*Journal of The Korea Institute of Information Security & Cryptology* ISSN 1598-3986(Print)
VOL.29, NO.2, Apr. 2019 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2019.29.2.347

# 멀티모달 기반 악성코드 유사도 계산 기법

유 정 도,[1†] 김 태 규,[2] 김 인 성,[2] 김 휘 강[1‡]
[1]고려대학교 정보보호대학원, [2]LIG넥스원 사이버전연구팀

# Multi-Modal Based Malware Similarity Estimation Method

Jeong Do Yoo,[1†] Taekyu Kim,[2] In-sung Kim,[2] Huy Kang Kim[1‡]
[1]Graduate School of Information Security, Korea University, [2]LIG Nex1

## 요 약

사람의 DNA가 변하지 않는 것과 같이 사이버상의 악성코드도 변하지 않는 고유의 행위 특징을 갖고 있다. APT(Advanced Persistent Threat) 공격에 대한 방어수단을 사전에 확보하기 위해서는 악성코드의 악성 행위 특징을 추출해야 한다. 이를 위해서는 먼저 악성코드 간의 유사도를 계산하여 유사한 악성코드끼리 분류할 수 있어야 한다. 본 논문에서는 Windows OS 상에서 동작하는 악성코드 간의 유사도 계산 방법으로 'TF-IDF 코사인 유사도', 'Nilsimsa 유사도', '악성코드 기능 유사도', 'Jaccard 유사도'를 사용해 악성코드의 유형을 예측해보고, 그 결과를 보인다. 실험결과, 유사도 계산 방식마다 악성코드 유형에 따라 예측률의 차이가 매우 컸음을 발견할 수 있었다. 모든 결과에 월등한 정확도를 보인 유사도는 존재하지 않았으나, 본 실험결과를 이용하여 특정 패밀리의 악성코드를 분류할 때 어떤 유사도 계산 방식을 활용하는 것이 상대적으로 유리할지를 결정할 때 도움이 될 것으로 판단된다.

## ABSTRACT

Malware has its own unique behavior characteristics, like DNA for living things. To respond APT (Advanced Persistent Threat) attacks in advance, it needs to extract behavioral characteristics from malware. To this end, it needs to do classification for each malware based on its behavioral similarity. In this paper, various similarity of Windows malware is estimated; and based on these similarity values, malware's family is predicted. The similarity measures used in this paper are as follows: 'TF-IDF cosine similarity', 'Nilsimsa similarity', 'malware function cosine similarity' and 'Jaccard similarity'. As a result, we find the prediction rate for each similarity measure is widely different. Although, there is no similarity measure which can be applied to malware classification with high accuracy, this result can be helpful to select a similarity measure to classify specific malware family.

**Keywords:** similarity, malware analysis, API call sequence, Nilsimsa, Jaccard, TF-IDF

## I. 서 론

IT 분야의 발전과 4차 산업 혁명 시대의 도래에 대한 예고로 많은 이들은 앞으로 보게 될 변화를 기대하고 있다. 그러나 IT 분야의 기술이 발전하는 만큼 사이버 범죄의 사례도 늘어나고 있다. 하나의 시스템을 공격 목표로 하여 지속해서 해킹을 시도하는 APT(Advanced Persistent Threat) 공격은 정부 기관은 물론 여러 공기업과 사기업에 있어 큰 위협이 되고 있다. 이러한 사이버 범죄를 예방하고 사후 대처를 하는 것이 중요하다. 사이버 범죄가 발생했을 때는 '누가 범죄를 저지른 것인가'를 알아내는

Received(12. 27. 2018), Modified(02. 19. 2019), Accepted(02. 20. 2019)
† 주저자, opteryx25104@korea.ac.kr
‡ 교신저자, cenda@korea.ac.kr(Corresponding author)

것이 중요하다. 공격자가 누구인지 식별할 수 있어야, 이후에도 미리 공격 의도를 파악하여 대응책을 마련하여 후속 공격을 예방할 수 있다. 이와 관련하여, Mee Lan Han 등 [2] 은 웹 사이트 해킹에 대해 공격자를 특징짓는 연구를 하였다. CBR(Case-Based Reasoning) 알고리즘을 적용하여 침해사고 프로파일링 시스템을 구현하였다.

APT 공격에는 하나의 악성코드만 사용되기보다는 여러 종류의 악성코드가 동시에 사용되며, APT 공격 그룹마다 주로 사용하는 악성코드 역시 다르다. 따라서, APT 공격에 사용된 악성코드를 분석하면 APT 공격 그룹을 알아낼 확률이 높아진다.

현실 사회의 범죄에서도 범인의 DNA는 변하지 않은 것과 같이 사이버상의 악성코드도 변하지 않는 고유의 행위 특징을 갖고 있다. 사이버 게놈(Cyber-Genome) 기술을 통해 악성 행위 특징을 추출하여 악성코드 유전자 데이터베이스를 구축하면 새로 나타난 악성코드에서 도출된 해커의 특징을 이용해 분석 및 대응에 걸리는 시간을 절약하여 수많은 공격에 효과적으로 대응할 수 있으며, 잠재적으로 발생 가능한 APT 공격에 대한 방어수단을 사전에 확보할 수 있다.

이를 위해서는 악성코드 간의 유사도를 계산하여 유사한 악성코드끼리 분류할 수 있어야 한다. 본 논문에서는 Windows OS 상에서 동작하는 악성코드 간의 유사도 계산 방법을 제시한다. 현재까지 다양한 유사도 계산 방식이 존재하는데, 모든 악성코드 패밀리 진단에 범용적으로 잘 맞는 유사도 기법을 만든다는 것은 쉽지 않기 때문에 어떤 유사도 알고리즘이 어떤 패밀리의 악성코드를 진단하는데 변별력이 높게 쓰일 수 있는지 연구해 보았다.

## II. 문헌연구

### 2.1 악성코드 분석 자동화

Craig Miles 등 [4] 은 CTI(Cyber Threat Intelligence)를 적용한 악성코드 분석 시스템인 'VirusBattle'을 개발하였다. 해당 시스템은 악성코드의 binary, code, code semantics, dynamic behaviors, malware metadata, distribution sites, e-mail과 같은 정보를 활용하여 악성코드 간 상호연관성을 분석하였다. VirusBattle은 정적 분석 기법과 동적 분석 기법을 사용하는데, 동적 분석 기법은 'Dynamic Multipath Tracer Analysis'를 이용하여 동적 추적 트리를 생성하고 트리를 이용해 악성코드를 비교한다.

Dhilung Kirat, Giovanni Vigna [5] 는 백신 프로그램의 분석을 회피하는 악성코드의 행동을 분석하여 분석 회피 행동을 탐지할 방법을 제시했다. 해당 저자들은 분석 회피 시그니쳐를 추출하기 위해서 자동화된 기술을 이용하는 'MalGene'를 고안하였다. 분석 회피 시그니쳐 추출은 두 개의 분석 환경에서 악성코드를 실행함으로써 수행한다. 회피 이벤트가 발생한 지점을 파악하기 위해 두 분석 환경에서의 콜 시퀀스 간 편차가 가장 큰 부분을 찾는다. 백신 회피 부분부터 악성코드 행위가 가장 크게 달라져 해당 부분을 회피 지점으로 간주한다. 유사도 계산을 통해 이 해당 회피 지점을 찾는다.

### 2.2 악성코드 기능 분석

Hisham Shehata Galal 등 [6] 은 악성코드 API 콜 시퀀스에서 특정 API를 이용해 시퀀스를 추출하고 행동을 정의했다. 그다음, 정의된 행동 데이터에 기계학습 알고리즘을 적용하여 악성코드를 분류했다.

Naoto Kawaguchi, Kazumasa Omote [7] 는 악성코드의 실행 후 90초간 나타나는 행동을 기반으로 악성코드를 분류했다. 먼저 90초 동안 실행된 API를 추출하여 고유 API 피처 벡터를 뽑는다. 이후 악성코드의 기능별로 Symantec 네이밍 컨벤션으로 라벨링한다. 이후 API 피처 벡터와 레이블 기준으로 기능 피처 벡터를 선언한다. 해당 두 피처 벡터를 이용해 classification을 수행하였다.

André Grégio 등 [8] 은 악성코드 탐지 문제를 효율적으로 처리하기 위한 온톨로지 모델을 제안했다. 저자는 '공격 시작', '회피', '원격 제어', '자기 방어', '도용', '파괴' 등 여섯 가지 이벤트를 설정하여 이 이벤트 중 하나 이상이 나타나면 해당 악성코드는 의심스러운 행동을 보인다고 간주하였다.

Sanchit Gupta 등 [9] 은 악성코드를 5가지의 클래스로 나눈 후, 각 악성코드의 API 콜 시퀀스 유사도를 계산하여 유사도에 따라 어느 클래스에 속하는 악성코드인지 판별하는 프레임워크를 제안하였다.

