Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.29, NO.3, Jun. 2019 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2019.29.3.531

악성코드 패밀리 분류를 위한 API 특징 기반 앙상블 모델 학습

이 현 종,† 어 성 율, 황 두 성‡
단국대학교

API Feature Based Ensemble Model for Malware Family Classification

Hyunjong Lee,† Seongyul Euh, Doosung Hwang‡
Dankook University

요 약

본 논문에서는 악성코드 패밀리 분류를 위한 훈련 데이터의 특징을 제안하고, 앙상블 모델을 이용한 다중 분류 성능을 분석한다. 악성코드 실행 파일로부터 API와 DLL 데이터를 추출하여 훈련 데이터를 구성하며, 의사 결정 트리 기반 Random Forest와 XGBoost 알고리즘으로 모델을 학습한다. 악성코드에서 빈번히 사용되는 API와 DLL 정보를 분석하며, 고차원의 훈련 데이터 특징을 저차원의 특징 표현으로 변환시켜, 악성코드 탐지와 패밀리 분류를 위한 API, API-DLL, DLL-CM 특징을 제안한다. 제안된 특징 선택 방법은 데이터 차원 축소와 빠른 학습의 장점을 제공한다. 성능 비교에서 악성코드 탐지율은 Random Forest가 93.0%, 악성코드 패밀리 분류 정확도는 XGBoost가 92.0%, 그리고 정상코드를 포함하는 테스트 오탐률은 Random Forest와 XGBoost가 3.5%이다.

ABSTRACT

This paper proposes the training features for malware family analysis and analyzes the multi-classification performance of ensemble models. We construct training data by extracting API and DLL information from malware executables and use Random Forest and XGBoost algorithms which are based on decision tree. API, API-DLL, and DLL-CM features for malware detection and family classification are proposed by analyzing frequently used API and DLL information from malware and converting high-dimensional features to low-dimensional features. The proposed feature selection method provides the advantages of data dimension reduction and fast learning. In performance comparison, the malware detection rate is 93.0% for Random Forest, the accuracy of malware family dataset is 92.0% for XGBoost, and the false positive rate of malware family dataset including benign is about 3.5% for Random Forest and XGBoost.

Keywords: Malware Detection, Malware Classification, Feature Selection, Tree-based Ensemble

I. 서 론

정보 기술의 발전에 따라 정보의 생산성과 접근성이 높아지면서 웹 사이트 또는 네트워크 채널 등의 다양한 경로로 악성코드가 빠르게 확산되고 있다. 악성코드는 사용자 컴퓨터에 침입하여 개인 정보 유출, 의도하지 않은 네트워크 트래픽 발생, 파일 삭제, 시스템 성능 저하, 메일 자동 발송, 원격 제어 등 비정상적인 행위를 실행한다.

악성코드의 실행 행위(execution behavior)에 따라서 패밀리(malware family)가 구분되며, 다양한 변종 악성코드가 존재한다. 변종 악성코드는

Received(01. 22. 2019), Modified(03. 25. 2019), Accepted(05. 11. 2019)
† 주저자, infin94@hotmail.com
‡ 교신저자, dshwang@dankook.ac.kr(Corresponding author)

API 호출 구조, 쓰레기 코드 삽입 등을 통해 파생되어 탐지를 회피한다. 이러한 변종 악성코드의 수는 매해 증가하는 추세이다. 2018년 시만텍의 “인터넷 보안 위협 보고서”에 따르면 사용자의 동의 없이 악성코드를 설치하는 다운로더 악성코드의 변종은 작년 대비 92.0% 증가했다. 뿐만 아니라, 문서 파일을 암호화시켜, 복호화에 대한 비용을 요구하는 랜섬웨어 악성코드의 변종은 46.0% 증가했다[1].

기존 악성코드 탐지 방법은 서명 탐지(signature detection)[2], 이상 탐지(anomaly detection)[3]로 분류된다. 서명 탐지는 악성코드의 행위 규칙 또는 해쉬 값을 이용한 서명(signature)을 데이터베이스에 저장하고 비교 분석을 통해 악성코드를 탐지하는 방법이다. 기존 악성코드에서 파생된 신변종 악성코드는 해쉬 값이 달라 서명 탐지가 불가능하다[4]. 이상 탐지는 정상코드의 분석을 통해 사용되는 API 규칙을 도출하고, 도출된 규칙을 만족하지 않는 경우 악성코드로 판단한다. 그러나 신종 악성코드에 대해서 사용되는 API 규칙의 범위를 정의하기 어렵고 오탐률(false positive rate)이 높다[4]. 기존 탐지 방법의 신변종 탐지와 오탐률 문제점을 극복하기 위해 기계 학습을 이용한 악성코드 탐지 기술이 활발히 연구되고 있다[5,6,7,8].

