논문 15-40-04-12 The Journal of Korean Institute of Communications and Information Sciences '15-04 Vol.40 No.04
http://dx.doi.org/10.7840/kics.2015.40.4.709

자바 자동 식별자 리네이밍 기법 및 보호 방법

김 지 윤◆, 홍 수 화*, 고 남 현**, 이 우 승***, 박 용 수°

Java Automatic Identifier Renaming Technique and Protection Method

Ji-yun Kim◆, Soo-hwa Hong*, Nam-hyeon Go**, Woo-Seung Lee***, Yong-su Park°

요 약

본 논문은 자바 언어로 작성된 코드에 선언된 임의의 변수에 관하여, 해당 변수가 사용되는 행위를 기반으로 적절한 이름을 붙여주는 리네이밍 서비스와 이러한 분석 기술에 대응하는 보안 서비스를 소개한다. 소개하는 리네이밍 서비스는 API 기반과 반복문 내부 조건문 기반의 2가지 방법으로 구분된다. 본문에서 제안 기법의 알고리즘과 함께 알려진 자바 난독화 기술과 도구를 다루어 독자의 이해를 돕고, 프로토타입을 구현하여 실용성을 보였다. 프로토타입을 이용한 실험 결과 73%의 변수명 리네이밍 성공률을 보였다. 제안 기법을 활용하면, 공동 작업자가 직관적으로 코드 전체를 파악할 수 있도록 도울 수 있다. 또한, 악성코드 분석가가 변수명을 통하여 행위를 예측할 수 있어 분석에 도움을 줄 수도 있다. 하지만, 자바로 개발한 어플리케이션의 소스코드에 제안 기법을 적용하면, 해커에게 쉽게 노출될 수 있다. 따라서 자바 어플리케이션의 코드를 보호하는 방법도 소개한다.

Key Words : Java, renaming, deobfuscation

ABSTRACT

This paper introduces a proper renaming service using variable action and security services against the analysis techniques in Java code. The renaming service that is introduced is separated into API pattern and loop condition. We present our scheme algorithm with known Java obfuscation techniques and tools in order to help readers understanding, and implement prototype to prove practicality in this paper. Test result using prototype shows 73% successful variable renaming rate. Using our scheme, cooperators can intuitionally understand all of code. Also, It helps malware analysts to predict malware action by variable name. But application source code that is developed by Java is exposed to hackers easily using our scheme. So we introduce Java application code protection methods, too.

Ⅰ. 서 론

객체 지향 프로그래밍 언어 중 하나인 자바 언어는 모바일 운영체제 안드로이드의 보급과 함께 대표적인 프로그래밍 언어로 자리 잡았다. 이는 안드로이드 운영체제의 어플리케이션이 자바 언어로 개발되기 때문

※ 본 연구는 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2012R1A1A2007263).
◆ First Author : Hanyang University Division of Computer Science & Engineering, kjycslab@gmail.com, 학생회원
° Corresponding Author : Hanyang University Division of Computer Science & Engineering, yongsu@hanyang.ac.kr, 종신회원
* Hanyang University Division of Computer Science & Engineering, ghdtnghk@naver.com
** Korea Open National University Department of Computer Science, isc7981@naver.com
*** Hanyang University Division of Computer Science & Engineering, whotname@gmail.com
논문번호 : KICS2015-03-065, Received March 23, 2015; Revised April 10, 2015; Accepted April 10, 2015

이다[1,2]. 시장 조사 기관 IDC에 따르면 안드로이드 운영체제의 2014년 선적량 기준 시장 점유율은 81.5%로 세계 1위이자 압도적인 시장 지배력을 가지고 있다[3]. 따라서 자바 언어는 앞으로도 널리 사용될 것으로 예상된다.

이러한 자바 언어로 작성된 프로그램은 소스코드를 컴파일할 때 바이트코드로 변환된 파일로 저장된다. 그리고 저장된 파일이 실행될 때 자바 가상 머신에 의해 기계어로 변환된다. 이처럼 중간언어 역할을 하는 바이트코드는 분석자가 쉽게 코드 내용을 파악할 수 있는 형태로 구성되어 있다. 이러한 특징 때문에 바이트코드를 다시 자바 언어로 변환하는 도구가 널리 활용되고 있다. 이 때문에 자바로 작성된 프로그램의 소스코드나 저작권 보호를 위하여 각종 난독화 기법과 도구가 개발되었고, 널리 활용되고 있다[2].

자바 언어가 보급됨에 따라 자바 언어로 만들어진 악성코드도 증가하고 있다. 자바 언어로 만들어진 악성코드 또한 바이트코드 형태로 저장되기 때문에 분석가가 쉽게 분석할 수 있다. 따라서 많은 악성코드 제작자들이 분석을 어렵게 만들기 위하여 각종 난독화 기법을 적용한다. 악성코드 분석가들은 자바로 작성된 악성코드를 디컴파일러로 쉽게 자바 언어로 변환하여 분석할 수 있다. 그러나 악성코드 제작자들이 적용한 난독화 기법에 의하여 분석에 어려움을 겪거나, 바이트코드에 원래 작성된 변수명에 관한 정보가 없기 때문에 디컴파일러가 변수명을 복구해주지 못해서 어려움을 겪는다. 따라서 이와 같은 어려움을 해결하기 위한 보안 서비스를 소개하며, 이 기술의 악용 가능성과 그에 대한 대비책도 소개한다.

본 연구를 요약하면, 첫째로 자바 언어로 작성된 소스코드의 변수명을 해당 변수가 소스코드에서 활용되는 역할에 알맞게 변환하는 두가지 방법을 제안한다. 두가지 방법으로 API 기반 방법과 반복문 내부의 조건문을 기반으로 하는 방법이 있다. 다음으로 본 기술이 저작권 침해에 악용될 경우에 대비하기 위한 서비스를 제안한다. 제안 기법을 활용하면, 대표적으로 2가지 장점이 있다. 우선, 악성코드 분석가는 악성코드의 행위 정보를 변수명을 통해 더 쉽게 확인할 수 있다. 또한, 공동 개발 작업에 본 연구 결과를 활용할 경우에 공동 개발자가 타인이 개발한 모듈의 소스코드를 더 쉽게 이해할 수 있다.