Jae-wook Jang 등 [19] 은 모바일 기기에서 동작하는 악성코드에 대한 신속한 대응 방안을 제시

하였다. 신속한 대응을 위해, 악성 모바일 어플리케이션의 기능이 어떤 것인지를 파악하여 분석하는 것이 중요함을 강조하였다.

### 2.3 악성코드 유사도 측정

Dong Woo Goh와 Ki, Youngjoon 등 [1,3]은 Windows API를 추상화시킨 뒤 콜 시퀀스 유사도 계산을 수행하는 방법을 제안하고 있다. [1]에서는 쿠쿠 샌드박스 [16] 에서 후킹하는 Windows API를 비슷한 기능을 수행하는 API끼리 분류하여 추상화한 후, 'ssdeep', 'simhash', 'TLSH' 등의 LSH(Locality Sensitive Hashing) 기법을 적용하여 유사도를 계산한다. [3]에서는 2,722개의 Windows API를 기능이 비슷한 것끼리 묶어 26가지로 분류하였다. 추상화된 API 시퀀스에 LCS(Longest Common Subsequence) 알고리즘을 사용하여 악성코드의 유사도를 측정하였다.

Jing Liu 등 [10,11] 은 그래프를 이용한 악성코드 유사도 계산 방법을 제안하고 있다. [10]에서는 function call 그래프를 비교하는 알고리즘인 SFA(Similarity Flooding Algorithm) 알고리즘을 이용한다. [11]에서는 정적 함수 호출 그래프와 동적 행동 추적 그래프를 통해 피처를 선정한 후 그래프 간의 거리를 이용해 유사도를 계산하는 방법을 제안하였다. 그래프 매칭 알고리즘은 GED (Graph Edit Distance)를 이용한다.

Madhu K. Shankarapani 등 [12] 은 난독화가 적용된 변종 악성코드를 탐지하기 위해 'SAVE'와 'MEDiC'이라는 두 가지 악성코드 탐지 방법이 사용될 수 있음을 보였다.

Jiyong Jang 등 [13] 은 대규모 악성코드 분석 및 클러스터링을 위한 시스템 BitShred를 제안하였다. BitShred는 피처 해싱을 이용한다. 피처 해싱을 이용하면 피처 차원을 극적으로 감소시킬 수 있어 메모리 공간을 적게 차지한다.

## III. 유사도 계산

### 3.1 Jaccard 유사도

Jaccard 유사도(Jaccard similarity)란 두 집합의 유사도를 계산하는 방법의 하나다. Jaccard 유사도 값은 0과 1 사이의 값으로 나타나며, 두 집합이 유사할수록 1에, 두 집합이 다를수록 0에 가깝다. Jaccard 유사도의 수식은 다음과 같다.

$$(\text{Jaccard similarity}) = \frac{|A \cap B|}{|A \cup B|} \quad (1)$$

본 논문의 실험에서는 악성코드가 호출하는 API의 집합을 이용하여 Jaccard 유사도 비교를 한다. Jaccard 유사도의 특성상 악성코드가 호출하는 고유한 API 종류를 비교한다. 같은 악성코드 패밀리에 속하는 악성코드는 변종일지라도 사용하는 API 종류는 크게 다르지 않은 경우가 많다. 따라서 유사도 알고리즘 중 하나로 Jaccard 유사도를 포함했다.

### 3.2 Nilsimsa 알고리즘

#### 3.2.1 소개

Nilsimsa란 LSH(Locality Sensitivity hashing) 알고리즘의 하나로 E. Damiani 등 [14] 은 Nilsimsa 알고리즘의 기본적인 내용을 정리하였다. Nilsimsa는 본래 스팸 메일을 탐지하기 위한 알고리즘으로 개발된 것이다. MD5, SHA 등의 알고리즘은 해싱하고자 하는 메시지의 내용이 조금만 바뀌어도 해시값이 완전히 바뀌기 때문에 스팸 메일을 필터링하기 힘들다. Nilsimsa의 경우, 메시지 일부가 바뀌어도 바뀐 부분만 해시값이 달라지고 나머지 부분의 해시값은 그대로 유지가 된다.

Nilsimsa는 문자열을 입력으로 받아 n비트의 해시값을 출력한다. 계산과정에서 문자열을 3개씩 잘라 trigram으로 만든 후, 해당 trigram을 해싱하여 해당 해시값의 수를 세는 방식을 사용한다. 이때 하나의 trigram에 대한 해시값은 0부터 255 사이의 정숫값을 갖게 된다.

Nilsimsa의 유사도는 Nilsimsa의 출력값을 가지고 비교한다. 두 개의 Nilsimsa 출력값의 비트를 비교하여 같은 위치에 같은 비트가 많을수록 높은 유사도를 갖는다고 할 수 있다.

Nilsimsa 외에 자주 사용되는 LSH 알고리즘에는 simhash, TLSH 등이 있으나 본 논문에서는 그 외의 LSH 알고리즘은 사용하지 않았다.

악성코드의 변종은 원본 악성코드의 API 시퀀스와 크게 다르지 않은 경우가 많다. 변종은 원본 악성코드와 전체적인 동작 순서는 비슷하되 세세한 동작

에서 다르다. 즉, 악성코드가 호출하는 API의 순서는 변종과 원본이 전체적으로는 비슷하나 중간중간 다른 부분이 존재한다. 같은 악성코드 패밀리에 속하는 악성코드의 API 시퀀스 유사도를 비교하기 위해 유사도 알고리즘 중 하나로 Nilsimsa 알고리즘을 포함했다.

#### 3.2.2 Nilsimsa 알고리즘 수정

본 논문의 실험에서는 악성코드가 호출하는 Windows API마다 번호를 부여하여 trigram을 계산할 수 있도록 하였다. 기존의 Nilsimsa는 0부터 255까지의 정수만 사용할 수 있다. 하지만, 악성코드가 호출하는 Windows API의 종류는 256가지가 넘으므로 보다 많은 종류의 API 들을 다룰 수 있도록, 본 논문에서는 Nilsimsa를 수정하여, 0부터 511 사이의 정숫값이 나오도록 만들어 사용했다. 즉, 512가지의 서로 다른 입력에 대해 유사도를 구할 수 있도록 출력값이 512bit가 되도록 하였다.

### 3.3 코사인 유사도

코사인 유사도(Cosine similarity)는 두 벡터의 유사도를 계산하는 방법의 하나다. 두 벡터가 이루는 각도의 코사인값을 이용해 유사도를 나타내며, 벡터의 방향이 같으면 1, 방향이 반대이면 -1로 나타난다. 코사인 유사도의 식은 다음과 같다.

$$(\text{Cosine similarity}) = \cos(\theta) = \frac{A \cdot B}{\|A\| \, \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}} \quad (2)$$

본 논문의 실험에서는 TF-IDF 정보(제3.4절)를 이용한 유사도 계산, '기능' 정보(제4.2절)를 이용한 유사도 계산에 코사인 유사도를 사용한다. TF-IDF 정보와 '기능' 정보 모두 본 논문에서 벡터로 나타내기 때문에 유사도 알고리즘 중 하나로 코사인 유사도를 포함했다.

### 3.4 TF-IDF

TF-IDF(Term Frequency - Inverse Document Frequency)는 텍스트 마이닝에서 이

Table 1. Malware function list

| No. | malware's functionality | description |
|---|---|---|
| 1 | Adware | A function which presents unwanted advertisements to users |
| 2 | Anti-av | A function which detects antivirus software and evades it |
| 3 | Anti-dbg | A function which interrupts debugging process to analyze malware |
| 4 | Anti-sandbox | A function which checks if the running environment is in the sandbox (running in the security analyst's machine environment) |
| 5 | Backdoor | A function which gets privileges without any normal authentication |
| 6 | Botnet | A function which makes victim's computers zombie PCs to be controlled remotely |
| 7 | Code-injection | A function which injects malicious code into processes |
| 8 | Downloader | A function which downloads additional malware and infects victim's computers |
| 9 | Dropper | A function which installs files and executes them |
| 10 | Infostealer | A function which steals private information from the Internet browsers, FTP clients and e-mails |
| 11 | Keylogger | A function which uses event hooks to snoop all keyboard input events |
| 12 | MBR-destroyer | A function which corrupts the Master Boot Record area of a hard disk drive |
| 13 | Ransomware | A function which encrypts users' files (especially photo and document files) |
| 14 | Stealth | A function which hides itself to avoid detection |
| 15 | Worm | A function which copies itself to infect any adjacent devices through the network |

용하는 통계적 수치이다. 여러 문서가 존재할 때 특정 단어가 특정 문서 내에서 얼마나 중요한지 나타내는 값이다.

