논문 2016-1-4

# 문자열 빈도 정보를 이용한 MS 윈도우 실행파일의 효과적인 분류기법

조대희*, 조성제*†

# An Effective Classification Scheme for Microsoft Windows Executables using Frequency Information of Strings

DaeHee Cho*, Seong-je Cho*†
Dept. of Computer Science and Engineering, Dankook University

요 약

불법 소프트웨어를 효율적으로 필터링하기 위해, 소프트웨어 분류 기법이 도입되었다. 본 논문에서는 문자열 빈도 정보에 기반하여 MS 윈도우 실행파일들을 효과적으로 분류하는 기법을 제안한다. 제안 기법은 윈도우 실행파일의 .text, .data, .rdata, .rsrc 등의 섹션에서 문자열들을 추출한 후, LDF(Local Document Frequency)-IDF(Inverse Document Frequency)-ICF(Inverse Category Frequency) 값들을 결합하여 카테고리 별 문자열 점수표를 계산하여 유지한다. 의심 프로그램(실행파일)이 유용한 문자열 정보를 포함하고 있을 경우, 제안 기법은 카테고리 별 문자열 점수표를 사용하여 의심 프로그램의 문자열(단어)들에 대한 각 점수를 구하고, 그 점수들을 합산하여 총 점수를 구한다. 마지막으로, 그 의심 프로그램을 최고 높은 총 점수를 나타내는 카테고리로 분류한다. 제안 기법의 효용성을 검증하기 위해, 9개의 카테고리, 각 카테고리마다 55개의 실행파일을 대상으로 실험하였다. 495개의 실행파일을 대상으로 한 실험결과, 제안 기법이 약 75%의 분류 정확도를 보임을 확인하였다.

Abstract

In order to filter and block illegal software efficiently, software classification is employed. In this paper, we propose a new software classification scheme that effectively categorizes Microsoft Windows executable files using frequency information of strings. The proposed scheme first extracts strings from the .text, .data, .rdata, and .rsrc sections of each Windows executable, and then computes and maintains a score table of the strings by combining Local Document Frequency (LDF), Inverse Document Frequency (IDF), and Inverse Category Frequency (ICF). If a suspicious program (Windows executable) got useful string information, we calculate scores of the strings in the suspicious program based on the score table of strings of each category, sums up the scores, and obtains a total score per each category. Finally, we classify the suspicious program into a specific category which represents the highest total score. For verifying the effectiveness of the proposed scheme, we perform experiments with nine categories and 55 programs each category. The experimental results show that about 75% of the 495 executables are correctly classified.

**한글키워드** : 소프트웨어 분류, 윈도우 실행파일, 문자열 빈도, 소프트웨어 필터링
**keywords** : Software classification, Windows executable, String frequency, Software filtering

* 단국대학교 컴퓨터학과
† 교신저자: 조성제(email: sjcho@dankook.ac.kr)
접수일자: 2016.5.18. 심사완료: 2016.6.6.
게재확정: 2016.6.19.

※ 이 논문은 2015년 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(no. NRF-2015R1D1A1A02061946), 그리고 미래창조과학부 및 한국인터넷진흥원의 "2014년도 고용계약형 정보보호 석사과정 사업"의 연구 결과로 수행되었음 (과제번호 H2101-14-1001)"

## 1. 서 론

2013년 IDC(international Data Corporation)가 발간한 백서, “The Dangerous World of Counterfeit and Pirated Software”에 따르면, 2011년 PC 소프트웨어 패키지의 42%가 불법복제 되었으며, 불법복제 소프트웨어의 최소 80%가 위조되었다고 추정된다[1].

불법 복제된 프로그램, 위·변조된 프로그램, 해킹된 모바일 앱 등의 불법 소프트웨어로 인하여 소프트웨어 개발사·저작권사 매출 하락, 소프트웨어 기술개발·인력양성·투자 감소와 같은 악영향이 나타나고 있다. 불법 소프트웨어는 웹하드(또는 Online Service Provider 사이트), 토렌트, 블랙마켓, 블로그, 카페 등 다양한 경로를 통해 배포된다. 불법 소프트웨어를 탐지하여 차단하기 위해서 소프트웨어 필터링 기법[2,18]이 도입되었다.

소프트웨어 필터링 기법은 의심스러운 프로그램이 업로드/다운로드 될 때, 의심 프로그램이 기존 데이터베이스(DB)에 등록된 많은 프로그램들 중에 하나인지 아닌 지를 식별하여 판단하게 된다. 이 경우, 데이터베이스에 저장된 프로그램 정보와 1대1로 단순 비교한다면 필터링 처리시간이 증가하게 된다. 이에, 소프트웨어 필터링 시에 프로그램들 간의 비교 오버헤드를 줄이기 위한 소프트웨어 분류 기법[2,3,20,21]들이 연구되고 있다.