본 논문은 2장 관련 연구에서 바이트코드의 디컴파일 방법과 자바 코드 난독화 기법, 난독화 도구 그리고 기존의 자바 스크립트 코드 변수명 변경 연구를 소개하고, 3장에서 제안 기법과 프로토타입 구현 과정 및 방법을 설명한다. 그리고 4장에서 실험 결과를 보이며, 5장 결론 및 고찰에서 마무리한다.

Ⅱ. 관련 연구

2.1 자바의 중간 언어인 바이트코드 디컴파일 방법

컴파일이 고급 프로그래밍 언어를 중간 언어나 기계어로 바꾸는 과정이라면, 디컴파일은 중간 언어나 기계어를 고급 프로그래밍 언어로 변환하는 과정을 의미한다. 즉, 디컴파일러는 변환된 소스코드를 다시 원래 상태의 소스코드로 되돌려 소스코드를 완전히 복원하는 것을 목표로 한다. 성능이 뛰어난 자바용 디컴파일러는 디컴파일된 소스코드를 약간의 수정만으로 다시 컴파일이 가능할 정도로 복원이 가능하며, 대부분의 바이트코드는 디컴파일러에 의하여 성공적으로 소스코드로 복원 가능하다[4].

널리 활용되는 자바 디컴파일러 중 하나는 JAD이다. 공식 웹사이트에서 jad.zip파일을 받고 압축을 풀고 jad.exe 파일을 실행해 설치를 하면 이용할 수 있다. JAD를 이용하여 example1.class 파일을 디컴파일하는 과정은 jad example1.class라는 커맨드를 입력하면 된다. 이 커맨드 입력의 결과로 example1.jad라는 디컴파일된 파일이 현재 디렉토리에 생성된다[5].

2.2 자바 코드 난독화 기법

자바 언어로 작성된 소스코드를 컴파일하면 바이트코드 형태로 변환되어 파일로 저장된다. 저장된 바이트코드를 소스코드로 복원하여 분석하는 것을 막기 위하여 각종 난독화 기법이 활용되는데, 이러한 난독화 기법은 소스코드의 전체적인 의미는 변하지 않지만 소스코드를 복원하는 사람이 그 소스코드를 쉽게 알아볼 수 없도록 해주는 작업을 의미한다. 바이트코드에 맞춤화하여 활용되고 있는 주요 난독화 기술로 식별자 변환(Renaming), 제어흐름 변환(Control Flow), 문자열 난독화(String Encryption), API 은닉(API Hiding)과 클래스 난독화(Class Encryption) 등이 있다. 이 기법들은 현재 다양한 자바 언어로 작성된 어플리케이션에 널리 이용되고 있으며, 자바 언어를 활용하는 안드로이드 어플리케이션에도 널리 적용되고 있다. 하지만 난독화된 소스코드도 결국 원래의 소스코드와 전체적인 의미는 같기 때문에 분석자는 시간이 걸리더라도 소스코드의 행위 관계를 유추할 수 있다. 따라서 현재에도 도구 개발사들과 학계에 의하여, 보안성을 높이기 위하여 다양한 기법들이 개발되고 있다[4].

2.2.1 식별자 변환

난독화 기법 중에서 가장 널리 사용되고 있는 식별자 변환 기법은 각 식별자들을 실제와 다른 의미이거나 무의미한 이름 또는 알아보기 힘든 이름으로 변경하여 분석을 방해하는데 목적이 있다[4].

2.2.2 제어흐름 난독화

프로그램의 실행결과는 같아도 실행하는 과정이나 흐름을 복잡하게 만드는 작업이 제어흐름 변환이다. 프로그램은 실행의 목적이 같더라도 다양한 프로그램의 흐름이 존재한다. 예를 들어, 메소드를 호출할 때 호출할 수도 있고 그 메소드를 메소드 호출로 실행시키지 않고 프로그램 내부에 넣을 수도 있다. 이러한 작업을 인라인이라고 한다. 또는 인라인과 반대로 아웃라인을 하는 방법도 존재한다. 제어흐름 난독화를 세부적으로 분류한다면 Aggregation, Ordering, Computation이 존재한다. 우선, Aggregation 변환은 위에 언급한 인라인, 아웃라인 등의 함수 호출 방법을 변경하는 것이다. Ordering 변환은 연산을 수행하기 위한 순서를 바꾸는 것이다. 다시 말하면, 증가 카운트를 감소 카운트로 바꾸거나 여러개의 루프를 한 개로 합치거나 해서 흐름을 바꾸는 기법이며, Computation 변환은 실행되지 않거나 쓸모없는 죽은 코드를 프로그램에 집어넣거나 알고리즘을 바꾸어 흐름을 복잡하게 해주는 기법이다[6-8].

2.2.3 문자열 난독화

문자열 난독화 기법은 소스코드에 포함된 문자열을 난독화하는 기법이다. 즉, 바이트코드에 존재하는 문자열을 암호화하거나 원본 문자열로 복구할 수 있는 어떤 것으로 대체한다. 이 기법을 적용하면 분석자의 코드 이해도를 낮춰 분석을 어렵게 만드는 효과가 있다. 다른 특징으로 실행 시점에는 결국 원본 문자열로 복구를 해야 하기 때문에 복구 루틴이 소스코드에 포함되어 있다는 점이 있다[9].

2.2.4 API은닉

API 은닉 기법은 문자열 난독화와 유사한 기법으로 민감한 라이브러리나 메소드를 감추기 위해 활용된다. 자바에서 API 호출을 위하여 활용하는 디스크립터(descriptor)를 난독화하여 API를 은닉한다. 난독화한 디스크립터를 정확하게 복호화하여 원래 목적대로 API를 호출하기 위해서 자바 리플렉션 API가 활용된다[9].

자바 리플렉션 API의 본래 용도는 자바 가상 머신에서 구동되는 어플리케이션의 런타임 동작을 검사하거나 조작하려는데 사용되는 것이다. 이러한 특징을 가진 자바 리플렉션 API는 강력한 기능을 가지고 있어 어플리케이션을 원래 동작과 다른 방향으로 동작하게 하거나 정상적인 사용을 불가능하게 만들 수 있다는 점을 주의해야 한다[10,11].

