프로그램을 비교할 때는 수집된 시퀀스를 통해 추출한 k-gram 집합의 유사도를 측정한다. Schuler의 논문에서는 이 방법을 이용하여 잘 알려진 이미지(PNG 포맷) 리더와 XML 파서 라이브러리들을 비교하였다. 실험 대상 라이브러리들은 현실적으로 사용될 수 있는 큰 규모(3MB)에서도 실험이 가능하였다. 이 방법은 기본적으로는 프로그램 전체를 실행하기 때문에 배치 처리 방법의 프로그램을 비교하는데 적합하다. 그러나 Schuler의 논문에서는 라이브러리의 도용 탐지 방법도 제시하고 있다. 라이브러리 도용 탐지를 위해서 이미지 리더 라이브러리와 해당 라이브러리를 사용하는 이미지 뷰어들을 비교하였다. 라이브러리가 이미지 뷰어에 사용되었을 때는 유사도가 67% 이상, 사용되지 않은 경우는 유사도가 36% 이하라는 결과가 나왔다는 것을 보여 준다. 이 수치는 라이브러리간의 비교에서 80% 이상의 정확한 수치에 비해서 정확성이 떨어진 것을 보여준다. 이런 부정확성은 Schuler의 버스마크가 프로그램 전체에 대한 API 시퀀스를 추출하기 때문에 초기화(initialize) 및 종료(finalize) 과정에 대한 노이즈가 포함되기 때문이다. 반면 FDAPI는 초기화 및 종료 과정에 대한 노이즈를 제외하고 기능에 대해서만 버스마크를 추출하여 정확성이 높아지게 된다.

기존의 자바 버스마크들은 플랫폼의 차이 때문에 FDAPI와의 비교 실험은 힘들다. 따라서 본 논문에서는 직접 비교가 가능한 Tamada의 Windows 버스마크와

비교 실험을 수행하였다. Tamada의 버스마크는 먼저 프로그램의 시작 부분의 API 시퀀스에서 전체 API 시퀀스의 열을 추출한다. EXESEQ는 추출된 API 시퀀스 열들을 최대 공통 문자열(LCSTR: Longest Common Substring) 알고리즘을 이용하여 비교한다. EXEFREQ는 각 API의 호출 빈도(frequency)를 측정한다. 여기서는 직접 비교 가능한 시퀀스 버스마크인 EXESEQ를 FDAPI와 비교한다.

Tamada 시퀀스 버스마크의 약점은 프로그램 시작 부분에 대한 특징만 추출한다는 것이다. 따라서 시작 부분은 같고 중요한 내용을 처리하는 부분이 다르다 해도 두 프로그램을 같다고 판정하게 된다. 그러나 FDAPI는 중요한 부분을 실행할 때 특징을 추출하기 때문에 시작 부분이 같다 하더라도 두 프로그램을 구분할 수 있어야 한다. 이 점을 확인하기 위하여 잘 알려진 그래픽 라이브러리인 CxImage[4]와 CImg[5]를 이용하여 간단한 이미지 뷰어 프로그램 demo-cx와 demo-cimg를 개발하였다. 두 프로그램은 Visual C++ MFC를 이용하여 공통된 GUI를 가지며 이미지 처리를 위해서만 이미지 라이브러리를 각각 호출한다.

실험은 demo-cx와 demo-cimg를 각각 실행하여 초기화 과정의 시퀀스와 이미지를 로드하고 표시하는 이미지 보기 과정의 시퀀스를 추출한다. 시퀀스의 비교는 초기화 시퀀스를 LCSTR 알고리즘으로 비교하는 EXESEQ 유사도를 계산하고, 이미지 보기 과정의 시퀀스를 준 전체 정렬 방식으로 비교하는 FDAPI(m=이미지 보기) 유사도를 측정한다. 또한, FDAPI와 EXESEQ의 유사도 결과의 차이가 기능 단위로 추출했기 때문인지, 아니면 시퀀스 비교 방법의 차이 때문인지에 대한 원인 분석을 위하여 초기화 시퀀스들을 준전체 정렬 방식으로 비교하는 FDAPI(m=초기화) 유사도를 측정하였다. 실험은 모두 5회 실행되어 평균 유사도를 측정했다.