소프트웨어 분류에서는, 서로 특징이 유사한 기존 프로그램들을 특정 카테고리 별로 나누어 DB에 관리되면서, 의심 프로그램을 DB에 등록된 프로그램들과 1:1 단순 비교하기 전에 먼저 어떤 카테고리에 속한 프로그램인지를 분류하는 것이다. 분류가 성공할 경우에는, 의심 프로그램을 해당 카테고리에 속한 프로그램들과 1:1로 비교한다. 따라서 소프트웨어 분류는, 소프트웨어 필터링 시의 비교 범위를 감소시켜 줄 수 있다.

본 논문에서는 마이크로소프트 윈도우 실행파일들을 대상으로 문자열(string, 또는 단어) 기반한 효과적인 프로그램 분류 기법을 제안한다. 제안 기법은 DB에 등록된 프로그램들을 9개의 카테고리로 나누어 관리한다. 또한 각 카테고리 별로, 실행 프로그램들의 문자열 정보를 추출한 후 문자열 빈도(frequency) 정보를 사용하여 카테고리 별 문자열 점수도 함께 관리한다. 의심 프로그램이 있을 때, 의심 프로그램의 문자열 빈도 점수를 계산하여 카테고리 별 문자열 빈도 점수와 비교하여, 의심 프로그램을 효과적으로 분류한다.

본 논문의 구성은 다음과 같다. 2장에서는 소프트웨어 버스마크, 소프트웨어 분류, 문자열 빈도 기반의 점수계산 시스템에 대하여 설명한다. 3장에서는 이 논문에서 제안하는 기법에 대해 설명한다. 4장에서는 제안하는 기법을 실험을 통해 성능을 확인한다. 마지막으로 5장에서는 결론을 맺는다.

## 2. 관련 연구

### 2.1 소프트웨어 버스마크

소프트웨어 버스마크(software birthmark)란 대상 프로그램에 존재하는 고유한 특성으로 그 프로그램을 식별하고 도용 여부를 탐지하는데 사용될 수 있다[18, 19, 21]. 실행 프로그램 관련 소프트웨어 버스마크는 실행 프로그램을 실행시키지 않고 특징정보를 추출하는 정적 버스마크(static birthmark)와 프로그램을 실행시키면서 특징정보를 추출하는 동적 버스마크(dynamic birthmark)로 분류된다. 소프트웨어에 대한 정적 버스마크와 동적 버스마크에 대한 간단한 비교가 표 1에 나타나 있다. 본 논문에서는 소프트웨어 필터링에 사용할 수 있는 정적 버스마크(또는 정적인 특징정보)를 고려한다.

표 1. 정적 버스마크와 동적 버스마크 비교
Table 1. Static and dynamic busmark

| 종류 | 정적 버스마크 | 동적 버스마크 |
|---|---|---|
| 장점 | • 높은 코드 커버리지<br>• 분석 및 추출이 효율적임 | • 간접 분기 분석이 가능<br>• 의미 보존 공격에 강인 |
| 단점 | • 간접 분기 예측이 어려움<br>• 의미 보존 변환 공격에 취약 | • 낮은 코드 커버리지<br>• 실행환경에 영향 받음 |

### 2.2 TF-IDF

TF-IDF (Term Frequency-Inverse Document Frequency)[7]는 문서에 포함된 문자들에 점수를 주어 문서 간 유사도를 측정하는데 사용하는 기법이다. 한 문서 내 단어 빈도(TF)와 전체 문서 내 단어 빈도의 역수(IDF)를 곱하여 문자의 점수를 측정한다. 본 논문에서는, TF-IDF를 일부 변형한 LDF(Local Document Frequency)와 ICF(Inverse Category Frequency)도 사용한다.

### 2.3 소프트웨어 분류

Kim [3, 22]은 API 호출 빈도를 사용한 기계학습 분류와 .rsrc 섹션에서 문자를 추출하여 TF-IDF를 적용한 소프트웨어 분류를 제안하였다. 이 연구에서는 .rsrc 섹션에서 리소스 문자열을 대상으로 하여, 추출할 수 있는 문자열 정보를 가진 프로그램 수가 충분하지 않았다. 또한 분석에 사용된 프로그램 실험 집합과 카테고리에는 동일 프로그램의 다른 버전들이 일부 포함되었다. 문자열 점수 계산에 TF-IDF-ICF를 사용하였다. 문자 추출에는 ResourceExtract[4]를 사용하며, 문자 추출 시 TF-IDF를 적용하기 위한 문자의 통일성[17]이 부족한 편이다. 또한, 불용어(StopWords)[5]를 체계적으로 제외하지 않았다.