2.2.5 클래스 난독화

클래스 난독화에는 대표적으로 class coalescing, class splitting, type hiding 난독화 기법이 있다. 우선, Class coalescing 난독화는 프로그램의 두 개 이상의 클래스를 하나의 클래스로 만드는 프로그램 변환이다. 극단적으로, 이 난독화는 프로그램에 사용되는 모든 클래스를 하나의 클래스로 바꿀 수 있다. 말 그대로, 객체지향 프로그램을 절차적 프로그램으로 난독화한다.

둘째, Class splitting 난독화는 프로그램의 하나의 클래스를 수많은 클래스로 만드는 프로그램 변환이다. 클래스를 분할할 때 클래스의 멤버간의 의존성 관계를 고려해서 결정되어야 하는 중요한 점이 있다.

셋째, Type hiding 난독화는 디자인 의도를 모호하게 만들기 위해 자바 인터페이스 개념을 사용한다. 이 난독화 기술의 특징은 주어진 클래스를 만드는 각각의 인터페이스가 클래스에 모든 public 메소드의 작은 무작위 subset만을 포함하게 한다. 프로그램의 특수한 오브젝트에 요구되는 기능을 이해하려고 시도하는 역공학자에게 이 난독화가 코드의 다른 위치에 같은 오브젝트를 나타내는 많은 다른 타입들로 어려움을 만든다.

이러한 기술 이외에 소스코드가 포함된 클래스 파일 또는 클래스 파일의 묶음인 JAR이나 DEX 파일 전체를 암호화하는 기법도 사용되고 있다[10,12].

2.3 기존 자바 코드 난독화 도구

자바 언어로 작성된 소소코드의 컴파일 결과물인 바이트코드는 비교적 분석가가 분석하기 쉬운 언어이다. 이 때문에 소스코드나 저작권을 보호하기 위하여 많은 바이트코드를 지원하는 난독화 도구들이 개발되었으며, 이러한 도구들은 널리 이용되고 있다.

현재 널리 사용되는 난독화 도구들은 대부분 식별자 변환 기법을 지원하며, 식별자변환 기능만을 제공하는 도구도 많다. 한편 자바 코드의 특성을 이용한 API 은닉, 클래스 난독화 기능을 제공하는 대중화된 안드로이드 전용 도구는 있으나, 자바 관련 대중화된 도구는 없다. 본 절에서는 6가지 자바 소스코드 난독화 도구를 소개한다. 각 도구의 설명과 기능, 기타 특징은 제작사

표 1. 자바 난독화 도구 기능 비교
Table 1. Java obfuscation tools feature comparison

	DashOPro	ProGuard jargyGuard JODE RetroGuard
Renaming	O	O
Control Flow	O	X
String Encryption	O	X
API Hiding	X	X
Class Encryption	X	X

웹사이트와 관련 논문을 참고하였다. 각 도구들이 지원하는 주요 난독화 기법은 표 1과 같다.

DashOPro[13]을 제외한 난독화 도구는 식별자 변환 기능만 제공한다. DashOPro는 제어흐름 변환, 문자열 암호화, 식별자 변환 기법 등의 다양한 기능을 지원한다. DashOPro의 특징으로 식별자 변환 기법을 적용할 때에 여러 가지 옵션을 선택할 수 있다. 식별자 변환 옵션으로 리네이밍 하거나 그대로 놔두고 싶은 식별자를 선택하는 Classes와 식별자를 변환할 때 새로운 이름 앞에 단어를 추가하는 Class Prefix, 그리고 모든 메소드와 필드의 식별자 변환을 고르고 public member에 이름을 유지하는 Members 옵션 등이 있다. 그밖에 안드로이드를 지원하여 관련 어플리케이션에 널리 이용되고 있는 난독화 도구인 프로가드[14]와 JODE[15], RetroGuard[16], jarg[17], yGuard[18]는 메소드, 클래스 식별자 변환 기법을 지원하고, 도구에 따라 필드 등 추가적인 식별자 변환 기법을 지원하기도 한다.

2.4 기존 자바 스크립트 기반 변수명 리네이밍 방법

본 연구는 자바 스크립트 언어를 기반으로 소스코드에서 변수에 관련된 행위를 머신러닝을 활용하여 분석하고, 이 과정에서 획득한 정보를 바탕으로 변수명을 부여하는 기존 논문(JSNICE)[19]을 참고하였다.

JSNICE는 머신러닝으로 입력 소스코드의 표현식을 프로그램 속성 추론을 위하여, 구조화된 예측으로 나타나는 표현식으로 바꾸는 것이다. 이 공식화는 프로그램 속성의 예측을 조건 임의 필드(CRFs) 확률 그래픽 모델로 수행한다. 이러한 접근 방식을 활용한 JSNICE는 자바 스크립트 소스코드에서 식별자들의 이름과 변수의 타입을 예측한다. 실험 결과, 정확한 이름으로 변환하는 확률이 63%이고, 타입 예측은

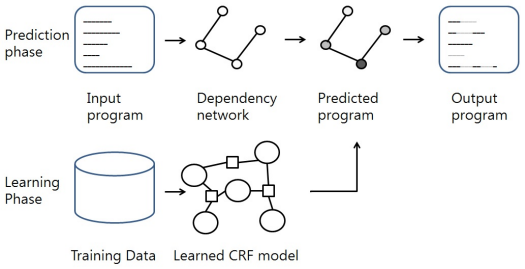


그림 1. 프로그램 속성의 통계적 예측
Fig. 1. Statistical prediction of program properties

81%의 정확성을 보여주었다.

JSNICE는 예측과 학습의 두 단계로 구조화된 예측 접근을 수행한다. 예측 단계에서는 처음으로 속성을 추론하고자 하는 프로그램을 입력한다. 다음으로 그 프로그램을 의존성 네트워크라고 불리는 표현으로 변환한다. 의존성 네트워크는 알고 있는 속성들의 요소로 예측될 수 있는 속성을 가진 프로그램 요소들 사이의 연관성을 이끌어내는 것이다. 일단 네트워크가 얻어지면 구조화된 예측과 최대 사후분포(MAP)로 언급되는 쿼리를 수행한다. 이 쿼리는 학습된 CRF 모델을 기반으로 하는 점수화 함수를 최적화함으로서 모든 요소들에 대한 예측을 한다. 종종 다른 요소들의 속성들이 연관되기 때문에 구조와 의존성을 고려한 예측을 하는 것이 특히 중요하다.