윈도우 운영체제에서 실행되는 대부분 악성 프로그램은 윈도우에서 제공하는 동적 라이브러리를 이용한다. 동적 라이브러리의 API(Application Programming Interface) 함수를 호출하여 프로세스, 네트워크, 메모리 등을 제어한다[9]. 동일 패밀리 악성코드들은 비슷한 행위를 수행하기 위해 같은 종류의 API 함수를 호출한다. 악성코드에서 사용된 API 분석은 악성코드 패밀리 분류에 이용될 수 있다[5,6].

본 논문에서는 악성코드 패밀리와 정상코드로 다중 분류 학습 문제를 구성하고 3가지의 API 관련 특징을 추출해 악성코드 탐지 모델을 학습한다. 2장에서는 악성코드 탐지 모델 관련 연구를 소개한다. 3장에서는 기계 학습을 이용한 악성코드 탐지 방법과 API 관련 특징 3가지를 제안하고 사용된 데이터를 기술한다. 4장에서는 악성코드 탐지를 위한 앙상블 모델을 평가 및 분석한다. 5장에서는 결론과 향후 연구 방향을 제시한다.

Ⅱ. 관련연구

악성코드 데이터에서 특징을 추출하기 위한 분석 방법은 정적 분석과 동적 분석으로 나누어진다[9]. 정적 분석은 대상 파일을 실행하지 않은 상태에서 분석하며 파일의 PE(Portable Executable) 구조, 어셈블리 코드 등을 분석하는 방법이다. 동적 분석은 프로그램을 실행시켜 API 호출, 프로세스, 메모리, 네트워크 리소스 등을 모니터링하여 이상 행위를 분석하는 방법이다. 악성코드의 동적 분석은 운영 시스템을 보호하기 위해 가상환경에서 수행한다[5]. Table 1은 데이터 수, 클래스 수, 사용된 특징, 학습 모델, 테스트 분석 등의 관련 연구의 비교이다.

Konrad Rieck 외 4명은 수집한 10,072개의 악성코드를 안티 바이러스 Avira AntiVir[10]에 의해 14개 클래스로 분류하여 다중 분류 문제를 구성했다. CWSandbox[11] 가상 환경을 구축해 수집된 데이터에 대해 동적 분석을 수행했다. 악성 프로그램이 실행되면 API 후킹을 이용해 호출되는 API와 입력된 파라미터를 기록했다[12]. 기록된 보고서로부터 사용된 API와 파라미터의 빈도 값으로 특징 벡터를 구성하고 SVM(Support Vector Machine) 학습 알고리즘으로 모델을 학습시켰다. 패밀리 분류 평가에서 0.88의 분류 정확도를 보고하였다[5].

Bowen Sun 외 4명은 VirusShare[13]로부터 65,536 개의 악성코드를 확보했다. 악성코드 패밀리 분류를 위한 다중 분류 문제의 정의는 패밀리 클래스는 Kaspersky[14] 기준을 적용했다. 수집된 데이터 중 PE 헤더가 없는 악성코드를 제외하고 총 9개 클래스의 다중 분류 문제를 구성했다. 바이트 기반 특징 3개, 어셈블리 기반 특징 4개, PE 기반 특징 4개를 추출해 종류별 특징 벡터를 구성했다. 큰 차원의 특징을 축소시키기 위해 Random

Table 1. Comparison of related works

Ref.	No. of data	No. of classes	Feature	Acc.
[5]	10,072	14	API, Parameter	88.0%
[6]	65,536	9	OP code	93.0%
[7]	22,741	9	Assembly, keyword	99.0%
[8]	210	2	API	97.0%

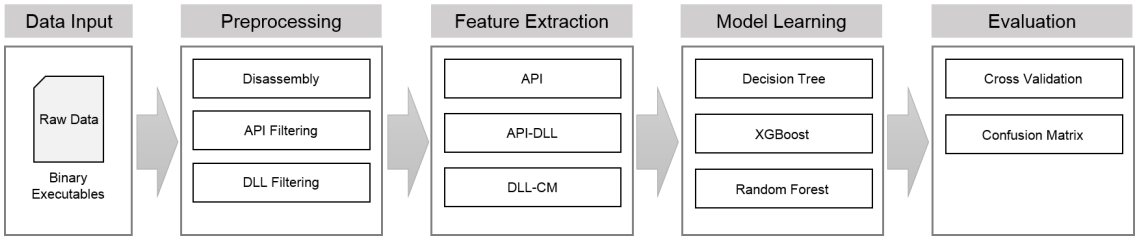

Fig. 1. Malware family detection process

Forest[15]로 특징 벡터를 축소시켰다. 분류 성능 비교를 위해 Bayesian, Random Forest, SVM, 의사 결정 트리(decision tree), k-nearest neighbor, AdaBoost를 평가했다. 실험에서 특징 추출의 효과를 입증하였으며, 특징 선택 후 분류 실험에서 Random Forest가 약 0.93으로 성능이 가장 우수했다[6].