TF는 특정 단어의 특정 문서 내 빈도를 나타낸다. 빈도가 높을수록 해당 문서에서 해당 단어의 중요성이 높다고 볼 수 있다. 그러나 특정 문서만이 아닌, 다른 문서에서도 자주 사용되는 단어라면 해당 단어의 중요성은 비교적 낮다. DF란 특정 단어를 포함하는 문서의 수다. IDF는 DF의 역수다. TF-IDF는 TF와 IDF를 곱한 값이다.

본 논문의 실험에서는 악성코드를 '문서'로, 악성코드가 호출하는 각각의 API를 '단어'로 취급하여 TF-IDF를 계산한다. TF-IDF 벡터를 가지고 코사인 유사도 함수로 유사도를 계산한다.

## IV. '악성코드 기능'의 정의

### 4.1 개요

쿠쿠 샌드박스(Cuckoo Sandbox)는 오픈소스로 된, 악성코드 자동화 분석 시스템이다. 이 샌드박스 시스템은 악성코드의 특정 API 호출 및 파라미터 사용, 혹은 특정 레지스터 접근 등과 같이 특정 행동에 대해 모니터링을 할 수 있다[16]. 예를 들어, 악성코드가 Windows API 중에서 SetWindowsHookExA나 SetWindowsHookExW를 사용한다면 해당 악성코드는 '마우스 후킹'을 하는 기능을 갖는다고 의심해볼 수 있다. 이렇게 모니터링 대상이 되는 행동을 각각 '악성코드 기능'으로 명명했다. 본 논문에서는 쿠쿠 샌드박스에서 특정 행동에 대해 모니터링하는 대상을 '악성코드 기능' 혹은 간단히 '기능'으로 부른다. 악성코드 기능은 총 15가지로 정의하였고, 이를 쿠쿠 샌드박스의 모니터링 대상으로 추가했다. 기능 목록은 Table 1.과 같다. 기능의 종류와 탐지 방법은 기존 악성코드 관련 문헌 [7, 17, 18]을 참고했다. 제4.2절에서는 악성코드 기능을 어떻게 정의했는지 간단히 기술하였다.

### 4.2 기능 정의

#### 4.2.1 AdWare

Adware는 사용자의 동의 없이 광고를 띄우는 기능이다. 임의로 브라우저를 실행하고 설정을 변경하거나 브라우저 내 개인 정보 유출 등을 한다. 이 기능이 동작하는 순서는 다음과 같다.

먼저, Process32FirstW와 Process32NextW API를 호출하여 실행 중인 프로세스를 확인한다. 실행 중인 프로세스를 확인하면 RegOpenKeyExA나 NtOpenProcess API를 호출하여 열려있는 프로세스에 접근하거나 설치된 인터넷 브라우저를 탐색한다. 인터넷 브라우저의 위치를 알아내면 임의로 실행시켜 사용자의 개인 정보를 탈취하거나 브라우저 설정을 변경한다. NtWriteFile API를 호출하여 브라우저의 프로필 관련 파일을 새로 생성하거나 브라우저 내 통신 프로토콜 관련 설정을 변경하기도 한다.

#### 4.2.2 Anti-av

Anti-av는 백신 프로그램을 탐지한 후 해당 백신 프로그램의 탐지를 우회하거나, 혹은 직접 백신 프로그램을 종료시키는 기능이다. 이 기능이 동작하는 순서는 다음과 같다.

먼저 백신 프로그램을 탐지하기 위해 특정 레지스트리 키나 파일 리스트 중에 Kaspersky나 Avast와 같이 널리 알려진 백신 프로그램이 있는지 찾는다. 또한, LdrLoadDll이나 LdrGetDllHandle 모듈로 백신 프로그램의 라이브러리가 있는지 탐지한다. 만약 백신 프로그램 탐지에 성공한다면 NtTerminateProcess API를 호출하여 직접 해당 프로세스를 종료시키거나, OpenServiceW, ControlService API를 호출하여 동작 중인 백신 관련 서비스를 종료시킨다. 혹은 백신 프로그램을 직접 종료시키지 않고 자신의 프로세스 명을 'svchost.exe'와 같이 정상적인 프로세스로 위장하기도 한다.

#### 4.2.3 Anti-dbg

Anti-dbg는 백신 프로그램이나 기타 프로그램의 디버깅 여부를 확인하고, 이를 방지하는 기능이다. 주로 디버거가 존재하는지 확인한 후 해당 디버거를 삭제하거나 종료시킨다. 이 기능이 동작하는 순서는 다음과 같다.

먼저, 디버거가 동작 중인지 확인하기 위해 IsDebuggerPresent, LdrGetDllHandle, GetProcAddress API를 순서대로 호출한다. 동작 중인 디버거가 있음이 확인되면 디버깅을 방해하기 위해 NtT

erminateProcess(프로세스 종료), DeleteService(서비스 삭제), DeleteFileW(파일 삭제) 등의 API를 호출한다.

#### 4.2.4 Anti-sandbox

Anti-sandbox는 현재 악성코드가 실행되는 환경이 샌드박스인지 확인하는 기능이다. 샌드박스는 보통, 악성코드를 분석하기 위해 사용되는 임시적인 환경이므로 악성코드가 원하는 감염 대상이 아니다. 따라서 악성코드는 본 기능을 통해 분석을 회피하려고 한다. 이 기능이 동작하는 정해진 순서는 없으며, 회피 방법은 각각 독립적이다.

쿠쿠 샌드박스와 같이 특정 샌드박스 환경만이 가진 폴더나 파일의 유무를 파악하여 현재 악성코드가 실행되는 환경이 샌드박스인지 확인한다.

GetForegroundWindow API와 NtDelayExecution API를 사용하여 사용자가 윈도우 화면을 사용하는지 지속해서 탐지하는 방법도 존재한다. 보통, 샌드박스 환경은 자동화되어 실행되므로 사용자가 따로 존재하지 않는다. 위 두 함수를 사용하여 사용자의 사용 여부를 확인할 수 있다.

NtQuerySystemInformation API의 첫 번째 인자로 SystemProcessofPerformanceInformation을 전달하면 윈도우의 idle 시간을 알 수 있다. 샌드박스는 호스트 환경 위에서 동작하므로 프로세스 스위치 시 약간의 idle 시간이 생길 수 있다. 위 함수를 사용하여 idle 시간을 탐지한다면 Anti-sandbox 기능을 수행 중인 것으로 의심해볼 수 있다.

NtDelayExecution API를 사용하면 프로세스를 특정 시간 동안 중단시킬 수 있다. 샌드박스는 정해진 시간 동안 실행되므로 위 함수를 사용하여 악성코드 자신의 실행 시간을 늦추면 샌드박스 상에서의 분석을 방해할 수 있다.

#### 4.2.5 Backdoor

Backdoor는 일반적인 인증 과정을 거치지 않고 권한을 획득하여 악성 행위를 하는 기능이다. 따로 생성해놓은 악성 파일 등을 작동시켜 인증 과정을 우회한다. 이 기능이 동작하는 순서는 다음과 같다.

먼저 네트워크가 활성화된다. 네트워크가 활성화되면 NtAllocateVirtualMemory, NtProtectVirtualMemory API를 호출하여 원격 프로세스를 실행한다. 원격 프로세스를 실행하면 NtMapViewOfSection, VirtualProtectEx, NtWriteVirtualMemory, WriteProcessMemory 등의 API를 호출하여 메모리값을 변경한다.

#### 4.2.6 Botnet

Botnet은 사용자의 컴퓨터를 악성코드에 감염시켜 봇으로 만든 후, 네트워크 연결을 통해 원격 접근이나 통제할 수 있게 하는 기능이다. 이 기능이 성공적으로 동작하면 감염 PC는 제어하기 어렵고 통제권을 공격자에게 빼앗긴다. 이 기능이 동작하는 순서는 다음과 같다.

먼저 GetSystemInfo를 호출하거나, RegQueryValueEx를 호출하여 목표 시스템의 정보를 수집한다. 혹은, 시스템 바이오스 날짜나 Windows OS 버전 설치 날짜 등의 레지스트리 키에 접근한다. 이후 시스템 정보를 C&C 서버로 전송하기 위해 C&C 서버와 네트워크 연결을 수행한다. Botnet 프로세스를 수행하기 위해 IWbemServices_ExecMethod, IWbemServices_ExecMethodAsync API 등을 호출하여 새로운 프로세스를 생성한다. 새로 생성된 프로세스를 통해 파일이 드롭 되거나, 악성코드를 자동 실행 레지스트리에 추가하기도 한다.

#### 4.2.7 Code-injection

Code-injection은 기존 프로세스에 악성코드를 주입하여 공격자가 사용자의 PC의 권한을 얻거나 원하는 행동을 취할 수 있게 하는 기능이다. 이 기능이 동작하는 순서는 다음과 같다.