마지막으로 새롭게 예측된 속성들을 포함하는 프로그램을 출력한다. 학습 단계에서는 프로그램들의 거대한 트레이닝 집합에서 CRF 모델을 학습함으로서 좋은 점수화 함수를 만든다. 프로그램 x와 알고 있는 속성 y, 알 수 없는 속성 z에 대해서 특성 함수 f_i는 네트워크 변의 집합에 대해서 일치하는 pairwise 특성 함수 ψ_i의 합인 수식 (1)로 정의한다.

$$f_i(y,x) = \sum_{(a,b,rel)\in E^x} \psi_i((y,z)_a,(y,z)_b,rel) \quad (1)$$

프로그램 x와 알고 있는 속성 y, 알 수 없는 속성 z에 대해서 특성 함수 f_i는 네트워크 변의 집합에 대해서 일치하는 pairwise 특성 함수 ψ_i의 합으로 정의한다. 각각의 $\psi_i: Labels \times Labels \times Rels \rightarrow R$는 한 쌍의 프로그램 속성에 연관된 pairwise 특성 함수이다. 각각의 변 $(a,b,rel)\in E^x$는 요소 a와 b쌍과 이것들 사이의 관계 $rel\in Rels$로 나타낸다. 점수화 함수는 가중치 w_i와 연관된 k개의 특성 함수 f_i의 합

으로 수식 (2)와 같다.

$$score(y,x) = \sum_{i=1}^{k} w_i f_i(y,x) = \sum_{(a,b,rel)\in E^x} \sum_{i=1}^{k} w_i \psi_i((y,z)_a,(y,z)_b,rel) \quad (2)$$

이는 y의 예측에 대한 점수 함수이다. 프로그램 x와 의존성 네트워크 G^x에 대해서 pairwise function의 ψ_i와 각각의 ψ_i와 연관된 학습된 가중치 w_i를 사용하여 y의 예측에 대한 점수를 얻을 수 있다.

여기서 CRFs와 MAP 추론 쿼리를 다루기 때문에 최신의 최대 마진 CRF 트레이닝 방법을 활용한다. 최대 마진 CRF 트레이닝은 참고 논문 Max-Margin Markov Networks[20]의 기법을 참고하였다. 제안 기법은 분류에 훌륭한 강점을 가진 kernel-based classifier와 복잡한 구조를 처리할 수 있는 probabilistic classifier의 통합으로 두 개의 강점 모두 갖는 기법이다.

실행 측면에서 설명하면 처음에 속성을 추론하기 위해 프로그램 요소의 집합을 결정한다. 프로그램의 지역 변수와 같이 현재 알 수 없는 속성을 가진 요소들과 프로그램의 필드 이름, 메소드와 같은 알고 있는 속성을 가진 요소들의 집합을 만든다. 둘째로는 프로그램 요소들 사이의 다양한 종류의 연관성을 보여주는 의존성 네트워크를 만든다. 의존성 네트워크는 예측이 수행될 때 구조를 포착하는 것이 핵심이고 예측된 속성들이 서로 어떻게 영향을 주는지를 직관적으로 나타낸다. 특히 두 개의 알지 못하는 속성들 사이의 링크가 두 속성에 대한 예측이 어떤 방식으로 연관되었는지 나타낸다. 예측 알고리즘을 디자인할 때 핵심을 예측 속도의 최적화로 선택한다. 빠른 예측 속도를 통해 프로그램 개발에서 대화식 서비스를 기대한다. 또한 예측 알고리즘이 트레이닝 루프의 대부분을 차지하기 때문에 예측 알고리즘 성능이 직접적으로 성능에 영향을 준다. JSNICE에서 사용한 예측 알고리즘은 탐욕 알고리즘(Greedy Algorithm)으로 최종적 최적해를 발견하기 위해 각 단계에 지역적 최적 선택을 만드는 계층적 문제 풀이 방법이다. 알고리즘의 계산적 복잡성을 향상시키기 위해 예측 작업의 본질에 맞추었다. 특히, 특성 함수 형태를 활용한다. 셋째는 MAP 추론으로 언급되는 쿼리인 네트워크 노드에 대해 가장 가능성 높은 값을 추론한다. 네트워크의 연결된 노드들 사이에 테이블을 추가한다. 각 테이블은 선으로 연결된 노드에 대한 속성 할당을 점수화한 함수를 사용한다. 함수는 두 개의 속성을 입력 값으로 하여, 그 쌍에 대한 점수들을 출력한다. 이 때, MAP 추론은 노드 사이의 구조와 의존성을 고려하기 때문에 단순히 각 함수의 최대 점수를 가지 추론을 선택할 수 없다. 마지막으로 원본 프로그램을 추론된 이름을 사용하게 변환한다. 이 출력 프로그램에서 추론된 식별자 이름이 그 프로그램이 무엇을 하는지를 정확히 보여준다[19].

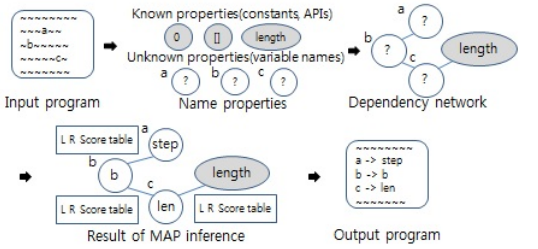


그림 2. 이름 추론 과정의 개요
Fig. 2. Overview of name inference procedure

Ⅲ. 본 론

3.1 제안 기법

본 논문의 제안 기법은 난독화된 자바 코드의 변수 이름을 직관적으로 알아볼 수 있도록 만드는 것이다. 즉, 어떤 자바 언어로 작성한 소스코드에서 정확한 의미를 지니지 않은 변수명을 변수의 용도에 알맞은 이름으로 바꾸기 위하여, 시그니처와 대조하여 변수의 용도를 추측하고 직관적으로 알아볼 수 있는 변수명으로 변경하는 것이다. 변수명을 변환하는 방법은 2가지를 제안한다. 2가지 방법은 API 정보가 저장된 시그니처를 기반으로 변수명을 변경하는 방법과 반복문을 구성하는 조건 부분과 증감 부분을 기반으로 변환하는 방법이다.