Choi [6]는 안드로이드 애플리케이션을 대상으로 API, 문자열, URL 정보를 몽타주로 만들어 분류한다. 이 연구에서는 1개의 카테고리만을 사용하였으며, 5개의 프로그램으로 몽타주를 작성하였다. 몽타주 작성에 필요한 프로그램의 수가 적으며, 카테고리가 늘어날 경우 정확하게 프로그램을 분류할 수 있다는 보장하기 어렵다.

## 3. 제안 기법

본 논문에서는 기존 소프트웨어 필터링 시 1대 1 비교의 오버헤드를 줄이기 위한 소프트웨어 분류 기법을 제안한다. 제안하는 기법은 실행파일에서 문자를 추출 후 LDF-IDF-ICF 방식을 활용하여 점수 기반 분류를 한다. 제안 기법에서 소프트웨어 분류 방식의 절차는 그림 1과 같다.

### 3.1 실행파일 수집

다양한 오픈소스 소프트웨어를 획득할 수 있는 사이트[8-14]들을 활용하여 MS 윈도우 프로그램들을 수집하였다. 또한 [3]과 [22] 연구에 사용된 9개의 카테고리로 프로그램을 수집하였다. 수집한 프로그램에서 실행파일, 즉 PE(Portable Executable) 포맷으로 '.exe' 확장자를 가진 것을 고려하였다.

### 3.2 실행압축 유/무 확인

제안하는 기법은 소프트웨어 필터링 시스템에 적용할 수 있는 정적 특징정보를 고려한다. 패킹(Packing) 등의 압축 기법이 적용된 실행파일에서 정적 특징정보를 추출할 경우, 추출된 정보가 정확하지 않을 수 있어, 본 논문에서는 실행압축이 적용되지 않은 경우만을 고려한다.

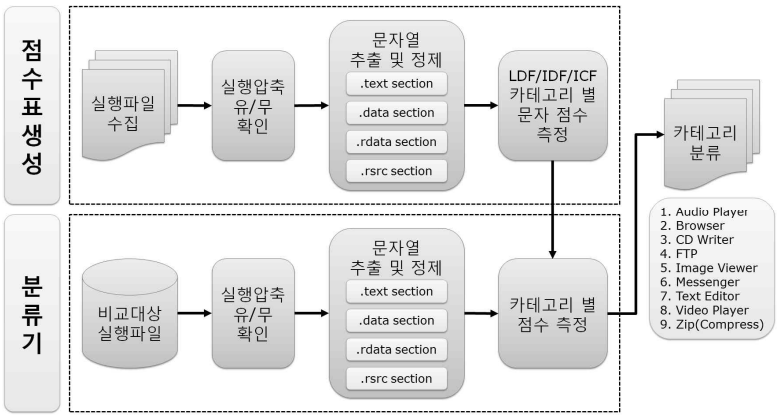


그림 1.문자열 정보를 이용한 소프트웨어 분류 기법 개요
Fig. 1. Software distribution method by string data

### 3.3 문자 추출

실행파일의 여러 섹션에서 문자열을 추출한다. 문자열 추출 방식은 크게 다음 두 가지이다.

.text/.data/.rdata 섹션에 존재하는 문자열은 IDApro[15]와 IDAPython[16]을 사용하여 추출한다.

.rsrc 섹션의 문자열 추출은 [22] 연구에서 사용된 방식과 유사하게, RT_STRING 문자열 추출 및 특정 카테고리의 특성과 관련이 많은 RT_MENU, RT_DIALOG 정보를 추출한다. .rsrc 섹션에서 문자열을 추출할 경우, 언어의 옵션에 따라 추출 유/무를 정한다. 언어의 유형 중에, LANG_NEUTRAL과 LANG_ENGLISH 두 가지만을 고려하여 추출한다[17].

### 3.4 문자열 정제

실행파일의 각 섹션(부분)에서 문자열을 추출할 경우, 문자열 정보가 하나의 단어로 되어있지 않고, 여러 단어들(문장)로 이루어져 있는 경우가 대부분이다. LDF-IDF-ICF를 적용할 경우 여러 문자열들을 각각의 단어(개별 문자열)로 나누어 주어야한다. 문자열 정제 절차가 그림 2에 나타나 있다.

문자열 정제 절차 중 2단계에서는 a, an, the, of 등과 같은 불용어(Stop words)를 제거한다. 또한, 문자열 추출 후, 의미 없는 문자열을 불용어 목록에 추가한다. 동일 문자가 세 번 연속으로 나타난 단어의 경우 불용어로 간주하나, ‘www’와 같은 의미가 있는 단어는 불용어로 간주하지 않고 포함한다.