표 5는 두 프로그램의 EXESEQ, FDAPI(m=이미지 보기), FDAPI(m=초기화) 유사도 결과를 보여 준다. 같은 프로그램 사이의 비교에서는 두 버스마크 모두 다 0.8 이상의 높은 유사도를 나타낸다. 같은 프로그램간의 유사도에 차이가 나는 이유는 실행 환경에 따라서 API 호출에 변화가 생길 수 있기 때문이다. 서로 다른 두 프로그램 사이의 결과는 FDAPI(m=이미지보기)의 경우 0.407로 같은 프로그램 사이의 유사도의 1/2 아래의 수치를 보여 주었으나 EXESEQ는 0.774로 같은 프로그램 사이의 유사도인 0.802와 별로 차이가 나지 않는 결과를 보여 준다. 또한 EXESEQ와 같은 시퀀스를 비교하는 FDAPI(m=초기화) 유사도의 결과는 같은 프로그램에서 0.923과 0.987로 높은 유사성을 나타내고 있기는 하지만 다른 프로그램 사이의 유사도도 0.818로 나타나 같은 프로그램 사이의 유사도가 0.1 정도로 미세한 차이만 나는 결과를 보여 준다. 이 결과는 EXESEQ에 비교 알고리즘만 개선한다고 하여 좋은 결과가 나타나는 것이 아니라 버스마크 추출 자체를 기능 단위로 해야 정확한 결과가 나타날 수 있다는 것을 보여준다.

이 실험을 통하여 FDAPI가 EXESEQ에 비해 실제 기능에 대한 특징을 구별할 수 있다는 사실을 확인할 수 있다. 완벽하게 구별하지 못한 이유는 이미지 라이브러리 호출 과정만 완벽하게 추출하지 못하였고 이미지를 로드하고 실행하는 부분에서 이미지가 표시되기까지의 전 과정을 추출하여 GUI 관련 컴포넌트가 호출되기 때문이다.

표 5 Tamada의 시퀀스 버스마크와 기능단위 동적 API 시퀀스 버스마크의 비교 실험 결과(I = 'lena.jpg')

Program1 (P_1)	Program2 (P_2)	FDAPI similarity (m=이미지 보기)	EXESEQ similarity	FDAPI similarity (m=초기화)
demo-cimg	demo-cimg	0.974	0.802	0.923
demo-cimg	demo-cx	0.407	0.774	0.818
demo-cx	demo-cx	0.904	0.885	0.987

4.4 도용 탐지 확인

좀 더 실제적인 평가를 위해 실제 코드 도용관계에 있다고 알려진 프로그램을 찾아 실험하였다. eMule은 ED2K 프로토콜을 지원하는 P2P 파일 전송 프로그램이다. Donkeyhote는 eMule의 소스를 이용해서 커스터마이즈한 클론 소프트웨어이다. 비교할 수 있는 기능들은 서버 접속, 접속 끊기, 파일 다운로드, 업로드 등 여러 가지가 있다. 파일 다운로드 및 업로드 등은 네트워크 트래픽 및 파일을 가지고 있는 클라이언트들에 따라서 버스마크가 쉽게 변하기 때문에 여기에서는 최초에 서버로 접속하는 기능의 버스마크를 비교하였다.

표 6은 Donkeyhote와 eMule의 서버 접속 기능을 비교한 실험 결과이다. 서버 접속의 입력이 되는 서버 목록은 기본적으로 주어지는 'server.met' 파일을 사용하였다. 자기 자신과 비교한 결과는 0.7 이상의 높은 수치가 나왔다. Donkeyhote와 eMule의 유사도는 0.684로 측정되었다. 이 수치는 자기 자신과 비교한 0.7과 거의 차이가 없는 수치이다. 이 결과는 FDAPI가 도용 관계에 있는 프로그램에 대한 탐지 능력이 있다는 것을 뒷받침하고 있다.

4) CxImage. http://www.codeproject.com/KB/graphics/cximage.aspx
5) CImage. http://cimg.sourceforge.net/

표 6 donkeyhote와 eMule의 서버 접속 기능 비교
(I = 'server.met', m = 서버 접속)

Program 1	Program 2	similarity
eMule 4.5b	eMule 4.5b	0.823
Donkeyhote 2.4	eMule 4.5b	0.684
Donkeyhote 2.4	Donkeyhote 2.4	0.723

5. 결론 및 향후 연구

버스마크는 프로그램의 고유하고 내재적인 특징을 추출한 것으로서 소프트웨어의 도용 탐지에 사용될 수 있다. 기존의 동적 버스마크들은 프로그램 전체의 도용을 탐지할 수 있었다. 본 논문에서는 특정 기능 단위의 도용을 탐지할 수 있는 Windows 프로그램을 위한 동적 API 시퀀스 버스마크를 제안한다. 특히 기존의 Windows 동적 버스마크가 프로그램을 시작했다가 곧바로 종료하는 방법으로 만들어져 있기 때문에 프로그램의 실제 기능의 특징을 반영하지 못했던 점을 보완하도록 고안되었다.