자바 언어의 컴파일 결과물인 바이트코드는 관련 연구에 소개한 바와 같이 손쉽게 자바 소스코드 형태로 변환할 수 있다. 이와 같이 변환한 자바 소스코드를 활용하여 분석 및 변수명 부여를 한다. 본 장에서는 가절에서 분석을 위한 시그니처 생성 방법을 다루고, 나절에서 소스코드 분석 방법을 다룬다. 그리고 다절에서 패턴 매칭에 따른 변수명 변환 방법을 소개하며, 라절에서 제안 기법의 성능 분석 결과를 보인다.

3.1.1 시그니처 생성 방법

시그니처는 오라클 자바 문서에 나와 있는 자바 API 목록[21]을 조사하여 작성하였다. 오라클 자바 문

서에 나와 있는 클래스를 나열하고, 연관된 클래스들을 하나로 묶어 대표 클래스로 변환시킨다. 예를 들어 float, double, integer는 num를 대표 클래스명으로 정하고 string, stringbuffer, stringbuilder는 str을 대표 클래스명으로 정한다. 그리고 각각의 대표 클래스에 포함된 클래스들의 메소드를 나열하고 중복을 제거한다. 대표 클래스 num의 compareTo와 대표 클래스 str의 compareTo와 같은 경우 앞의 토큰을 정확히 확신할 수 없기 때문에 다른 대표 클래스와 동일한 메소드를 가지는 경우도 제거 한다. 마지막으로 남은 메소드를 가지고 시그니처 형식으로 변환하여 이용한다.

시그니처 형식은 {"대표 클래스", "메소드", "조건"}으로 저장한다. 조건은 메소드를 이용한 패턴 매칭인 경우에는 A, 반복문의 조건 부분의 패턴 매칭인 경우에는 B, 반복문의 증감 부분의 패턴 매칭인 경우에는 C로 명명하였다. 즉, string 클래스의 substring의 메소드는 {"str", "substring", "A"}이다.

3.1.2 자바로 작성된 소스코드 분석

(1) 변수명을 변환하려는 소스코드를 바이트 단위로 읽어온다.

(2) 다음으로 입력받은 소스코드 원본을 토큰 단위로 분해하기 위한 작업을 수행한다. 이때 변수의 용도 추측에 불필요한 문자는 배제하여 토큰 모음에 저장한다. 읽어온 바이트를 앞에서부터 확인하면서 =, ., ,, <, >, +, -, !, ', ", [,], {, }, %, &, |, ^, ;, (,), (CR), (HT), (LF), (SP)와 같은 특정 문자가 나오는 부분을 분할하여 하나의 토큰으로 생각하고 저장한다. 위에 나온 특정 문자 중에서 (CR), (HT), (SP)와 같은 문자를 제외 나머지 문자들도 토큰으로 저장한다. 위의 방식으로 바이트 형식의 데이터를 토큰과 특정 문자로 분할하여 리스트로 저장한다.

(3) 앞서 저장한 토큰을 시그니처와 대조한다. 시그니처는 이미 잘 알려진 API를 위주로 구성되며, 시그니처와 일치하는 토큰이 나오면 해당 시그니처를 사용하는 경우에 맞게 주변 토큰이 정렬되어 있는지 확인한다.

3.1.3 패턴 매칭에 따른 변수명 변환

(1) 시그니처와 일치하는 토큰이 나오면 해당 시그니처를 바탕으로 리네이밍하는 것이 맞는지 확인하기 위해 주변 토큰을 불러온다. a.b 에서 b가 시그니처와 일치하는 것이 발견되면 앞의 토큰이 .인지 확인하고 토큰 a를 b를 사용하는 직관적인 형태의 변수명으로 변경한다. 예를 들어, variable.substring의 순서로 토큰이 저장되어 있고 현재의 토큰 확인 위치가 substring인 경우에 저장된 시그니처 substring과 일치하고 앞의 토큰이 .인지를 확인한다. 두 가지 모두 일치하는 경우에 variable를 변경한다. 이 경우에는 string변수임을 직관적으로 알 수 있게 variable를 str로 리네이밍한다.

(2) 변경된 토큰 끝에 넘버링을 붙여 준다. 동일한 클래스에 여러 개의 변수가 선언되어 이용하는 경우가 많다. 이 경우에 동일한 시그니처를 사용하여 발견되면 같은 대표 클래스명으로 변경된다. 이때 대표 클래스명 뒤에 구별 가능한 숫자를 추가한 상태로 리네이밍하여 각각의 변수를 식별 가능하게 한다.

(3) 반복문에 반복 횟수와 반복 단계를 나타내기 위해 boundary와 step 시그니처를 추가로 검색한다. 토큰을 탐색하는 중 for를 만나게 되면 반복문의 시작임을 알게 된다. for문의 조건 부분에서 부등호를 이용한 변수들의 비교를 통해 반복을 결정하는 경우가 있다. 이때 부등호 다음 토큰이 숫자가 아닌 하나의 변수명인 경우에 부등호 다음 토큰을 boundary로 변경한다. 그리고 증감 부분에서 현재 토큰과 다음 토큰이 ++, --, +=, -=으로 이루어진 경우 단조 증가, 감소가 일어남을 확인하고 이전의 토큰을 step으로 변경한다. 예를 들어, for(a=0;a<b;a++)의 소스가 있는 경우에 for토큰을 만나게 되면서 반복문의 시작을 알게 된다. 첫 번째 ;에서 다음 ;이 나올 때 까지가 조건 부분이다. 이 사이에 부등호 토큰 <이 나오고 다음 토큰 b가 숫자가 아니기 때문에 b를 boundary로 변경한다. 두 번째 ; 다음 부분이 증감 부분이다. 이 부분에서 +토큰이 나오고 다음 토큰이 +로 두 토큰을 연결하면 ++의 단조 증가이기 때문에 이전 토큰인 a를 step으로 변경한다.