Mansour Ahmadi 외 3명은 Malware Challenge 데이터 셋[16]에 XGBoost[17] 탐지 모델 학습을 적용하였다. 악성코드 학습 데이터에서 정적 분석을 통해 바이트 기반 특징 5개와 디스어셈블리 기반 특징 8개를 선택하여 13개 특징과 조합에 따른 분류 성능을 평가했다. 실험은 교차검증(cross-validation)으로 진행되었다. 대부분 선택된 훈련 특징에서 0.95 이상의 분류 정확도를 보였으며 모든 특징을 훈련 데이터로 구성 시 0.99의 정확도 성능을 보고했다[7].

Veeramani R 외 1명은 300개의 정상 프로그램과 210개의 악성 프로그램을 수집해 분류 문제를 구성했다. IDA(Interactive Disassembler) Pro[18] 도구를 이용해 준비된 데이터의 어셈블리 코드를 생성했다. 어셈블리코드에서 호출된 API를 추출했으며 분류 성능을 위해 정상과 악성 클래스에 관련된 API를 선정해 특징으로 사용했다. 선정된 API 특징의 n-gram 구조를 생성하고 파라미터 n=1,2,..4 일 경우 탐지 성능을 비교했다. 학습 알고리즘으로 SVM을 선택하여, 1-gram 특징이 0.97의 정확도로 성능이 가장 우수했다[8].

Ⅲ. 제안 방법

악성코드 탐지 모델의 학습 과정은 데이터 전처리(data preprocessing), 특징 추출(feature extraction), 모델 학습(model learning), 모델 평가(model evaluation) 단계를 적용한다(Fig 1). API 기반 악성코드 분류 모델의 학습 과정을 단계에 따라 기술한다.

3.1 데이터 준비

멀웨어스닷컴[19]으로부터 102,963개 악성코드와 정상코드 데이터를 확보했다. 다중 분류 문제의 클래스를 정의하기 위해 안티 바이러스 소프트웨어 Kaspersky[14]의 분류를 사용했다. 102,963개 데이터 중 Kaspersky가 정상 프로그램이라고 판단한 30,324개의 데이터는 제외시켰다. 데이터가 많이 수집된 패밀리 5개를 선정했다. 고려된 데이터 중 44,373개의 “generic” 키워드가 포함된 Kaspersky 진단명은 여러 패밀리에 속해있는 복합성 악성코드이므로, 패밀리별 학습을 위해 학습 데이터 집합에서 제외시켰다. 10,000개의 정상코드를 포함시켜 구성된 훈련 데이터는 총 38,266개이며 Table 2와 같다.

Table 2. Dataset for malware family classification

Malware family	No. of data	Ratio
Adware	8,412	21.98%
Trojan	7,134	18.64%
Downloader	8,654	22.62%
Virus	2,692	7.03%
Backdoor	1,374	3.59%
Benign	10,000	26.13%
Total	38,266	100.0%

3.2 데이터 전처리

API를 추출하는 방법은 2가지로 수행되었다. 첫 번째로 PE 구조의 IAT(Import Address Table)

를 분석해 추출하는 방법이다. IAT는 해당 실행 파일이 사용하는 DLL(Dynamic Link Library)과 API 목록을 갖고 있다. PE 구조 내 .idata 섹션에 위치하며 해당 실행 파일에서 호출하는 DLL과 API 정보를 추출할 수 있다. 두 번째는 분석 대상 실행 파일을 어셈블리 코드로 변환해 API를 추출하는 방법이다. 어셈블리 코드에서 jmp 또는 call 명령어로 API를 호출하기 때문에 명령어의 파라미터를 분석하면 API 추출이 가능하다[9].

IAT를 분석하는 방법은 API의 사용 여부를 판단할 수 있지만 호출 빈도를 계산하기 어려워 특징 벡터를 이진 데이터(binary data)로 구성한다. 반면에, 어셈블리 분석은 API의 호출 빈도를 계산해 특징 벡터를 연속형 데이터(continuous data)로 표현한다. 사용된 디스어셈블리 툴은 Radare2[20]이며, 어셈블리 코드를 분석해 API 정보를 추출하였다.