문자열 정제 후 50개 이상의 단어들을 포함하고 있는 프로그램들만을 고려하여 진행한다. 단어들의 수를 50개 이상으로 한 이유는, 너무 적은 단어들로 분류에 필요한 점수표를 생성 시, 소프트웨어 분류에 오차를 줄 수 있기 때문이다.

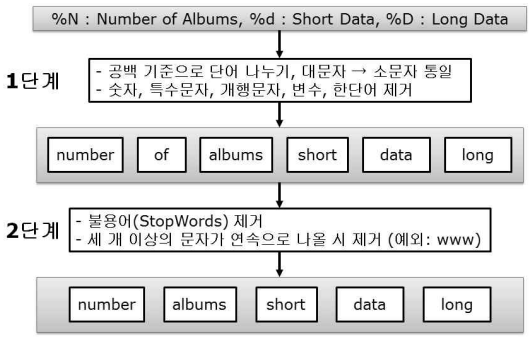


그림 2. 문자열 정제 절차
Fig. 2. String refining procedure

### 3.5 문자열 점수 계산

프로그램 분류에 사용될 각 문자열 점수는 TF-IDF를 변형하여 계산한다. 문자열 점수 계산식이 그림 3에 주어져 있다.

$$LDF(f, C_i) = log_{10}\left(\left(\frac{|P|}{|C_i|}\right) + 1\right) where\ P = \{p | f \in p, p \in C_i\}$$

$$IDF(f, D) = log_{10}\left(\frac{|D|}{|P|}\right) where\ P = \{p | f \in p, p \in D\}$$

$$ICF(f, D) = log_{10}\left(\left(\frac{n}{|C|}\right) + 1\right) where\ P = \{C_i | f \in p, p \in C_i\}$$

$$SCORE(f, C_i) = LDF(f, C_i) * IDF(f, D) * ICF(f, D)$$

$D$: 프로그램 셋 $n$: 카테고리 번호 $f$: 문자열

$C_1, C_2 \ldots C_n$ : 카테고리, $C_i \cap C_j = \emptyset$

그림 3. LDF-IDF-ICF 점수 계산식
Fig. 3. LDF-IDF-ICF score calculation

LDF(Local Document Frequency)는 하나의 특정 카테고리 내에서 해당 문자열을 포함한 프로그램의 수를 반영한다. IDF(Inverse Document Frequency)는 모든 카테고리에서 해당 문자열을 포함하고 있는 프로그램들의 개수에 대한 역이다. ICF(Inverse Category Frequency)는 해당 문자열을 포함하고 있는 카테고리들 개수에 대한 역이다. 특정 카테고리 Ci에서 한 문자열 f에 대한 점수는 그림 3에서와 같이 SCORE(f, Ci)로 계산된다. 카테고리 별로 각 문자열에 대한 SCORE(f, Ci) 표를 만들어 관리한다.

### 3.6 문자열 점수 기반 분류

문자열 정제 후에 50개 이상의 단어(문자열)를 갖는 실행파일이라면, 문자열들에 대한 점수를 카테고리 별로 계산한다. 각 문자열에 대한 점수를 계산하여 합산한 결과 가장 높은 점수 값을 갖는 카테고리로 분류한다. 만약, 가장 높은 점수 값을 갖는 카테고리가 2개 이상일 경우에는 분류되지 않은 것으로 간주한다.

## 4. 실험 및 분석

본 논문에서 제안하는 기법의 성능을 평가하기 위해, 9개의 카테고리로 나누고, 각 카테고리 별 55개 실행파일, 총 495개의 실행파일을 실행 프로그램 집합(Program Set)으로 하여 실험하였다. 총 495개 실행파일이 포함된 각 프로그램 집합을 두 가지 종류로 구성하였다.

'***첫 번째 프로그램 집합***'은 한 카테고리 내에 55개 실행파일 중 동일한 프로그램의 서로 다른 버전의 실행파일들을 포함하고 있다(평균적으로 각 카테고리마다 30여개의 서로 다른 프로그램, 특정 프로그램에 각각에 대해 2~3개의 다른 버전으로 구성됨. 따라서 프로그램 수집이 비교적 용이).

'***두 번째 프로그램 집합***'은 한 카테고리의 55개 실행파일 중 최소 50여개는 서로 다른 프로그램이고, 특정한 프로그램에 대해 2개의 다른 버전으로 구성된다(가능한 서로 다른 프로그램으로 구성하여, '첫 번째 프로그램 집합'에 비해 프로그램 수집이 좀 더 어려움).

실험에 사용된 프로그램 집합에 대한 설명은 표 5를 참고 바란다.