3.1.4 성능 분석

본 절에서는 성능 분석을 위하여 시간 복잡도를 계산한다. 계산에서 앞서 소스코드에서 얻은 토큰의 개수가 n이고 시그니처 대조로 변경할 토큰의 개수를 k이라 가정한다. 우선, 입력한 소스코드에서 얻은 모든 토큰과 시그니처 대조를 n번 수행한다. 그리고 동일한 시그니처가 발견된 토큰을 변경하기 위해 모든 토큰을 검색하는데 n번 수행이 필요하고, 변경할 토큰의 개수가 k이기 때문에 $n \times k$번 수행이 필요하다. 따라서, $n \times k + n$의 수행 횟수를 나타낸다. 여기서 발견된 시그니처가 없거나 충분히 작은 경우에는 O(n)이지만, 모든 토큰에 대해 시그니처가 발견된 경우에는 k=n이 되기 때문에 O(n^2)이다. 하지만 일반적으로 자

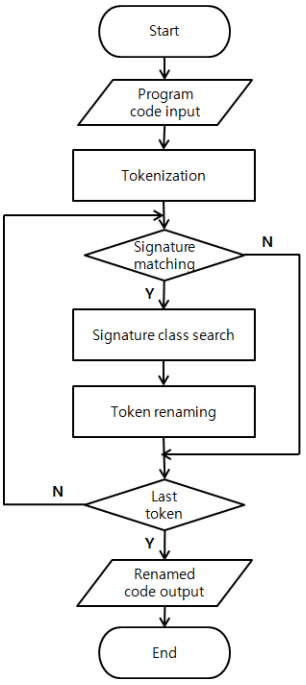


그림 3. 자바 리네이밍 알고리즘
Fig. 3. Java renaming algorithm

바소스 코드 전체가 API 정보나 반복문 내의 조건 부분일 수가 없기 때문에 실질적으로 시간 복잡도는 O(n)이다.

3.2 프로토타입 구현

프로토타입은 자바 언어를 활용하여 구현하였으며, 구현한 소스는 eclipse(버전 eclipse-java-luna-SR1a-win32-x86_64) 프로그램을 활용하여 컴파일하고 실행하였다. 실행된 프로토타입이 변환할 자바 언어로 작성한 소스 코드 파일명을 입력 받으면 바이트 단위로 파일을 읽는다. 읽어온 바이트는 토큰화하기 위해서 =, ., ,, <, >, +, -, !, ', ", [,], {, }, %, &, |, ^, ;, (,), (CR), (HT), (LF), (SP)와 같은 특정 문자가 있기 전까지를 하나의 토큰으로 저장한다. 그리고 특정 문자 중에서 (CR), (HT), (SP)을 제외한 나머지 문자들도 하나의 토큰으로 저장한다. 소스코드 전체를 토큰화한 후에 토큰의 처음부터 알려진 패턴으로 만들어진 시그니처와 대조하여 변환한다. 패턴은 오라클 자바 문서에 나와 있는 자바 API 목록[21]을 바탕으로 제작하였다. 시그니처와 일치하는 토큰이 나온 경우 해당 토큰의 앞과 뒤를 대조하고 모든 경우가 일치하는 경우에 변환을 수행한다. 변환되는 변수명은 num, str, file 등과 같이 직관적으로 알아 볼 수 있는 이름

```
import java.lang.*;

public class ThreadDemo {

   public static void main(String[] args) {

      Thread t = Thread.currentThread();
      t.setName("Admin Thread");
      // set thread priority to 1
      t.setPriority(1);

      // prints the current thread
      System.out.println("Thread = " + t);

      int c = Thread.activeCount();
      System.out.println("currently active threads = " + c);

      Thread th[] = new Thread[c];
      // returns the number of threads put into the array
      Thread.enumerate(th);

      // prints active threads
      for (int i = 0; i < c; i++) {
         System.out.println(i + ": " + th[i]);
      }
   }
}
```

```
import java.lang.* ;

public class ThreadDemo {

        public static void main ( String [ ] args ) {

                Thread thread= Thread.currentThread ( ) ;
                thread.setName ( " Admin Thread " ) ;
                // set thread priority to 1
                thread.setPriority ( 1 ) ;

                // prints the current thread
                System.out.println ( " Thread= "+ thread ) ;

                int boundary= Thread.activeCount ( ) ;
                System.out.println ( " currently active threads= "+ boundary ) ;

                Thread th [ ]= new Thread [ boundary ] ;
                // returns the number of threads put into the array
                Thread.enumerate ( th ) ;

                // prints active threads
                for ( int step= 0 ; step< boundary ; step++ ) {
                        System.out.println ( step+ " : "+ th [ step ] ) ;
                }
        }
}
```

그림 4. 식별자 리네이밍 전과 후
Fig. 4. Identifier renaming before and after

으로 변환된다. 그리고 변환 대상 변수명 끝에 숫자를 붙여서 동일한 클래스의 변수를 구분할 수 있게 한다. 또한, for문의 조건 부분을 참고하여 변수명 변환을 실시한다. for문의 조건문 내에서 부등호가 나오고 부등호 다음 토큰이 숫자가 아닌 하나의 변수인 경우 반복의 끝을 나타내는 변수명 boundary로 변경한다. for문의 증감 부분에서 ++, --, +=, -=이 나오는 경우에는 이전 토큰을 반복문의 단계를 나타내는 변수명 step으로 변경한다. 토큰의 끝까지 대조하여 변환을 완료하면, 결과를 출력한다. {이 나온 경우에 (LF)출력 후 새로운 라인의 (HT)횟수를 증가시키고 }이 나온 경우에 (LF)출력 후 새로운 라인의 (HT)횟수를 감소시키면서 토큰을 순서대로 출력하게 된다.

Ⅳ. 실험 결과

실험을 위하여 오라클사 자바 공식 웹사이트의 자바 API 목록[21]과 for문의 조건 부분과 증감 부분으로 구성한 시그니처를 만들었다. 그리고 널리 알려진 자바 예제 사이트인 tutorialspoint[22]의 예제 45개를 선정하여 실험을 진행하였다. 실험 결과는 API 용도에 따라 구분하여 발견된 패턴과 발견된 개수를 아래 표