3.3 특징 추출

학습 시간과 성능은 특징 차원에 크게 의존된다. 수집된 악성코드에서 사용되는 모든 API를 특징 벡터 구성에 사용하면 훈련 벡터가 고차원으로 구성된다. 고차원의 훈련 데이터 생성은 편협된 데이터 분포의 가능성이 높아 데이터 희소성(data sparsity)이 높아질 수 있다. 고차원의 희소 행렬 형태의 데이터 학습은 과적합(overfitting) 현상으로 인해 높은 학습 성능을 보장할 수 없다. 따라서 빈도 분석을 통해 낮은 빈도의 DLL과 API 특징을 제거하는 특징 축소 기법을 제안한다.

3.3.1 API

API 특징은 API 함수를 기준으로 필터링한 후 구성된 특징이다. 악성코드는 의도된 목적을 위해 특정한 함수를 호출해야 한다[21]. 예를 들면, 악성코드에서 윈도우 프로세스를 생성하기 위해서 "CreateProcess" 같은 API 함수를 호출하거나 네트워크 소켓 통신을 하기 위해 "WSAStartup" 함수를 호출한다. 필터링 목록 구성을 위해 악성코드에서 자주 사용되는 API 목록을 분석하여 총 49,158종류의 API가 사용되었다. 전체 사용 빈도가 0.10% 이상인 API를 선택하여 Table 3과 같이 218개의 API 목록을 구성하였다. 추출된 API가 목록에 존재하지 않으면 제외해 특징 벡터의 차원을 축소한다.

Table 3. APIs frequently used in malware

No.	Name	Frequency	Ratio
1	GetLastError	552,535	2.20%
2	GetProcAddress	349,623	1.39%
3	CloseHandle	330,714	1.31%
4	SendMessageW	307,877	1.22%
5	LeaveCriticalSection	286,195	1.14%
6	SendMessageA	266,766	1.06%
7	EnterCriticalSection	259,806	1.03%
8	RegCloseKey	224,691	0.89%
9	MultiByteToWideChar	217,928	0.87%
10	lstrlenW	211,035	0.84%
11	WideCharToMultiByte	207,674	0.83%
12	IsWindow	206,868	0.82%
13	WriteFile	196,323	0.78%
…	…	…	…
218	PostMessageA	24,634	0.10%
Total		15,600,880	61.97%

3.3.2 API-DLL

API-DLL 특징은 악성코드에서 빈번히 사용된 DLL을 기준으로 필터링한 후 구성된다. DLL은 마이크로소프트 윈도우에서 사용하는 동적 라이브러리로 네트워크, 메모리, 프로세스 등을 제어할 수 있는 함수를 제공한다. 악성코드에선 새로운 프로세스를 생성하거나 네트워크 리소스를 사용하기 위해 윈도우에서 제공하는 DLL이 사용된다. 학습에 사용될 API를 선정하기 위해 수집된 악성코드 데이터 집합에서 빈번히 사용되는 DLL을 분석했다. 사용된

Table 4. DLLs frequently used in malware

No.	DLL Name	Frequency	Ratio	# APIs
1	KERNEL32	10,799,529	42.91%	947
2	USER32	7,268,561	28.88%	663
3	GDI32	1,695,760	6.74%	333
4	ADVAPI32	986,327	3.92%	374
5	OLEAUT32	894,541	3.55%	152
6	MSVBVM60	751,166	2.98%	204
7	OLE32	382,779	1.52%	159
8	COMCTL32	321,401	1.28%	33
9	MSVCRT	208,191	0.83%	190
10	SHELL32	180,044	0.72%	139
11	QT5CORE	175,359	0.70%	3
12	WS2_32	125,505	0.50%	109
13	WININET	96,148	0.38%	124
14	GDIPLUS	91,759	0.36%	332
15	SHLWAPI	68,326	0.27%	239
16	WSOCK32	64,141	0.25%	39
Total		24,109,537	95.79%	4,040

DLL의 전체 개수는 1,276개이며 사용 빈도가 0.25% 이상인 DLL을 선택해 Table 4와 같이 DLL 목록을 구성했다. 목록을 구성하는 DLL에서 사용된 API 종류는 총 4,040 종류로 나타났다. 추출된 API가 필터링 목록의 DLL이 제공하지 않는 API이면 제외해 특징 벡터의 차원을 축소한다.