실행 중인 프로세스에 악성코드 인젝션을 할 경우 WriteProcessMemory, NtWriteVirtualMemory(메모리 쓰기 접근), NtMapViewOfSection, NtUnmapViewOfSection(섹션 메모리 접근), 혹은 NtSetContextThread, NtResumeThread(쓰레드 접근) 등의 API를 호출한다. 인젝션에 성공하면 정상 프로세스에 CreateRemoteThread API를 호출하여 악성 쓰레드를 생성한다. 결과적으로 기존 프로세스를 악성 프로세스로 변경시키고 Non-Child 프로세스를 실행하여 공격자는 원격 접근 권한을 얻는다.

#### 4.2.8 Downloader

Downloader는 목표가 되는 PC 내에서 동작하여 외부에 있는 악성코드를 목표 PC로 다운로드하여 감염시키는 기능이다. 이 기능이 동작하는 순서는 다음과 같다.

먼저 네트워크가 활성화된다. 다운로드할 수 있는 네트워크 환경이 되면 악성코드를 다운로드하기 위해 recv, recvfrom 등의 API를 호출한다. 이어서 ShellExecuteW API를 호출하여 다운로드한 악성코드를 실행한다.

#### 4.2.9 Dropper

Dropper는 단순히 파일만 생성하는 것이 아니라 실행할 수 있는 바이너리 파일이나 exe 파일을 설치 및 실행하는 기능이다. 이 기능이 동작하는 순서는 다음과 같다.

ShellExecuteExW나 CreateProcessInternalW를 호출하여 바이너리 파일을 생성하고 해당 파일을 실행한다. 일부 악성코드는 사용자의 AppData 폴더에 악성 바이너리를 생성하기도 한다.

#### 4.2.10 Infostealer

Infostealer는 기기에 설치된 인터넷 브라우저나 FTP 클라이언트 소프트웨어, 메일 소프트웨어 등에 저장된 개인 정보를 탈취하는 기능이다. 이 기능이 동작하는 순서는 다음과 같다.

먼저 정보 탈취 이전에 인터넷 브라우저 설치 경로를 검색하거나 권한 체크, 브라우저 보안 설정 변경 등을 수행한다. 인터넷 브라우저는 특정한 레지스트리에 키와 값을 생성하고, 특정 경로에 설치된다. 이러한 특징을 이용해 설치된 인터넷 브라우저의 위치를 파악한다. 권한 확인은 LookupPrivilegeValue API를 호출하여 수행한다. 브라우저 보안 설정은 브라우저마다 지정된 특정 레지스트리 값을 변경함으로써 수정할 수 있다. 개인 정보 탈취는 FireFox, Chrome, Opera 등 인터넷 브라우저나, FTP Explorer 등의 FTP 클라이언트, Outlook Express 등의 메일 클라이언트에 저장된 정보를 빼내는 방식으로 진행한다.

#### 4.2.11 Keylogger

Keylogger는 키보드 입력 이벤트를 후킹하여 키보드 입력 정보를 저장하는 기능이다. 여기서는 Keylogger 기능에 마우스 로깅까지 포함한다. 이 기능이 동작하는 순서는 다음과 같다.

키보드 후킹과 마우스 후킹은 SetWindowsHookExA나 SetWindowsHookExW API를 호출함으로써 수행한다. 해당 API의 'hook_identifier'의 값에 따라 키보드 혹은 마우스 후킹을 수행한다. 이후 C&C 서버와 네트워크 연결을 수행한다. 키보드나 마우스 이벤트를 후킹하여 얻은 로그 정보는 C&C 서버로 전송한다.

#### 4.2.12 MBR-destroyer

MBR-destroyer는 하드디스크의 Master Boot Record 영역을 비정상적인 값으로 변경하는 기능이다. MBR 영역은 컴퓨터 부팅 시 가장 먼저 읽는 부분이기도 하므로 비정상적인 값으로 변경될 경우 부팅이 불가능할 수 있는 중요한 부분이다. 이 기능이 동작하는 순서는 다음과 같다.

MBR 영역을 열고 쓸 때는 각각 CreateFile과 WriteFile API를 사용한다. 이때 CreateFile API의 파라미터로 전달하는 파일명은 '\\.\PhysicalDrive0'부터 '\\.\PhysicalDrive25' 중 하나이다. 대부분은 부팅 루틴이 있는 첫 번째 물리적 하드디스크의 MBR 영역을 가리키는 '\\.\PhysicalDrive0'를 전달한다. CreateFile에서 리턴된 파일 핸들값을 WriteFile로 전달하여 MBR 영역을 공격자가 원하는 값으로 변경한다.

#### 4.2.13 Ransomware

Ransomware는 사용자의 파일을 모두 암호화하여 이후 복호화를 위한 개인키의 대가로 돈을 요구하는 기능이다. 이 기능이 동작하는 순서는 다음과 같다.

FindFirstFile이나 FindNextFile API를 호출하여 목표 대상으로 하는 파일을 탐색한 후 암호화하기 쉽도록 대상 파일들을 모두 하나의 폴더로 이동시킨다. 파일 이동은 MoveFileWithProgress이나 MoveFileWithProgressTransactedW API 등을 이용한다. 이후 SetFileAttributes API를 호

출하여 파일 속성을 ENCRYPTED로 변경시킨 뒤, CreateFile이나 CopyFile API를 호출하여 새 파일을 생성한다.

#### 4.2.14 Stealth

Stealth는 악성코드가 악성 행위를 사용자에게 드러내지 않음으로써 자신의 행위를 들키지 않도록 하는 기능이다. 백그라운드에서 프로세스를 실행하거나 흔한 프로세스로 위장하는 것이 일반적이다. 많은 악성코드가 은폐 기능을 보이며, 이로 인해 악성코드 실행의 인지가 어렵다. 이 기능이 동작하는 순서는 다음과 같다.

백그라운드 프로세스를 실행하기 위해 ShellExecuteExW나 CreateProcessInternalW API 함수가 호출된다. 이때의 파라미터는 각각 hidden과 CREATE_NO_WINDOW이다. 백그라운드 프로세스의 이름은 'csrss.exe', 'lasass.exe'와 같은 흔한 프로세스의 이름을 사용하여 위장한다.

#### 4.2.15 Worm

Worm은 자신을 복제하여 네트워크상에 전파해 다른 장비를 감염시키는 기능이다. 이 기능이 동작하는 순서는 다음과 같다.

PC가 부팅되었을 때 자동으로 실행되도록 자동 실행 레지스트리나 디렉토리에 악성코드를 등록시킨다. 그다음 악성 행위를 담은 파일을 패킹한다. 패킹이 완료되면 네트워크 포트를 열어 패킷을 전송하며, 일부 악성코드의 경우, 패킷 전송 이전에 악성코드 전파를 하기 위해 주변 장치를 탐색하기도 한다.

Table 2. The number of samples for each malware label

| Malware Label | # of Samples |
|---|---|
| AdWare | 204 |
| Backdoor | 203 |
| Downloader | 200 |
| Trojan-Ransom | 203 |
| Worm | 210 |
| Virus | 203 |

## V. 데이터셋 및 환경

### 5.1 데이터셋

2018년 4월에서 2018년 9월까지 멀웨어즈닷컴[15]에 제출된 악성코드 중에서 Windows 32-bit 실행 파일 포맷의 악성코드를 1차로 선택했다. 선택된 악성코드 중에서 Kaspersky의 악성코드 네이밍을 기준으로 'AdWare', 'Backdoor', 'Downloader', 'Trojan-Ransom', 'Worm', 'Virus' 레이블에 속하는 악성코드를 레이블 당 200여 개씩, 총 1223개의 악성코드를 선택했다. 위 6개의 레이블을 선택한 이유는 해당 레이블 외의 악성코드는 그 수가 적었고, 'Trojan'의 경우에는 상대적으로 그 수가 매우 많아 레이블에 속하는 악성코드만이 갖는 특징을 찾기 어려울 것으로 예상하여 제외했다.