표 2. 시그니처 대조를 통한 식별자 리네이밍 결과
Table 2. Identifier renaming result by signature comparison

	detected pattern	detected number
iterator	i.hasNext → itemIter1.hasNext	2
number	f.floatToIntBits → num.floatToIntBits a.intValue → num.intValue a.isInfinite → num.isInfinite f.parseFloat → num.parseFloat d.toHexString → num.toHexString	7
char	c.charValue → ch1.charValue	1
string	a.compareToIgnoreCase → str1.compareToIgnoreCase s.substring → str3.substring a.replace → str1.replace b.replaceFirst → str2.replaceFirst a.split → str5.split	13
input stream	is.available → localinputstream1.available in.readUTF → localinputstream1.readUTF is.mark → localInputStream.mark	4
output stream	os.write → localoutputstream1.write ou.writeUTF → localoutputstream1.writeUTF	2
time	a.parse → sysTime.parse a.setTime → sysTime.setTime b.roll → sysTime1.roll a.before → sysTime.befor a.getTime → sysTime.getTime	6
timer	a.schedule → timer.schedule a.scheduleAtFixedRate→ timer.schdeduleAtFixdRate	6
list	c.toArray → itemList.toArray	1
stack	st.push → stack.push	1
file	f.createNewFile → file.createNewFile f.getPath → file.getPath f.isHidden → file.isHidden f.setWritable → file.setWritable	4
inetaddress	a.getHostAddress → inetaddr.getHostAddress a.getHostName → inetaddr.getHostName	2
socket	s.getInetAddress → sock.getInetAddress s.getRemoteSocketAddress → sock.getRemoteSocketAddress	2
url	u.getProtocol → urlconn.getProtocol u2.getHost → urlconn2.getHost u5.getDefaultPort → urlconn3.getDefaultPort u.openConnection → urlConn.openConnection u.openStream → urlConn.openStream	6
process	p.getErrorStream → process.getErrorStream p.waitfor → process.waitfor	2
thread	t.checkAccess → thread.checkAccess	6
	t.isAlive → thread.isAlive	
	t.setName →thread.setName	
	t.setPriority → thread.setPriority	
	t.getContextClassLoader → thread.getContextClassLoader	

에 기술하였다.

실험 결과를 분석한 결과 토큰이 시그니처와 동일한 경우에는 정확한 변환이 이루어졌다. 그러나 메소드를 사용하지 않는 변수가 있는 예제 소스코드를 대상으로 실험한 경우에 해당 변수명을 리네이밍하지 못했다. 그 결과 전체 45개의 예제에 포함된 83개 변수 중에서 61개 변수에 식별자 변환이 이루어졌다. 이는 본 실험에서의 성공률이 73%임을 의미한다. 제안 기법은 기존 연구와 달리, 이미 저장된 시그니처를 바

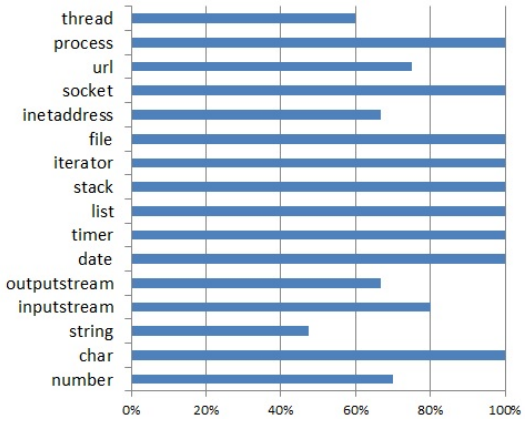


그림 5. 각각의 대표 클래스의 식별자 리네이밍 정확성
Fig. 5. Identifier renaming precision of each representation class

탕으로 실험을 진행하기 때문에 리네이밍 대상 변수명은 분명한 정답을 가지고 있다. 따라서 본 실험에서 리네이밍된 변수명은 모두 정확한 결과를 보였다.

V. 결론 및 고찰

본 연구는 자바 언어로 작성한 난독화된 코드나 디컴파일하여 얻은 코드의 변수 이름을 직관적으로 알아보기 위한 방법에 관한 것이다. 구체적으로 소스코드를 토큰 단위로 분할하고, 자바 언어의 특성을 고려하여 API와 반복문의 조건부분과 증감부분을 시그니처로 저장하여 소스코드의 변수명을 변수의 행위와 연관된 이름으로 바꿔주는 내용이다. 기존의 머신 러닝기반 연구와 달리 오라클 자바 API 목록[21]을 시그니처로 구성하였기 때문에 더 나은 정확도를 보이는 장점이 있다.

제안 기법의 실용성을 보이기 위하여 프로토타입을 구현하고, 공신력 있는 예제를 활용하여 실험을 진행하였다. 그 결과 73%의 높은 리네이밍 성공률을 보였다. 따라서 본 연구 결과를 바탕으로 리네이밍 서비스를 활용하면, 공동 작업 과정에서 공동 작업자가 각 변수의 역할을 변수명을 바탕으로 직관적으로 이해할 수 있으며, 악성코드 분석 등을 위하여 자바 어플리케이션의 바이트코드를 디컴파일한 소스코드에 적용하여 분석 시간을 줄이는 효과가 있다.

하지만 제안 리네이밍 서비스는 저작물의 불법 복제에 활용되는 등 저작권 침해의 소지를 가지고 있다. 따라서 본 기법을 우회할 수 있는 방법을 간략히 소개한다.

첫째, 클래스 난독화 기법을 적용하여 난독화된 자바 코드는 디컴파일러만을 이용하여 소스코드로 변환할 수 없다. 난독화된 코드는 바이트코드와 전혀 다른 형태를 가지기 때문이다. 따라서 분석자가 클래스 난독화 기법이 적용된 코드를 디컴파일하려면, 복호화 루틴을 제작하여 활용하는 절차가 필요하다[10].

둘째, API 정보를 숨기면 본 기술을 우회할 수 있다. 관련 기술로 앞서 소개한 API 은닉 기법과 API Wrapping 기법이 있다. API Wrapping은 API를 직접 사용하지 않고, 특정 API와 같은 기능을 하는 함수를 별도로 코드에 추가하는 기법이다. 이러한 기법으로 제안 기법의 적용이 회피되는 것은 제안 기법이 알려진 API 명을 시그니처로 저장하고, 시그니처의 범위 내에서 판단하여 변수명을 리네이밍하기 때문이다. API Wrapping 기법은 실행 파일 패커인 더미다[23]에서 사용하는 기법이다. 더미다는 이미 제작된 실행 파일의 바이너리를 바꿔주지만, 저작권자가 직접 이 기법을 소스코드에 적용하면 자바 언어를 사용하는 프로그램 개발 과정에서도 활용할 수 있다.

References

[1] M. K. Son and N. H. Kang, “Design and implementation of java crypto provider for android platform,” *J. KICS*, vol. 37C, no. 09, pp. 851-858, Sept. 2012.

[2] B. H. Choi, H. J. Shim, C. H. Lee, S. W. Cho, and S. J. Cho, “An APK overwrite scheme for preventing modification of android applications,” *J. KICS*, vol. 39B, no. 05, pp. 309-136, Jun. 2014.