### 4.1 문자열 추출률 및 정제율

'첫 번째 프로그램 집합'의 경우, 495개의 실행파일에서 문자열 추출이 가능한 실행파일은 427개로 추출률은 86.3%(= 427/495*100)이다. 문자열 정제 후에 50개 이상의 단어를 포함한 실행파일 수는 413개이다. 각 카테고리에서 '***정제율***'은 문자열 추출하고 정제한 후 50개 이상의 단어를 포함한 실행파일들의 비율이라고 할 경우, 정제율은 문자열 추출 가능한 실행파일 개수 대비 96.7%(= 413/427*100)의 정제율을, 전체 실행파일 개수 495개 대비 83.4%(= 413/495*100)의 정

표 2. 문자열 추출 개수 및 정제율 (단, 개수는 프로그램 수를 의미)
Table 2. Refining rate and string extraction

| 카테고리 | 첫 번째 프로그램 집합 대상 | | | | | 두 번째 프로그램 집합 대상 | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 추출 (개수) | 추출률 (%) | 정제 (개수) | 정제율 (%) 추출 대비 | 정제율 (%) 55개 대비 | 추출 (개수) | 추출률 (%) | 정제 (개수) | 정제율 (%) 추출 대비 | 정제율 (%) 55개 대비 |
| Audio Player | 51 | 92.7 | 50 | 98 | 90.9 | 52 | 94.5 | 50 | 96.2 | 90.9 |
| Browser | 45 | 81.8 | 41 | 91.1 | 74.5 | 46 | 83.6 | 40 | 87 | 72.7 |
| CD Writer | 50 | 90.9 | 49 | 98.0 | 89.1 | 55 | 100 | 55 | 100 | 100 |
| FTP | 51 | 92.7 | 48 | 94.1 | 87.3 | 55 | 100 | 54 | 98.2 | 98.2 |
| Image Viewer | 50 | 90.9 | 48 | 96 | 87.3 | 52 | 94.5 | 52 | 100 | 94.5 |
| Messenger | 32 | 58.2 | 32 | 100 | 58.2 | 44 | 80 | 44 | 100 | 80 |
| Text Editor | 48 | 87.3 | 46 | 95.8 | 83.6 | 52 | 94.5 | 50 | 96.2 | 90.9 |
| Video Player | 48 | 87.3 | 47 | 97.9 | 85.5 | 48 | 87.3 | 47 | 97.9 | 85.5 |
| Zip(Compress) | 52 | 94.5 | 52 | 100 | 94.5 | 51 | 92.7 | 51 | 100 | 92.7 |
| 합계 또는 비율 | 427 | 86.3 | 413 | 96.7 | 83.4 | 455 | 91.9 | 443 | 97.4 | 89.5 |

제율을 보인다.

'두 번째 프로그램 집합'의 경우, 총 495개의 실행파일에서 문자열 추출이 가능한 실행파일은 455개로 91.9% 추출이 가능하다. 그리고 문자열 정제 후 443개를 점수표 생성 및 분류에 사용이 가능하며, 문자열 추출이 가능한 실행파일 대비 99.4%, 전체 실행파일 개수 대비 89.5%의 정제율을 보여준다(표 2 참고).

### 4.2 프로그램 분류율

9개의 카테고리, 각 카테고리별 55개 실행파일로 구성된 495개의 실행파일들에 대한 문자열 점수표를 생성한 후, 이 495개 실행파일을 분류한 실험 결과가 표 3과 표 4에 나타나 있다. 일반적으로는 문자열 점수표 생성에 포함되지 않은 별도의 실행파일을 대상으로 분류 실험을 해야 하

표 3. '첫 번째 프로그램 집합'에 대한 분류율
Table 3. Distribution rate of '1st program group'

| 카테고리 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 분류 (개) | 분류율 (%) 정제 대비 | 분류율 (%) 55개 대비 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 44 | 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 44 | 88 | 80 |
| 2 | 0 | 38 | 0 | 0 | 0 | 0 | 3 | 0 | 0 | 38 | 92.7 | 69.1 |
| 3 | 0 | 9 | 33 | 4 | 0 | 0 | 0 | 0 | 3 | 33 | 67.3 | 60 |
| 4 | 0 | 6 | 0 | 42 | 0 | 0 | 0 | 0 | 0 | 42 | 87.5 | 76.4 |
| 5 | 0 | 3 | 0 | 2 | 43 | 0 | 0 | 0 | 0 | 43 | 89.6 | 78.2 |
| 6 | 0 | 0 | 0 | 0 | 0 | 32 | 0 | 0 | 0 | 32 | 100 | 58.2 |
| 7 | 0 | 5 | 0 | 0 | 0 | 0 | 41 | 0 | 0 | 41 | 89.1 | 74.5 |
| 8 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 46 | 0 | 46 | 97.9 | 83.6 |
| 9 | 0 | 2 | 6 | 1 | 1 | 0 | 0 | 0 | 42 | 42 | 80.8 | 76.4 |
| 합계/비율 | 44 | 70 | 39 | 49 | 44 | 32 | 44 | 46 | 45 | 361 | 87.4 | 72.9 |