3.3.3 DLL-CM(Call Matrix)

DLL은 동적 라이브러리로써 관련된 API를 포함한다. 악성코드 제작자는 이러한 DLL을 프로그램에 링크시켜 API를 사용한다. 어셈블리 코드 내 API 호출 순서는 DLL 호출 순서로 변환이 가능하다. DLL-CM(Call Matrix) 특징은 프로그램의 DLL의 호출 관계를 표현한다. 특징 추출 이전에 DLL 종류의 집합 $D=\{d_1,d_2,...,d_N\}$을 사전에 계산한다. 여기서 집합 D의 원소 $d_i(1\le i\le N)$은 DLL이다. 어셈블리 코드에서 DLL 호출 순서 $S=\langle s_1,s_2,...,s_L\rangle$를 추출한다. 여기서 $s_j(1\le j\le L)$는 j번째 호출된 DLL이며, $s_j\in D$이다. 추출된 DLL 호출 순서 S에서 $G=\{\langle s_j,s_{j+1}\rangle|1\le j\le L-1\}$를 계산한다. 2-gram 구조인 G는 순서가 존재하는 벡터에서 2개의 인접한 원소에 대한 조합을 순서대로 계산한다.

Algorithm 1 DLL-CM Algorithm

```
procedure DLLCM(D, S)   ▷ Where D - DLL set, S - DLL call sequence
    L ← length of S
    N ← length of D
    Initialize matrix M[N][N] to zero
    p ← 0
    t ← 0
    i ← 0
    while i ≤ L − 1 do
        p ← S[i]
        t ← S[i + 1]
        M[p][t] ← M[p][t] + 1
        i ← i + 1
    return M
```

Fig. 2. Pseudo code for DLL-CM

빈도 분석을 통해 인접한 원소의 연관성을 파악할 수 있다. 원소 조합인 $\langle s_j,s_{j+1}\rangle$을 $N\times N$ 크기의 행렬 M 좌표로 사용한다. 해당 좌표의 행렬 원소 값을 1씩 증가시켜 2-gram 구조를 2차원 행렬로 변환한다. Fig 2는 DLL-CM 특징을 준비하는 함수이다.

Fig 3은 클래스 별 DLL-CM 특징의 평균값을 가시화한 영상이다. 영상의 행은 현재 호출된 DLL이며, 열은 다음에 호출된 DLL을 나타낸다. 선정된 16개의 DLL을 특징 구성에 사용해 영상의 크기는 256(16x16)이다. 영상의 픽셀 값이 클수록 호출의 전후 관계의 빈도가 높다. 영상으로부터 DLL 호출 관계를 분석했을 때 Kernel32, User32, GDI32 DLL의 사용 비중이 높았으며 클래스 별로 값의 차이가 존재한다. 현재 행위에서 특정 DLL을 호출하

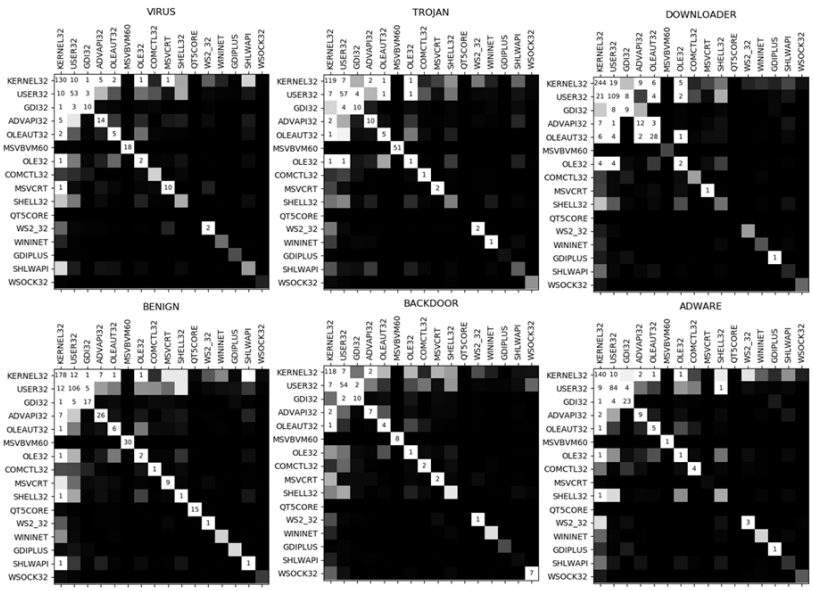

Fig. 3. Average number of features per class for DLL-CM

고 다음 행위에서 동일한 DLL을 다시 호출하는 경우가 빈번하게 나타나며, 훈련 특징 구성 시 대칭행렬과 희소행렬의 특징이 분석되었다.

3.4 트리 기반 앙상블 학습 알고리즘

악성코드 학습을 위해 의사 결정 트리 알고리즘과 트리 기반 앙상블 알고리즘인 XGBoost와 Random Forest를 사용한다. 의사 결정 트리 알고리즘은 학습 데이터로부터 클래스를 분류할 수 있는 속성을 반복적으로 찾아 최대 깊이만큼 트리를 구성한다. 최적의 분류를 수행할 수 있는 속성을 결정하기 위해 정보 획득(information gain) 방법인 엔트로피를 이용한다. 테스트 데이터는 구성된 트리 노드의 분류 규칙에 따라 터미널 노드로 맵핑된다. 터미널 노드는 분류될 클래스 값을 갖는다.