[3] IDC, *Android and iOS squeeze the competition, swelling to 96.3% of the smartphone operating system market for both 4Q14 and CY14*, According to IDC(2015), Retrieved Mar., 12, 2015, from http://www.idc.com/getdoc.jsp?containerId=prUS25450615

[4] Y. K. Kim and H. Y. Youn, “The java decompilation-preventive method by java class file encryption,” *Korea Computer Congress 2009*, vol. 36, no. 1C, pp. 571-574, Jeju Island, Korea, Jun. 2009.

[5] T. Varaneckas, README(readme.txt), Retrieved Mar, 12, 2015, from http://varaneckas.com/jad/

[6] B. Y. Lee and Y. S. Choi, “The status and

analysis of obfuscation techniques and perspective development," *J. Security Eng.*, vol. 5, no. 3, pp. 219-228, Jun. 2009.

[7] J. U. Noh, B. M. Cho, H. S. Oh, H. Y. Chang, M. Y. Jung, S. W. Lee, Y. S. Park, J. H. Woo, and S. J. Cho, "An implementation of control flow obfuscator for C++ language," *Korea Computer Congress*, vol. 33, no. 1, pp. 295-297, Yongpyong, Korea, Jun. 2006.

[8] C. Christian, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," Dept. Computer Sci., The University of Auckland, New Zealand, 1997.

[9] Y. Piao, "Server-based bytecode obfuscation scheme for tamper detection of android applications," M. S. Thesis, Dept. Computer, Soongsil Univ., Korea, 2013.

[10] J. Y. Kim, N. H. Go, and Y. S. Park, "A code concealment method using java reflection and dynamic loading in android," *J. The Korea Inst. Inf. Security & Cryptol.*, vol. 25, no. 1, pp. 17-30, Feb. 2015.

[11] ORACLE, The Java™ Tutorials - Trail: The Reflection API(2015), Retrieved Mar. 6, 2015, from http://docs.oracle.com/javase/tutorial/reflect/

[12] M. Sosonkin, G. Naumovich, and N. Memon, "Obfuscation of design intent in object-oriented applications," in *Proc. 3rd ACM Workshop on Digital Rights Management (DRM '03)*, pp. 142-153, Washington, DC, USA, Oct. 2003.

[13] PREEMPTIVE SOLUTIONS, *User's Guide* (2009), Retrieved Mar. 6, 2015., from http://www.agtech.co.jp/products/preemptive/dasho/files/userguide6.pdf

[14] Y. Piao, J. H. Jung, and J. H. Yi, "Structural and functional analyses of ProGuard obfuscation tool," *J. KICS*, vol. 38B, no. 08, pp. 654-662, Aug. 2013.

[15] J. Hoenicke, *JODE*(2002), Retrieved Mar. 12, 2015., from http://jode.sourceforge.net/

[16] *Retrologic Systems, User's Manual*(2010), Retrieved Mar. 12, 2015., from http://www.retrologic.com/retroguard-docs.html

[17] jarg, *jarg - Java Archive Grinder*(2003), Retrieved Mar. 12, 2015., from http://jarg.sourceforge.net

[18] yWorks, yGuard - *Java™ Bytecode Obfuscator and Shrinker*(2015), Retrieved Mar. 12, 2015., from http://www.yworks.com/en/products_yguard_about.html

[19] V. Raychev, M. Vechev, and A. Krause, "Predicting program properties from "Big Code"," in *Proc. 42nd Annu. ACM SIGPLAN-SIGACT Symp. Principles of Programming Languages,* pp. 111-124, Mumbai, India, Jan. 2015.

[20] B. Taskar, C. Guestrin, and D. Koller, "Max-margin markov networks," *Advances in Neural Inf. Process. Syst. 16 (NIPS 2003)*, pp. 25-32, Vancouver and Whistler, British Columbia, Canada, Dec. 2003.

[21] ORACLE, Java™ Platform, Standard Edition 7 - API Specification(2014), Retrieved Mar., 12, 2015, from http://docs.oracle.com/javase/7/docs/api/

[22] tutorialspoint, *Java Tutorial*(2014) Retrieved Mar., 9, 2015, from http://www.tutorialspoint.com/java/

[23] OREANS TECHNOLOGIES, *THEMIDA | OVERVIEW*(2015), Retrieved Mar. 12, 2015, from http://www.oreans.com/themida.php

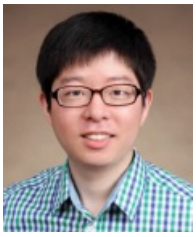
김 지 윤 (Ji-yun Kim)

2013년 2월 : 한양대학교 자원환경공학과 졸업

2014년 3월~현재 : 한양대학교 컴퓨터소프트웨어학과 석사과정

<관심분야> 네트워크, 컴퓨터 시스템, 모바일 시스템, 시스템 보안, 금융 보안, 보안 정책

홍 수 화 (Soo-hwa Hong)



2014년 2월 : 한양대학교 컴퓨터공학부 졸업
2015년 3월~현재 : 한양대학교 컴퓨터소프트웨어학과 석사과정
<관심분야> 컴퓨터 보안, 시스템 보안

이 우 승 (Woo-Seung Lee)

2015년 2월 : 한국외국어대학교 컴퓨터공학과 졸업
2015년 3월~현재 : 한양대학교 컴퓨터소프트웨어학과 석사과정
<관심분야> 웹 보안, 모의 해킹

고 남 현 (Nam-hyeon Go)

2015년 2월 : 한국폴리텍2대학 컴퓨터정보과 졸업
2015년 3월~현재 : 한국방송통신대학교 컴퓨터과학과 학사과정
<관심분야> 인터넷 보안, 컴퓨터 보안, 임베디드 보안, 금융 보안, 보안 정책

박 용 수 (Yong-su Park)

1996년 : KAIST 전산학과 졸업
1998년 : 서울대학교 대학원 컴퓨터공학과 졸업(석사)
2003년 : 서울대학교 대학원 컴퓨터공학과 졸업(박사)
2003년~2004년 : 서울대학교 자동제어특화연구센터 연수연구원
2005년~현재 : 한양대학교 소프트웨어전공 부교수
<관심분야> 컴퓨터 보안, 인터넷 보안