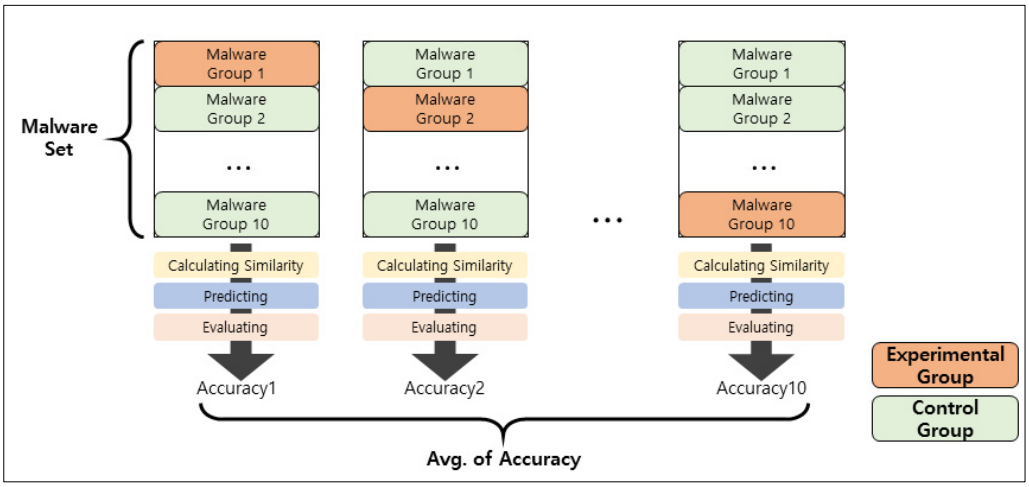


Fig. 1. How to conduct each experiment

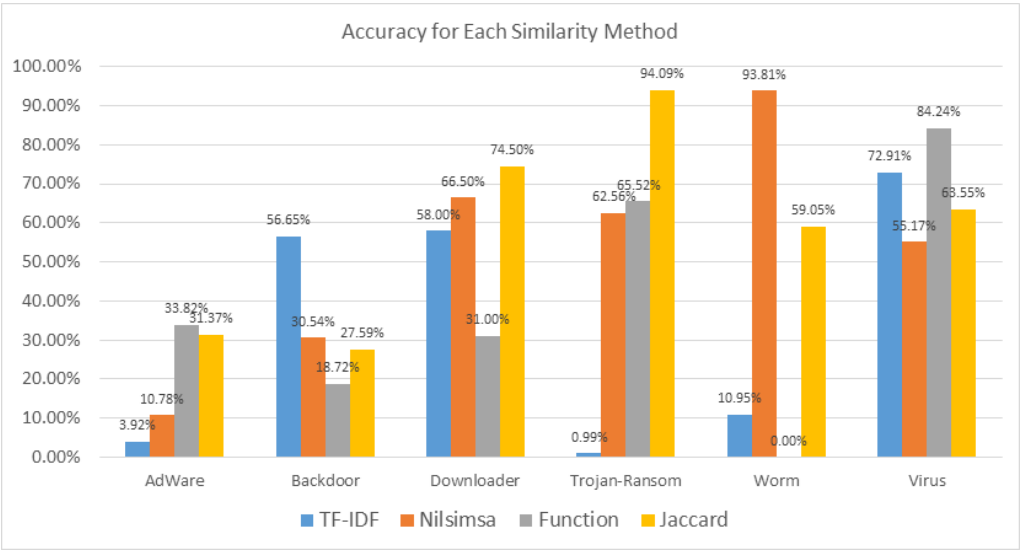


Fig. 2. Accuracy for each similarity method

백신 회사가 악성코드에 레이블을 부여할 때 별도로 정해진 네이밍 컨벤션 관련 표준이 존재하지 않는다. 즉, 다른 백신 회사가 같은 악성코드에 대해 Kaspersky가 부여한 레이블과 다를 수 있다는 것을 고려해서 한 악성코드에 Kaspersky 외에 국내 백신 회사인 안랩과 이스트소프트, 그리고 세계적으로 인지도가 높은 Bitdefender와 Avast의 레이블을 추가로 부여하여 예측률을 비교해보았다. 그러나 백신 회사마다 레이블을 부여하는 규칙이나 사용하는 용어가 서로 달라 대부분의 악성코드에 추가로 레이블을 부여하기에 한계가 있었고, 추가로 레이블을 부여하더라도 Kaspersky의 레이블과 대동소이한 경우가 많았다. 각 악성코드에 Kaspersky의 레이블만 부여한 예측 결과와 다른 백신 회사의 레이블을 추가로 부여한 예측 결과가 크게 다르지 않으므로, 본 논문의 실험에서는 Kaspersky의 레이블만 사용한 실험만을 서술하였다.

레이블별 악성코드 샘플의 수는 Table 2. 와 같다.

### 5.2 분석 환경

악성코드 분석은 오픈소스 악성코드 자동 분석 시스템인 쿠쿠 샌드박스 [16] 를 이용했다. 악성코드가 실제 실행되는, 쿠쿠 샌드박스의 내부 환경은 Windows 7 32-bit로 설정했다.

쿠쿠 샌드박스는 악성코드가 호출하는 Windows API 중에서 원하는 Windows API만을 후킹할 수 있다. 쿠쿠 샌드박스는 기본적으로 364개의 Windows API를 후킹한다. 여기에 56개의 Windows API를 추가로 후킹할 수 있게 수정했다. 추가한 API의 선정 기준은 Michael Sikorski와 Andrew Honig의 연구 [17] 를 참조하였고, 그중에서 악성코드가 자주 이용할만한 API를 선택해 추가했다.

쿠쿠 샌드박스가 모니터링하는 대상이 되는 '악성코드 기능'은 제4.2절에서 기술한 것처럼 총 15가지로 정의하여 분석 시스템에 추가했다.

## VI. 실험 및 결과

### 6.1 실험

실험은 총 네 가지로 진행했다. 각 실험에서는 하나의 유사도 계산 방법을 가지고 악성코드의 유사도를 계산했다. 각 실험의 전체적인 진행은 Fig.1.과 같다. 먼저, 각 레이블에 속하는 악성코드 샘플을 임의로 균등하게 10개의 그룹으로 나눈 뒤 9개의 그룹은 대조군으로 둔다. 나머지 1개의 그룹을 실험군으로 둔 뒤, 해당 그룹에 속하는 각 악성코드 샘플에 대해 레이블별 대조군과의 유사도를 계산하여 유사도의 평균이 가장 높은 레이블로 예측한 후, 예측 레이

Table 3. Prediction rate for TF-IDF

| | | Real Label | | | | | |
|---|---|---|---|---|---|---|---|
| | | AdWare | Backdoor | Downloader | Trojan -Ransom | Worm | Virus |
| Predicted Label | AdWare | 3.92% | 0.49% | 4.50% | 2.96% | 6.67% | 0.99% |
| | Backdoor | 21.57% | **56.65%** | 27.00% | **85.22%** | 15.24% | 16.26% |
| | Downloader | 23.53% | 7.39% | **58.00%** | 0.49% | 0.48% | 1.97% |
| | Trojan-Ransom | 22.06% | 4.43% | 1.00% | 0.99% | 20.95% | 6.40% |
| | Worm | 4.90% | 3.94% | 5.00% | 0.99% | 10.95% | 1.48% |
| | Virus | **24.02%** | 27.09% | 4.50% | 9.36% | **45.71%** | **72.91%** |

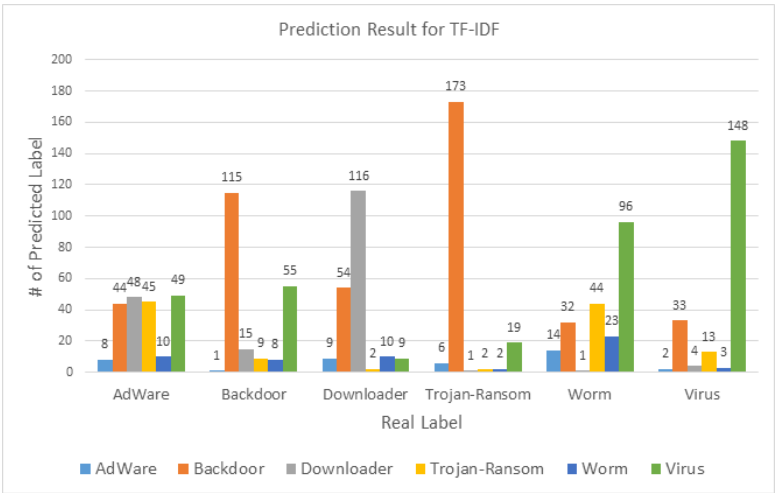


Fig. 3. The number of predicted labels for each real label by TF-IDF

블이 실제 레이블과 같은지의 여부를 평가한다. 이전에 나눈 10개의 그룹에서 대조군과의 유사도를 계산한 그룹은 대조군으로 넣고 나머지 9개의 그룹에서 1개의 그룹을 실험군으로 선택하여 나머지 그룹(대조군)과의 유사도를 계산한다. 이 과정을 총 10번의 거친 뒤, 정확도의 평균을 계산한다.

실험 1에서는 TF-IDF를 이용하여 코사인 유사도를 계산했다. 악성코드가 호출하는 Windows API는 '단어'로 취급하고, 악성코드는 '문서'로, 레이블별 악성코드 샘플 그룹은 '문서군'으로 취급한다. 레이블별로 악성코드 샘플의 API 쿼리 벡터값을 계산한 뒤, 실험군에 속하는 각 악성코드 샘플의 TF-IDF 벡터값과의 코사인 유사도를 계산했다. 레이블별로 유사도 값을 합한 뒤 가장 높은 유사도가 나타난 레이블로 예측했다.