XGBoost는 부스팅(gradient boosting) 기법을 사용한 앙상블 알고리즘이다. XGBoost는 다수의 회귀 트리로 구성된다. 기울기 하강(gradient descent) 최적화 알고리즘을 이용해 학습된다[17]. 임의 데이터를 초기화 후 이 데이터들을 특징으로 분류하는 함수를 구성시킨다. 학습 단계에서 이들 분류 함수 값은 트리의 터미널 노드로 표현된다. 그러므로 의사 결정 트리의 터미널 노드는 많은 분류 함수로 구성되며 노드가 많을수록 학습이 최적화되는 경향이 있다. XGBoost는 분류 또는 회귀 모델로 사용하고 있지만 트리 구성 시 사용된 특징의 중요도를 분석할 수 있어 분류 결과 해석에 이용될 수 있다.

Random Forest 알고리즘은 여러 개의 의사 결정 트리를 학습한 다음 각 트리의 예측 결과 중에서 가장 많은 분류 결과가 나타난 클래스로 예측한다[15]. 의사 결정 트리의 학습 시 학습 특징 집합을 무작위로 선택하는 배깅(bagging) 기법을 따른다. 그러므로 각 트리의 학습에 임의로 선택된 특징으로 구성된 부분 특징 데이터를 학습시킨다. 이 앙상블기법은 학습 데이터의 과적합이 기대되는 단일 의사 결정 트리의 단점을 보완할 수 있다. 실험 분석에서 Random Forest는 작은 훈련 데이터에 대해 분류 예측이 높지 않은 경향이 보고되었다.

3.5 모델 평가

Table 5는 악성코드 탐지에서 교차 행렬을 보여주며 정상 코드를 양성(+1), 악성코드를 음성(-1)으로 정의했다. TP(True Positive)는 정상코드를 올바르게 분류한 데이터 수, FN(False Negative)은 정상코드를 악성코드로 분류한 데이터 수, FP(False Positive)는 악성코드를 정상코드로 분류한 데이터 수, TN(True Negative)은 악성코드를 올바르게 분류한 데이터 수를 의미한다. 모델 평가의 성능 척도로서 분류 정확도 ACC(accuracy)와 오탐률 FPR(false positive rate)을 비교한다. 분류 정확도는 전체 데이터에서 정확히 분류된 데이터의 비율을 나타내며 모델의 일반적인 분류 성능을 측정할 수 있다(식 1). 오탐률은 정상코드로 분류한 데이터에서 실제 악성코드인 데이터의 비율이며 악성코드 탐지 모델에서 주로 사용되는 성능 척도이다(식 2).

사전 모델 평가로부터 학습 알고리즘의 파라미터를 결정했다. 의사 결정 트리 모델은 트리 깊이를 15로 하고 두 앙상블 모델의 트리 수는 50개, 트리의 깊이는 15, 특징 반영 비율은 50.0%로 설정했다. 평가 방법은 5-식 교차 평가를 사용했다.

$$\mathrm{ACC} = \frac{TN + TP}{TN + FP + FN + TP} \quad (1)$$

$$\mathrm{FPR} = \frac{FP}{FP + TP} \quad (2)$$

Table 5. Confusion matrix for malware detection

		Predicted	
		Benign	Malware
Actual	Benign	TP	FN
	Malware	FP	TN

Ⅳ. 실험 및 토의

준비된 악성코드 데이터로부터 API, API-DLL, DLL-CM 특징을 추출했다. Table 6은 특징의 차원 비교이다. API-DLL, DLL-CM, API 순서대로 차원이 크게 나타났다. API의 특징 선택률은 약 0.44%이며, API-DLL의 특징 선택률은 약 8.22%이다. 1,276 종 DLL의 2-gram과 16 종 DLL으로 구성된 DLL-CM의 차원을 비교했을 때 특징 선택률은 0.02%의 매우 낮은 차원 선택률로 분석된다.

의사 결정 트리(DT), XGBoost(XGB),

Table 6. Dimension of selected features

Feature	Dimension	Rate(%)
API	218	0.443
API-DLL	4,040	8.218
DLL-CM	256	0.016

Random forest(RF) 학습 알고리즘과 추출된 특징에 따라 분류 정확도와 학습 시간을 비교했다. Table 7은 정상과 악성을 구분하는 이진 분류 문제에 대한 정확도 성능 비교이며 Table 8은 악성코드 패밀리를 구분하는 다중 분류 문제의 정확도 성능 비교이다. 두 분류 문제에서 특징 별 분류 성능은 약 0.90 이상으로 비슷하게 나타났다. 특징에 따른 정확도는 API-DLL, API, DLL-CM 특징 순으로 우수했으며 API-DLL 특징이 다른 특징보다 정확도가 가장 높았지만 차원이 높아 학습 시간도 가장 오래 소요 되었다.