표 4. '두 번째 프로그램 집합'에 대한 분류율
Table 4. Distribution rate of '2nd program group'

| 카테고리 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 분류(개) | 분류율(%) | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | 정제 대비 | 55개 대비 |
| 1 | 46 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 46 | 92 | 83.6 |
| 2 | 0 | 28 | 0 | 0 | 0 | 6 | 5 | 0 | 1 | 28 | 70 | 50.9 |
| 3 | 0 | 5 | 48 | 0 | 0 | 2 | 0 | 0 | 0 | 48 | 87.3 | 87.3 |
| 4 | 0 | 6 | 0 | 46 | 0 | 0 | 2 | 0 | 0 | 46 | 85.2 | 83.6 |
| 5 | 0 | 3 | 0 | 0 | 47 | 2 | 0 | 0 | 0 | 47 | 90.4 | 85.5 |
| 6 | 0 | 3 | 0 | 0 | 0 | 39 | 2 | 0 | 0 | 39 | 88.6 | 70.9 |
| 7 | 0 | 5 | 0 | 0 | 0 | 0 | 42 | 1 | 2 | 42 | 84 | 76.4 |
| 8 | 1 | 2 | 0 | 0 | 0 | 1 | 0 | 43 | 0 | 43 | 91.5 | 78.2 |
| 9 | 0 | 4 | 2 | 0 | 1 | 2 | 1 | 0 | 41 | 41 | 80.4 | 74.5 |
| 합계/비율 | 47 | 60 | 50 | 46 | 48 | 52 | 52 | 44 | 44 | 380 | 85.8 | 76.8 |

지만, 별도의 추가 실행파일을 수집하는데 어려움이 있어, 문자열 점수표 생성에 포함된 실행파일들을 사용하여 분류율을 실험하였다.

'첫 번째 프로그램 집합'의 경우, 유용한 문자열 정보가 획득된 413개 실행파일 중 361개를 성공적으로 분류하였다. 이는 문자열이 정제된 실행파일들 중 87.4%가, 전체 실행파일들 중 72.9%가 정상 분류된 것이다(표 3 참고).

'두 번째 프로그램 집합'의 경우, 유용한 문자열 정보가 획득된 443개 실행파일 중 380개를 분류하였다. 즉, 문자열이 정제된 실행파일들 중 85.8%가, 전체 실행파일들 중 76.8%가 정상 분류되었다(표 4 참고).

### 4.3 실험 프로그램 집합에 따른 분류율 분석

표 3과 표 4를 비교하면 '정제 대비' 분류율은 '첫 번째 프로그램 집합'이 높지만, '전체 실행파일 개수'(495개) 대비 분류율은 '두 번째 프로그램 집합'이 높다. 이러한 이유는, 표 5의 '첫 번째 프로그램 집합'과 '두 번째 프로그램 집합'의 서로 다른 프로그램의 개수 차이에서 확인할 수 있다.

'첫 번째 프로그램 집합'의 경우, 서로 다른 프로그램의 개수가 총 273개이고, 유용한 문자열 정보를 포함한 프로그램들 중 서로 다른 프로그램 개수는 221개, 성공적으로 분류된 프로그램들 중 서로 다른 프로그램 개수는 197개였다. 495개 대비 성공적인 분류율은 72.2%이다.

'두 번째 프로그램 집합'의 경우, 서로 다른 프로그램의 개수가 총 487개이고, 유용한 문자열 정보를 포함한 프로그램들 중 서로 다른 프로그램 개수는 436개, 성공적으로 분류된 프로그램들 중 서로 다른 프로그램 개수는 376개였다. 495개 대비 성공적인 분류율은 77.2%이다(표 5 참고).

### 4.4 기존 기법과의 비교

제안 기법의 효과적임을 비교하기 위해 [22]에서 사용된 기법과 비교하였다. 기존 [22]에서는, 문자열 추출 시 언어의 통일성을 고려하지 않았고 .rsrc 섹션의 RT_STRING 문자열만을 추출하였다. 비교 실험을 위해서 '첫 번째 프로그램 집합'을 사용한다.