실험 2에서는 Nilsimsa를 이용했다. Nilsimsa에 적용하기 위해 Windows API 각각에 고유한 정숫값을 부여했다. 이때 부여한 수는 0부터 418까지이다. 악성코드가 호출한 API 시퀀스를 각 API에 부여된 정숫값의 시퀀스로 변환한 뒤 Nilsimsa값을 계산했다. 계산한 Nilsimsa값을 가지고 Nilsimsa 유사도를 계산했다. 레이블별로 유사도 값을 합한 뒤 가장 높은 유사도가 나타난 레이블로 예측했다.

실험 3에서는 악성코드 기능을 가지고 코사인 유사도를 계산했다. 쿠쿠 샌드박스의 모니터링 결과를 바탕으로 각 샘플이 위에서 정의한 15가지의 기능을 갖는지 아닌지를 확인하여 해당 기능을 가지면 1로, 기능을 가지지 않으면 0으로 값을 매겼다. 0과 1로 이루어진 15차원의 벡터로 나타낸 뒤, 벡터의 코사인 유사도를 계산했다. 레이블별로 유사도 값을 합한 뒤 가장 높은 유사도가 나타난 레이블로 예측했다.

실험 4에서는 Jaccard 유사도를 이용했다. 각 악성코드 샘플이 한 번이라도 호출한 Windows API를 집합으로 나타내어 Jaccard 유사도를 계산했다. 레이블별로 유사도 값을 합한 뒤 가장 높은 유사도가 나타난 레이블로 예측했다.

Table 4. Prediction rate for Nilsimsa

| | | Real Label | | | | | |
|---|---|---|---|---|---|---|---|
| | | AdWare | Backdoor | Downloader | Trojan -Ransom | Worm | Virus |
| Predicted Label | AdWare | 10.78% | 0.99% | 2.00% | 0.49% | 0.00% | 0.00% |
| | Backdoor | 0.00% | 30.54% | 2.00% | 6.40% | 0.00% | 3.94% |
| | Downloader | 15.20% | 0.00% | **66.50%** | 0.00% | 0.48% | 0.49% |
| | Trojan-Ransom | 1.96% | 1.97% | 4.50% | **62.56%** | 1.90% | 1.48% |
| | Worm | **72.06%** | **66.50%** | 25.00% | 30.54% | **93.81%** | 38.92% |
| | Virus | 0.00% | 0.00% | 0.00% | 0.00% | 3.81% | **55.17%** |

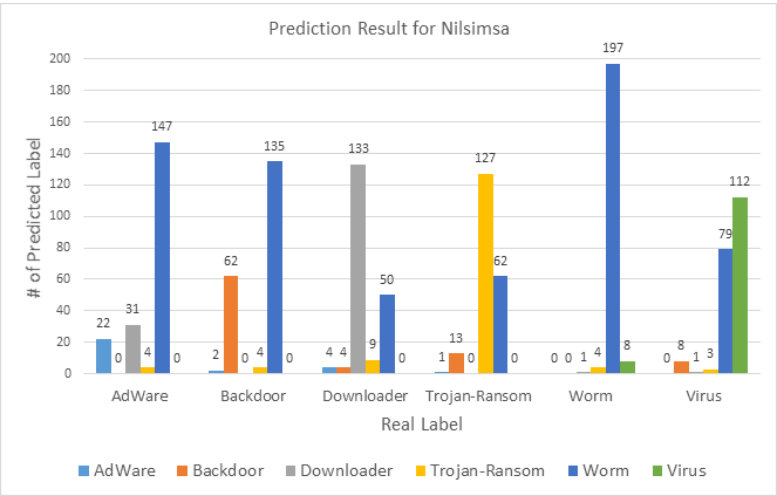


Fig. 4. The number of predicted labels for each real label by Nilsimsa

### 6.2 결과

실험 1부터 실험 4까지의 예측 정확도 결과는 Fig.2.와 같다. 악성코드 레이블별 정확도는 유사도 계산 방식에 따라 차이가 크게 났다. 실험 1인 TF-IDF 방식은 Backdoor(56.65%), Downloader(58.00%), Virus(72.92%)에 속하는 악성코드 샘플에 대해 다른 레이블과 비교하여 상대적으로 높은 예측률을 보여주었다. 그러나 AdWare(3.92%), Trojan-Ransom(0.99%), Worm(10.95%)은 매우 낮은 예측률이 나타났다. Table 3.과 Fig.3.은 TF-IDF 방식으로 예측했을 때의 예측 결과이다. 예측률이 가장 낮았던 Trojan-Ransom(0.99%)은 Backdoor로 잘못 예측한 경우(203개 중 173개)가 가장 많았다. Backdoor와 Trojan-Ransom은 Windows API의 TF-IDF 관점에서 서로 비슷한 것으로 보인다. 실제로도 현재의 많은 Trojan 프로그램들이 Backdoor 기능이 있다고 볼 수 있다. 더불어 백신 회사가 악성코드에 대해 레이블을 부여할 때에 표준이 있지는 않고 해당 악성코드가 가진 기능 중 하나를 기준으로 부여하는 경우가 일반적이므로 이에 부합한다고 볼 수 있다.

실험 2인 Nilsimsa 방식은 Worm에서 매우 높은 예측률(93.81%)을 보였으나 AdWare(10.78%)와 Backdoor(30.54%)에 속하는 악성코드 샘플은 다른 레이블과 비교하여 매우 낮은 예측률을 보여주었다. Table 4.와 Fig.4.는 Nilsimsa 방식으로 예측했을 때의 예측 결과이다. 예측률이 가장 낮은 AdWare와 Backdoor는 Worm으로 잘못 예측한 경우(각각 204개 중 147개, 203개 중 135개)가 가장 많았다. Downloader(200개 중 50개), Trojan-Ransom(203개 중 62개), Virus(203개 중 79개) 역시 두 번째로 많이 예측한 레이블이 Worm이었다. 많은 레이블이 Windows API의 Nilsimsa 관점에서 Worm과 유사한 것으로 나타났다.

실험 3인 악성코드 기능 방식은 Virus(84.24%)

Table 5. Prediction rate for malware function

| | | Real Label | | | | | |
|---|---|---|---|---|---|---|---|
| | | AdWare | Backdoor | Downloader | Trojan -Ransom | Worm | Virus |
| Predicted Label | AdWare | 33.82% | 6.90% | 1.50% | 1.48% | 11.43% | 6.90% |
| | Backdoor | 0.98% | 18.72% | 8.50% | 15.27% | 0.00% | 1.97% |
| | Downloader | 5.88% | 25.12% | 31.00% | 3.94% | 10.00% | 1.97% |
| | Trojan-Ransom | 3.43% | 8.37% | 3.50% | **65.52%** | 10.00% | 4.93% |
| | Worm | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| | Virus | **55.88%** | **40.89%** | **55.50%** | 13.79% | **68.57%** | **84.24%** |

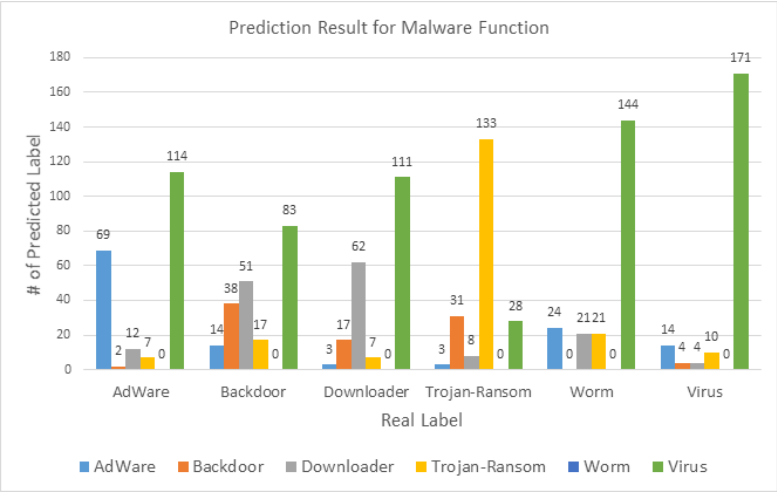


Fig. 5. The number of predicted labels for each real label by malware function

에서 가장 높은 예측률을 보였고 그 뒤를 이어 Trojan-Ransom(65.52%)에서 상대적으로 높은 예측률을 보였으나 나머지 레이블에 대해서는 매우 낮은 예측률을 기록했다. Table 5.와 Fig.5.는 악성코드 기능 방식으로 예측했을 때의 예측 결과이다. Virus와 Trojan-Ransom을 제외한, 예측률이 낮은 레이블은 모두 Virus로 잘못 예측한 경우가 많았다.

실험 4인 Jaccard 방식은 Trojan-Ransom(94.09%)에서 가장 높은 예측률을 보였다. 반면, AdWare(31.37%), Backdoor(27.59%)에서 상대적으로 낮은 예측률을 보였다. Table 6.과 Fig.6.은 Jaccard 방식으로 예측했을 때의 예측 결과이다. 상대적으로 예측률이 높았던 Downloader, Worm, Virus는 Trojan-Ransom으로 잘못 예측하는 경우가 많았다. Jaccard 유사도 계산 방식을 고려했을 때, 이들 레이블에 속하는 악성코드에서 사용하는 API 종류 자체는 Trojan-Ransom과 크게 차이나지 않는 것으로 보인다.