Table 9와 Table 10은 정상코드를 포함하는 악성코드 패밀리를 같이 분류하는 문제에 대한 ACC와 FPR 분류 성능이다. 5개의 악성코드 패밀리를 분류하는 실험 결과보다 정확도가 약 4.0% 낮게 분석되었다. 오탐률의 경우 모든 특징에서 0.05 이하의 성능을 보였으며 API-DLL 특징으로 학습된 XGBoost 모델이 성능이 가장 우수했다(Table 10).

Tpable 7. 2-class classification performance

Algorithm		API	API-DLL	DLL-CM	Avg.
DT	Train	0.970	0.973	0.965	0.969
	Test	0.917	0.918	0.897	0.911
	Time	0.642	6.762	0.500	2.635
XGB	Train	0.973	0.979	0.959	0.970
	Test	0.935	0.941	0.912	0.929
	Time	6.233	25.287	6.373	12.631
RF	Train	0.973	0.975	0.968	0.972
	Test	0.936	0.938	0.915	0.930
	Time	9.815	101.888	6.749	39.484

Table 8. 5-class classification performance

Algorithm		API	API-DLL	DLL-CM	Avg.
DT	Train	0.964	0.970	0.961	0.965
	Test	0.907	0.916	0.899	0.907
	Time	0.357	3.206	0.229	1.264
XGB	Train	0.964	0.976	0.957	0.966
	Test	0.923	0.935	0.909	0.922
	Time	16.880	68.607	18.176	34.554
RF	Train	0.965	0.973	0.961	0.966
	Test	0.922	0.932	0.910	0.921
	Time	5.022	44.284	3.298	17.535

Table 9. ACC performance of 6-class classification

Algorithm		API	API-DLL	DLL-CM	Avg.
DT	Train	0.948	0.952	0.937	0.946
	Test	0.867	0.867	0.845	0.860
	Time	0.570	5.763	0.435	2.256
XGB	Train	0.950	0.965	0.933	0.949
	Test	0.897	0.908	0.871	0.892
	Time	26.748	107.742	28.579	54.356
RF	Train	0.950	0.960	0.941	0.950
	Test	0.893	0.903	0.870	0.889
	Time	8.702	85.230	6.280	33.404

Table 10. FPR performance of 6-class classification

Algorithm		API	API-DLL	DLL-CM	Avg.
DT	Train	0.011	0.010	0.016	0.012
	Test	0.045	0.044	0.049	0.046
XGB	Train	0.008	0.009	0.015	0.011
	Test	0.033	0.030	0.043	0.035
RF	Train	0.008	0.007	0.010	0.008
	Test	0.038	0.031	0.043	0.037

Fig 4는 API-DLL 특징으로 학습된 XGBoost 모델의 분류 성능에 따른 결과를 교차 행렬로 비교한다. 분류 성능을 분석했을 때, Backdoor 클래스에서 53.15%, Virus 클래스에서 46.45%의 오분류가 있었다. Backdoor의 45.9% 데이터는 Trojan으로 오분류 되었고 Virus의 30.3%는 Benign, 11.0%는 Trojan으로 오분류 되었다. 구성된 학습 데이터는 클래스 불균형 문제가 있으며 Backdoor

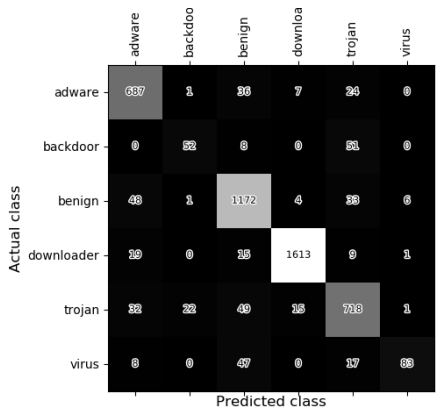

Fig. 4. Confusion matrix of XGBoost model

와 Virus 클래스는 데이터 수가 적은 마이너 클래스이기 때문에 학습 과정에 영향이 적어 성능이 낮은 것으로 해석된다. 또한, 행위를 표현하는 API 기반 특징에서 Trojan 악성코드가 수행하는 다양한 행위가 악성코드 패밀리 간 유사도를 높여 오분류가 발생한 것으로 분석된다.