우리는 제안한 버스마크를 평가하기 위해 신뢰성과 강인성에 대해 실험하였고, 기존의 동적 Windows 버스마크와 비교 실험을 수행하였으며, 실제로 우리의 버스마크를 이용하여 도용 여부를 판별할 수 있는지에 대한 실험을 수행하였다. 실험에는 이미지 뷰어, FTP 클라이언트, 그리고 P2P 프로그램들을 사용하였다. 실험 결과 우리의 버스마크는 신뢰성, 강인성이 있으며, 도용 탐지에도 활용될 수 있음을 보였다.

향후 연구로 실행 환경의 영향으로 나타나는 노이즈를 줄이기 위한 방법, 동등한 기능을 하는 다른 API를 사용하도록 컴파일 되는 경우에도 판정을 할 수 있도록 API 함수 비교를 위해 적절한 유사도를 선택할 수 있는 방법, 컴파일러에 의해 자동적으로 생성되는 API 호출 코드 등을 필터링 하는 방법 등이 필요할 것이다.

참 고 문 헌

[1] G. Myles and C. S. Collberg, "Software theft detection through program identification," PhD. thesis. University of Arizona, 2006.

[2] C. Linn and S. Debray, "Obfuscation of executable code to improve resistance to static disassembly," in *Proceedings of the 10th ACM conference on Computer and communications security*, pp.290-299, 2003.

[3] S. Choi, H. Park, H. Lim, and T. Han, "A static birthmark of binary executables based on API call structure," *Lecture Notes in Computer Science*, vol.4846, pp.2-16, 2007.

[4] H. Tamada, K. Okamoto, M. Nakamura, A. Monden, and K. Matsumoto, "Dynamic software birthmarks to detect the theft of windows applications," *Proc. International Symposium on Future Software Technology*, pp.20-22, 2004.

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

[6] H. Tamada, M. Nakamura, A. Monden, and K. I. Matsumoto, "Java Birthmarks-Detecting the Software Theft," *IEICE Transactions on Information and Systems*, pp.2148-2158, 2005.

[7] G. Myles and C. Collberg, "K-gram based software birthmarks," Proceedings of the 2005 ACM symposium on Applied computing (SAC'05), 2005.

[8] Hyun-il Lim, Heewan Park, Seokwoo Choi, and Taisook Han, "Detecting Theft of Java Applications via a Static Birthmark Based on Weighted Stack Patterns," *IEICE Trans. On Information and Systems*, vol.91, no.9, September 2008.

[9] Heewan Park, Seokwoo Choi, Hyun-il Lim, and Taisook Han, "Detecting Java Theft Based on Static API Trace Birthmark," *Third International Workshop on Security (IWSEC 2008)*, LNCS 5312-0121, November 25-27, 2008.

[10] Ginger Myles and Christian S. Collberg, Detecting software theft via whole program path birthmarks, In *Proc. of the 7th Int. Conf. on Information Security*, vol.3225 of LNCS, Springer, pp.404-415, 2004.

[11] D. Schuler, V. Dallmeier, and C. Lindig, "A Dynamic Birthmark for Java," in *Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering*, 2007.

[12] G. Hunt and D. Brubacher, "Detours: Binary interception of Win32 functions," *Proceedings of the 3rd USENIX Windows NT Symposium*, pp. 135-143, 1999.

[13] C. Wang, "A security architecture for survivability mechanisms," PhD thesis. University of Virginia, 2000.

최 석 우

1998년 KAIST 전산학과 학사. 2000년 KAIST 전자전산학과 전산학전공 석사 2009년 KAIST 전자전산학부 전산학전공 박사. 2009년~현재 KAIST 전산학과 위촉연구원. 관심분야는 이진 프로그램 분석, 프로그램 포렌식스, 컴파일러

조 우 영

2003년~2005년 (주)네오위즈 소프트웨어 엔지니어. 2007년 KAIST 전자전산학과 전산학전공 학사. 2009년 KAIST 전자전산학과 전산학전공 석사. 2009년~현재 (주)티맥스코어 선임연구원. 관심분야는 프로그래밍 언어, 컴파일러, 프로그램 분석

한 태 숙

1976년 서울대학교 전자공학과 학사. 1978년 KAIST 전산학과 석사. 1990년 Univ. of North Carolina at Chapel Hill 박사 1991년~현재 한국과학기술원 전산학과 교수. 관심분야는 프로그래밍 언어론, 함수형 언어, 임베디드 시스템 설계 및 분석