### 6.3 결과 고찰

네 가지 유사도 측정 방법 중 모든 악성코드 레이블에 대해 괜찮은 예측률을 보여주는 유사도 측정 방법은 존재하지 않았다. 위와 같은 실험결과가 도출된 것에는 아래와 같은 이유가 있을 것으로 생각된다.

첫째로, 요즘 악성코드들은 기능이 종합적이다. 백신 프로그램의 발전으로 악성코드는 공격 성공률을 높이기 위해 다양한 기능을 포함하고 있다. 예를 들어, 어떤 악성코드의 주요 목적은 사용자의 파일을 암호화시키는 'Ransomware'이지만 공격을 성공시키기 위해 이 악성코드에 Trojan 및 Stealth 기능, Downloader 및 Dropper 기능을 탑재하는 것이다. 따라서, 해당 악성코드의 레이블이 'Trojan-Ransom'라고 해도 API 시퀀스를 분석해보면 오히려 'Downloader'와 비슷할 수 있다.

Table 6. Prediction rate for Jaccard

| | | Real Label | | | | | |
|---|---|---|---|---|---|---|---|
| | | AdWare | Backdoor | Downloader | Trojan -Ransom | Worm | Virus |
| Predicted Label | AdWare | **31.37%** | 0.99% | 1.50% | 0.00% | 0.00% | 0.99% |
| | Backdoor | 1.96% | 27.59% | 0.00% | 1.97% | 0.48% | 1.97% |
| | Downloader | 28.92% | 11.82% | **74.50%** | 0.99% | 3.81% | 0.99% |
| | Trojan-Ransom | 20.10% | **40.89%** | 18.50% | **94.09%** | 29.52% | 20.20% |
| | Worm | 15.20% | 17.24% | 1.50% | 2.46% | **59.05%** | 12.32% |
| | Virus | 2.45% | 1.48% | 4.00% | 0.49% | 7.14% | **63.55%** |

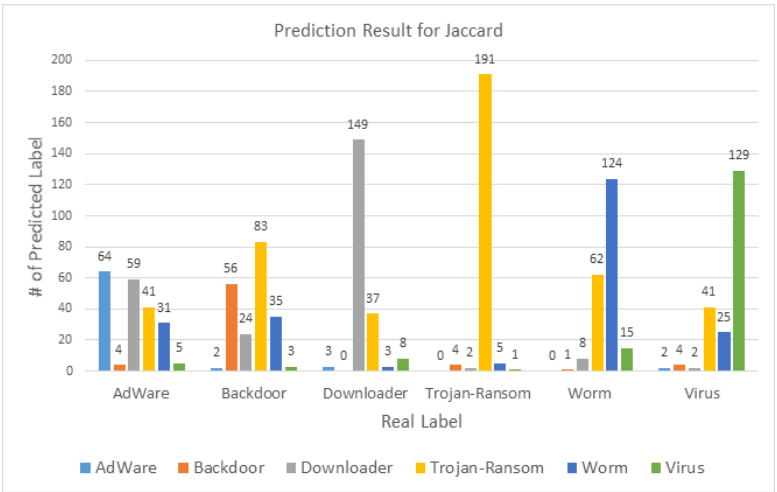


Fig. 6. The number of predicted labels for each real label by Jaccard

둘째로, 악성코드 패밀리의 분류 기준이 백신 회사마다 다르고 주관적이다. 본 논문에서는 Kaspersky 사의 악성코드 분류를 기준으로 실험을 진행했다. 그러나, 악성코드 패밀리 분류 기준은 여전히 사람의 판단이 들어가기 때문에 주관적이므로 정확하지 않을 수 있다. 또한, 위에서 언급한, 악성코드의 다양한 기능 탑재라는 이유와 맞물려 악성코드 레이블은 백신 회사마다 크게 차이 나는 경우가 많으므로 악성코드마다 부여된 레이블이 꼭 정확하다고는 말할 수 없다.

마지막으로, 악성코드 레이블 분류 예측 실험을 위해 API 시퀀스 분석만을 했다는 한계점이다. 악성코드의 행위 정보 분석에는 악성코드가 호출하는 API를 분석하는 것이 상당 부분 도움이 된다. 그러나 API 콜 시퀀스 외에도 네트워크 통신 IP나 송수신 패킷 정보, 생성 파일 등 악성코드의 행위를 분석하는 데 도움이 될 수 있는 피쳐들이 상당수 존재한다. API 콜 시퀀스가 여러 문헌에서 핵심적인 악성 행위 특징을 파악하기 위해 많이 활용되나, 최근 복합적인 악성 기능을 가진 악성코드들을 API 시퀀스 정보만을 이용하여 분류하기에는 적합하지 않고, 이 외에도 부가적인 데이터를 이용하여 multi-modal 분석을 하는 것이 필요하다는 것을 알 수 있다.

## VII. 결 론

본 논문에서는 네 가지의 유사도 계산을 이용하여 악성코드의 유사도를 계산해 보았다.

첫 번째 방법인 TF-IDF 코사인 유사도 계산은 악성코드를 '문서'로, 악성코드가 호출하는 API 하나하나를 '단어'로, 악성코드 레이블을 '문서군'으로 취급하여 TF-IDF를 계산한 뒤 코사인 유사도를 이용하는 방법이다. 이 방법은 Virus 레이블에 속하는 악성코드에 대해 준수한 예측률을 보여주었으나 AdWare, Trojan-Ransom, Worm에 대해서는 낮은 예측률을 보여주었다. 이는 Backdoor와

Trojan-Ransom이 Windows API의 TF-IDF 관점에서 서로 비슷하기 때문으로 보인다. 실제로 현재 많은 Trojan 프로그램들에 Backdoor 기능이 있으며 또한, 백신 회사가 악성코드에 레이블을 부여할 때에 어떠한 표준을 따르는 것이 아니라 해당 악성코드가 가진 기능 중 하나를 기준으로 부여하는 경우가 일반적이다. AdWare의 경우 AdWare에 속하는 악성코드만이 갖는 Windows API 특징이 약하거나, 혹은 실험에서 후킹한 420개의 Windows API 목록에 AdWare에 속하는 악성코드가 자주 호출하는 특정 Windows API가 포함되어 있지 않기 때문으로 보인다.

두 번째 방법인 Nilsimsa 방식은 악성코드가 호출하는 API 시퀀스에 Nilsimsa 해싱을 적용한 후 유사도를 구하는 방법이다. 이 방법은 Worm 레이블에 속하는 악성코드에 대해 높은 예측률을 보여주었으나 AdWare와 Backdoor에 대해서는 저조한 예측률을 보여주었다. AdWare에 속하는 악성코드와 Backdoor에 속하는 악성코드는 대부분 Worm으로 잘못 예측하는 경우가 많았다. AdWare와 Worm 역시 Trojan과 마찬가지로 Backdoor 기능이 있는 경우가 많다. Worm은 스스로 복제하여 네트워크를 통해 빠르게 전파되며, 불특정 다수의 기기에 Backdoor를 설치한다. AdWare 역시 광고수익을 위해 빠르게 배포되며, 종종 Backdoor 기능이 있어 해커가 악용하기도 한다. 따라서 LSH 알고리즘인 Nilsimsa 유사도를 계산했을 때 비슷한 악성코드로 잘못 예측되는 경우가 많은 것으로 보인다.

세 번째 방법인 악성코드 기능 방식은 본 저자가 정의한 '악성코드 기능'의 유무에 따라 벡터로 나타내어 코사인 유사도를 구하는 방법이다. 이 방법은 Virus 레이블에 속하는 악성코드에 대해 높은 예측률을 보여주었으나 AdWare, Backdoor, Downloader, Worm에 대해서는 낮은 예측률을 보여주었으며, 특히 Worm의 경우에는 예측률이 0%를 기록했다. 이는 Virus가 독립적인 실행이 불가능하기 때문으로 보인다. Virus는 보통, 독립적인 실행이 불가능하므로 주로 다른 유형의 악성코드의 Dropper나 Downloader에 의해 설치된 후 실행된다. Virus라는 레이블에 Virus 레이블에 해당하는 악성코드뿐만 아니라 해당 Virus 악성코드를 설치하는 악성코드를 포함했을 가능성이 있다. 특히 Virus는 스스로를 복제한다는 점에서 Worm과 성격이 비슷하여 Worm의 예측률이 낮았다고 생각된다.