비교 결과, 문자열 추출 비율은 제안 기법이

표 6. '첫 번째 프로그램 집합' 및 '두 번째 프로그램 집합'에서 서로 다른 프로그램의 수
Table 5. Different program score of '1st program group' and '2nd program group'

| 카테고리 | 첫 번째 프로그램 집합 | | | 두 번째 프로그램 집합 | | |
|---|---|---|---|---|---|---|
| | 서로 다른 프로그램 수 | 정제 후, 서로 다른 프로그램 수 | 정상 분류 시, 서로 다른 프로그램 수 | 서로 다른 프로그램 수 | 정제 후, 서로 다른 프로그램 수 | 정상 분류 시, 서로 다른 프로그램 수 |
| Audio Player | 33 | 28 | 24 | 54 | 49 | 45 |
| Browser | 22 | 16 | 15 | 51 | 38 | 27 |
| CD Writer | 32 | 23 | 18 | 55 | 54 | 47 |
| FTP | 29 | 24 | 22 | 55 | 54 | 46 |
| Image Viewer | 34 | 28 | 23 | 54 | 51 | 46 |
| Messenger | 35 | 24 | 24 | 55 | 44 | 39 |
| Text Editor | 36 | 31 | 28 | 54 | 49 | 42 |
| Video Player | 22 | 19 | 18 | 55 | 47 | 43 |
| Zip(Compress) | 30 | 28 | 25 | 54 | 50 | 41 |
| 합 계 (개) | 273 | 221 | 197 | 487 | 436 | 376 |
| 분류율 (%) 정제 대비 | 89.1 | | | 86.2 | | |
| 분류율 (%) 전체 대비 | 72.2 | | | 77.2 | | |

표 5. 제안 기법과 [22]의 기법 비교 결과 (실험 집합: 첫 번째 프로그램 집합)
Table 6. Compare result of suggested method and [22]

| 카테고리 | 제안 기법 | | | | [22] 기법 | | | |
|---|---|---|---|---|---|---|---|---|
| | 정제 (개수) | 55개 대비 정제율(%) | 분류 (개수) | 55개 대비 분류율 (%) | 정제 (개수) | 55개 대비 정제율(%) | 분류 (개수) | 55개 대비 분류율 (%) |
| Audio Player | 50 | 90.9 | 44 | 80 | 20 | 36.4 | 14 | 36.4 |
| Browser | 41 | 74.5 | 38 | 69.1 | 12 | 21.8 | 12 | 21.8 |
| CD Writer | 49 | 89.1 | 33 | 60 | 29 | 52.7 | 24 | 52.7 |
| FTP | 48 | 87.3 | 42 | 76.4 | 32 | 58.2 | 24 | 58.2 |
| Image Viewer | 48 | 87.3 | 43 | 78.2 | 33 | 60 | 26 | 60 |
| Messenger | 32 | 58.2 | 32 | 58.2 | 9 | 16.4 | 9 | 16.4 |
| Text Editor | 46 | 83.6 | 41 | 74.5 | 29 | 52.7 | 27 | 52.7 |
| Video Player | 47 | 85.5 | 46 | 83.6 | 17 | 30.9 | 14 | 30.9 |
| Zip(Compress) | 52 | 94.5 | 42 | 76.4 | 42 | 76.4 | 40 | 76.4 |
| 합계 또는 비율 | 413 | 83.4 | 361 | 72.9 | 223 | 45.1 | 190 | 38.4 |

83.4%, [22]의 기법이 45.1%였다. 전체 프로그램에 대한 분류율은, 제안 기법이 72.9%, [22]의 기법은 38.4%였다(표 6 참고).

제안 기법은, 문자열 추출률 및 소프트웨어 분류율 면에서 [22]의 기법보다 성능이 약 1.9배 개선되었음을 확인할 수 있었다.

## 5. 결론 및 향후 연구방향

본 논문에서는 MS 윈도우 실행파일들을 대상으로 문자열 빈도 정보에 기반한 효과적인 소프트웨어 분류 기법을 제안하였다. 문자열 빈도에

기반을 둔 점수를 LDF-IDF-ICF 수식을 사용하여 9개의 카테고리 별 문자열 점수표를 생성하고, 이 점수표를 기반으로 실행파일을 분류하였다. 총 495개의 실행파일들(9개의 카테고리 및 카테고리 별 55개 실행파일들로 구성)을 두 가지 종류의 집합으로 구성하여 제안 기법을 실험·평가하였다. 실험 결과 제안 기법이 72.9%-76.8%의 소프트웨어 분류율을 보였다. 특히, .rsrc 섹션에서만 문자열을 추출하여 소프트웨어를 분류하는 기존 연구에 비해, .text/.data/.rdata 섹션 등에서도 문자열을 추출하여 소프트웨어를 분류하는 본 기법이 상대적으로 더 높은 문자열 추출률과 소프트웨어 분류율을 보였다.