네 번째 방법인 Jaccard 방식은 악성코드가 호출하는 API 집합을 Jaccard 유사도로 계산하는 방법이다. 이 방법은 Trojan-Ransom 레이블에 속하는 악성코드에 대해 높은 예측률을 보여주었으나 AdWare와 Backdoor에 대해서는 상대적으로 저조한 예측률을 보여주었다. Backdoor는 Trojan-Ransom으로 잘못 예측되는 경우가 많았는데, 이 역시 실험1의 결론에서 언급한 것과 마찬가지로 많은 Trojan 프로그램들이 Backdoor 기능을 갖기 때문으로 보인다. 또한, AdWare에 속하는 악성코드가 자주 호출하는 특정 Windows API을 후킹하고 있지 않기 때문으로 보인다.

레이블별로 정확도가 70% 이상인 유사도 알고리즘은 Table 7.과 같다.

네 가지 유사도 방법인 TF-IDF, Nilsimsa, 악성코드 기능 코사인 유사도, Jaccard 유사도를 사용하여 악성코드 레이블을 예측한 결과, 위에서 서술한 것처럼 각 방법에 대해 한계점이 드러났다. TF-IDF, Nilsimsa, Jaccard 유사도는 악성코드가 호출하는 Windows API를 이용했다. 특히 모든 유사도 계산 방법에 대해, AdWare 레이블에 속하는 악성코드의 예측률이 가장 낮았다. 이는 AdWare에 속하는 악성코드만이 호출하는 특정한 API가 없었기 때문으로 보인다. 본 논문에서는 실험을 위해 제5.2.절과 같이 420개의 API만 후킹할 수 있도록 했다. 후킹되지 않은 API의 존재로 예측률에 크게 영향을 줬을 것으로 생각된다. 해당 유사도 알고리즘의 한계점을 보완하고 더욱 정확한 실험을 위해서는 다양한 종류의 API를 후킹하도록 할 필요가 있다. 악성코드 기능에 대한 코사인 유사도 계산은 제4.2.절에서 정의한 기능을 이용했다. 이는 현재 정의한 15개의 기능을 좀 더 세밀하게 재정의하고, 15개 외의 다른 악성코드의 기능을 추가하는 방법으로 그 한계점을 보완할 수 있을 것으로 생각된다.

본 논문에서는 텍스트 마이닝에서 사용되는

Table 7. Recommended similarity algorithms for each label (whose accuracy is over 70%)

| Label | Recommended Similarity Algorithm |
|---|---|
| Downloader | Jaccard |
| Trojan-Ransom | Jaccard |
| Worm | Nilsimsa |
| Virus | Malware function, TF-IDF |

TF-IDF 유사도를 악성코드의 API 시퀀스에 적용해보았다. 비록 아직 그 예측률은 부족했으나 연구가 더 필요한 부분이고, 더욱이 다른 분야에서 자주 사용되는 방법론을 가져와 적용해봤다는 점이 긍정적이다.

악성코드 역시 의도가 다를 뿐, 다른 정상 프로그램과 마찬가지로 프로그래머의 작품이다. 악성코드 자체를 '문서'라는 관점으로 본다면 TF-IDF 유사도 외에도 텍스트 마이닝에서 사용되는 다양한 방법론을 가져와 연구에 적용해볼 만한 하다고 생각한다. 또한, 악성코드를 '문서'라는 관점으로 봤을 때 해당 작품(문서)을 만든 '작가'인 프로그래머 혹은 APT 공격 그룹을 추측하는 데에도 도움이 될 것이다. 이는 서론에서 밝힌 연구 의도인 'APT 공격 그룹을 알아낼 확률을 높인다.'라는 것에 부합한다. 향후 사이버 게놈 연구에는 악성코드를 대상으로 텍스트 마이닝 등 다른 연구 분야에서 사용되고 있는 방법론을 적용한 연구를 해볼 예정이다.

## References


[1] Dong Woo Goh and Huy Kang Kim, "A Study on Malware Clustering Technique Using API Call Sequence and Locality Sensitive Hashing," Journal of the Korea Institute of Information Security & Cryptology, 27(1), pp. 91-101. Feb. 2017.

[2] Mee Lan Han, Deok Jin Kim, and Huy Kang Kim, "Applying CBR algorithm for cyber infringement profiling system," Journal of the Korea Institute of Information Security & Cryptology, 23(6), pp. 1069-1086. Dec. 2013.

[3] Ki, Youngjoon, Eunjin Kim, and Huy Kang Kim, "A novel approach to detect malware based on API call sequence analysis," International Journal of Distributed Sensor Networks, vol. 11, no. 6, 659101, 2015.

[4] Craig Miles and Vivek Notani, "VirusBattle: State-of-the-art malware analysis for better cyber threat intelligence," 2014 7th International Symposium on Resilient Control Systems (ISRCS), pp. 1-6, 2014.

[5] Dhilung Kirat and Giovanni Vigna, "MalGene: Automatic extraction of malware analysis evasion signature," Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM, pp. 769-780, 2015.

[6] Hisham Shehata Galal, Yousef Bassyouni Mahdy, and Mohammed Ali Atiea, "Behavior-based features model for malware detection," Journal of Computer Virology and Hacking Techniques, vol. 12, no. 2, pp. 59-67. 2016.

[7] Kawaguchi, Naoto, and Kazumasa Omote, "Malware function classification using APIs in initial behavior," 2015 10th Asia Joint Conference on Information Security. IEEE, pp. 138-144, 2015.

[8] André Grégio and Mario Jino, "Ontology for malware behavior: A core model proposal," 2014 IEEE 23rd International WETICE Conference, pp. 453-458, 2014.

[9] Sanchit Gupta, Harshit Sharma, and Sarvjeet Kaur, "Malware Characterization Using Windows API Call Sequences," International Conference on Security, Privacy, and Applied Cryptography Engineering. Springer. Cham, pp. 271-280, 2016.

[10] Jing Liu and Zhijian Huang, "Malware Similarity Analysis Based on Graph Similarity Flooding Algorithm," Advances in Computer Science and Ubiquitous Computing. Springer. Singapore, pp. 31-37, 2015.

[11] Jing Liu and Xingkong Ma, "Using a Fine-Grained Hybrid Feature for Malware Similarity Analysis,"

Advances in Computer Science and Ubiquitous Computing. Springer. Singapore, pp. 54-60, 2016.

[12] Madhu K. Shankarapani and Subbu Mukkamala, "Malware detection using assembly and API call sequences," Journal in computer virology, vol. 7, no. 2, pp. 107-119, 2011.

[13] Jiyong Jang, David Brumley, and Shobha Venkataraman, "Bitshred: feature hashing malware for scalable triage and semantic analysis," Proceedings of the 18th ACM conference on Computer and communications security. ACM, pp. 309-320, 2011.

[14] E. Damiani and P. Samarati, "An Open Digest-based Technique for Spam Detection," ISCA PDCS, pp. 559-564, 2004.

[15] malwares.com, "malware analysis" www.malwares.com, 2018.

[16] cuckoo, "malware analysis sandbox" cuckoosandbox.org, 2018.

[17] Michael Sikorski and Andrew Honig, "Practical Malware Analysis: A Hands-On Guide to Dissecting Malicious Software," No Starch Press 2012.

[18] ESTsecurity, "malware reports" www.estsecurity.com/securityCenter/malwareReport/1, 2018.

[19] Jae-wook Jang and Huy Kang Kim, "Function-Oriented Mobile Malware Analysis as First Aid," Mobile Information Systems, 2016.

## 〈저자소개〉

유 정 도 (Jeong Do Yoo) 학생회원
2018년 2월: 고려대학교 컴퓨터학과 졸업
2018년 3월~현재: 고려대학교 정보보호대학원 정보보호학과 통합과정
〈관심분야〉 시스템보안, 온라인게임 보안

김 태 규 (Taekyu Kim) 정회원
2000년 2월: 중앙대학교 컴퓨터공학 학사
2006년 5월: the University of Arizona 컴퓨터공학 석사
2008년 5월: the University of Arizona 컴퓨터공학 박사
〈관심분야〉 Cybersecurity Killchain and TTP(Tactics, Techniques, and Procedures), 임베디드 시스템보안, System Modeling and Simulation

김 인 성 (In-sung Kim) 정회원
2000년 8월: 중앙대학교 산업정보학과 졸업
2008년 2월: 고려대학교 정보경영공학전문대학원 정보경영공학과 석사
2016년 3월~현재: 중앙대학교 융합보안학과 박사과정
〈관심분야〉 사이버보안, 위협 인텔리전스

김 휘 강 (Huy Kang Kim) 종신회원
1998년 2월: KAIST 산업경영학과 학사
2000년 2월: KAIST 산업공학과 석사
2009년 2월: KAIST 산업및시스템공학과 박사
2004년 5월~2010년 2월: 엔씨소프트 정보보안실장, Technical Director
2010년 3월~2014년 12월: 고려대학교 정보보호대학원 조교수
2015년 1월~현재: 고려대학교 정보보호대학원 부교수
〈관심분야〉 온라인게임 보안, 네트워크 보안, 네트워크 포렌직, 침입탐지시스템, 봇넷탐지