향후에는 제안 기법을 실제 소프트웨어 필터링 시스템에 적용하여 오버헤드와 처리시간을 측정하고, 실행압축이 된 실행파일도 고려한 소프트웨어 분류 기법에 대해 연구할 계획이다.

## 참 고 문 헌


[1] IDC, “The Dangerous World of Counterfeit and Pirated Software”, 2013.

[2] Dongjin Kim, Yesol Kim, Seong-je Cho, Minkyu Park, Sangchul Han, Guk-seon Lee, Young-sup Hwang, “An Effective intelligent Windows application filtering system using software similarity”, Soft Computing (A Fusion of Foundations, Methodologies and Applications)(Online version), Vol.20, No.5, pp.1821-1827, 2016.

[3] Yesol Kim, Jonghyuk Park, Seong-je Cho, Yunmook Nah, Sangchul Han, Minkyu Park, “Machine learning-based software classification scheme for efficient program similarity analysis”, In Proceedings of the 2015 Conference on research in adaptive and convergent systems, ACM, pp.114-118, 2015.

[4] ResourceExtract, http://www.nirsoft.net/utils/resources_extract.html, 2016.

[5] Stopwords, http://www.ranks.nl/stopwords, 2016.

[6] S. Choi, H. Park, “An Automated Classification Technique for Android Application Based on Software Montage”, Journal of KIISE : Computing Practices and Letters, Vol.18, No.11, pp.756-761, Nov. 2012.

[7] TF-IDF, https://en.wikipedia.org/wiki/Tf%E2%80%93idf, 2016.

[8] software.informer, http://software.informer.com/, 2016.

[9] alternativeTo, http://alternativeto.net/, 2016.

[10] sourceforge, https://sourceforge.net/, 2016.

[11] filehippo, http://filehippo.com/, 2016.

[12] FreewareFiles, http://www.freewarefiles.com/, 2016.

[13] uptodown, http://en.uptodown.com/, 2016.

[14] oldversion.com,http://www.oldversion.com/, 2016.

[15] IDApro, https://www.hex-rays.com/products ida/, 2016.

[16] IDAPython: https://code.google.com/p/idapython/, 2016.

[17] Language Identifier Constants and Strings, https://msdn.microsoft.com/en-us/library/dd318693(v=vs.85).aspx, 2016.

[18] SeongWook Kang, Hyungjoon Shim, Seong-je Cho, Minkyu Park, Sangchul Han, “A Robust and Efficient Birthmark-based Android Application Filtering System”, 2014 Research in Adaptive and Convergent System (ACM RACS 2014), Oct. 2014.

[19] D. Chae, J. Ha, S. Lee, S. Kim, “A Software Birthmarking System Using Static API-Call Frequency”, Journal of KIISE : Computing Practices and Letters, Vol.19, No.5, pp.298-302, May 2013.

[20] 황윤하, 한용만, 김동진, 조성제, 유해영, 우진운, “유사한 MS 윈도우 애플리케이션 탐지를 위한 소프트웨어 분류에 관한 연구”,

2013 한국소프트웨어감정평가학회 춘계학술대회, 2013.05.
[21] 한용만, 황윤하, 김동진, 최종천, 조성제, 유해영, 우진운, “동적 연결 라이브러리 기반 대회, 2013.05
[21] 한용만, 황윤하, 김동진, 최종천, 조성제, 유해영, 우진운, “동적 연결 라이브러리 기반의 MS 윈도우 애플리케이션 분류 기법,” 2013 한국소프트웨어감정평가학회 논문지, 9(1), 2013.12
[22] 김예솔, “프로그램 간 효율적인 유사성 분석을 위한 특징정보 기반의 소프트웨어 분류 기법,” 단국대학교 컴퓨터학과 석사학위 논문, 2015.06

## 저 자 소 개

조대희 (DaeHee Cho)

2014년 2월 단국대학교 멀티미디어공학과 학사 졸업
2016년 2월 단국대학교 컴퓨터학과 소프트웨어보안 석사과정 수료

<관심분야>
컴퓨터보안, 소프트웨어 지적재산권보호, 악성코드, 기계학습

조성제 (Seong-je Cho)

1989년 서울대학교 컴퓨터공학과 공학사
1991년 서울대학교 컴퓨터공학과 공학석사
1996년 서울대학교 컴퓨터공학과 공학박사
2001년 미국 University of California, Irvine 객원 연구원
2009년 미국 University of Cincinnati 객원 연구원
1997년 3월~현재 단국대학교 소프트웨어학과/컴퓨터학과 교수

<관심분야>
컴퓨터보안, 소프트웨어 지적재산권 보호, 시스템 소프트웨어, 스마트폰 보안 등