Ⅴ. 결 론

본 논문에서는 악성코드와 정상코드 분석을 통해서 빈번히 사용되는 DLL과 API 목록을 선정해 API, API-DLL, DLL-CM 특징의 차원을 축소했다. 준비된 특징 벡터로 의사 결정 트리 모델과 앙상블 모델을 학습시키고 일반화 성능을 비교했다.

정상코드와 악성코드를 분류하는 이진 분류 실험과 악성코드 패밀리를 분류하는 다중 분류 실험을 진행했다. 두 실험에서 모두 0.90 이상의 분류 정확도를 보였다. 훈련 특징 별 분류 성능을 비교했을 때 API-DLL 특징이 분류 정확도가 가장 우수했다. 하지만, 차원이 높아 학습 시간이 오래 소요되었고 고차원으로 인한 과적합 분석이 요구된다.

악성코드 패밀리와 정상코드를 같이 분류하는 문제에서 모델과 특징에 따른 분류 정확도와 오탐률을 비교했다. 분류 결과에 대한 교차 행렬 분석을 통해 클래스 불균형 문제를 확인했다. 성능을 개선시키기 위해서 패밀리 별 데이터 확보와 샘플링 과정이 필요하다.

API 특징 기반 탐지 모델은 API 목록을 사전에 정의해야 한다. 윈도우 운영체제가 업데이트 되거나 지원하는 API 함수 이름이 변경될 경우 프로그램에서 사용되는 API도 변경된다. 변경된 API 함수는 필터링 목록에 없기 때문에 구성되는 특징 값이 클래스에 상관없이 희소해진다. 따라서 탐지 모델을 장기간 사용하기에 부적절하며 필터링 목록과 학습 모델의 파라미터가 지속적으로 갱신되어야 한다.

References

[1] Symantec, "Symantec internet security threat report," ISTR-23-2018, Symantec, 2018.

[2] P. Vinod, R. Jaipur, V. Laxmi and M. Gaur, "Survey on malware detection methods," Proceedings of the 3rd Hacker's workshop on computer and internet security, pp. 74-79, Mar. 2009.

[3] N. Idika, and A.P. Mathur, "A survey of malware detection techniques," Purdue University, Feb. 2007.

[4] A. Patcha and J.M Park, "An overview of anomaly detection techniques : Existing solutions and latest technological trends," Computer networks, vol. 51, no. 12, pp. 3448-3470, Aug. 2007.

[5] K. Rieck, T. Holz, C. Willems, P. Dussel and P. Laskov, "Learning and classification of malware behavior," Proceeding of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 108-125, 2008.

[6] B. Sun, Q. Li, Y. Quo, Q. Wen, X. Lin and W. Liu, "Malware family classification method based on static feature extraction," Proceeding of the 3rd IEEE International Conference on Computer and Communications(ICCC), pp. 507-513, Dec. 2017.

[7] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov and G. Giacinto, "Novel feature extraction, selection and fusion for effective malware family classification," Proceedings of the 6th ACM conference on data and application security and privacy, pp. 183-194, Mar. 2016.

[8] R. Veeramani, and N. Rai. "Windows API based malware detection and framework analysis," Proceeding of the International conference on networks and cyber security, Vol. 25. Jan. 2012.

[9] M. Sikorski and A. Honig, Practical Malware Analysis: the hands-on guide to dissecting malicious

software, No Starch Press, Feb. 2012.
〔10〕 Avira, “avira antivirus for windows” https://www.avira.com/en/free-antivirus-windows, Jan. 2019.
〔11〕 C. Willems, T. Holz, F. Freiling, “Toward Automated Dynamic Malware Analysis Using CWSandbox,” IEEE Security and Privacy, vol. 5, no. 2, pp. 32-39, Apr. 2007.
〔12〕 G. Hunt, D. Brubacker, “Detours: Binary interception of Win32 functions,” Proceedings of the 3rd USENIX Windows NT Symposium, pp. 135 - 143, Jul. 1999.
〔13〕 VirusShare, “virus share” https://virusshare.com/, Jan. 2019.
〔14〕 Kaspersky, “kaspersky lab” http://www.kaspersky.com/, Jan. 2019.
〔15〕 L. Breiman, “Random forests,” Machine learning, vol. 45, no. 1, pp. 5-32, Oct. 2001.
〔16〕 R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov and M. Ahmadi, ″Microsoft Malware Classification Challenge,″ arXiv preprint arXiv:1802.10135, Feb. 2018.
〔17〕 T. Chen, and C. Guestrin, ″XGBoost: A scalable tree boosting system,″ Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, pp. 785-794, Aug. 2016.
〔18〕 C. Eagle, The IDA pro book, 2md Ed., No Starch Press, Jul. 2011. Malwares.com, “malware.com” https://malwares.com, Jan. 2019.
〔19〕 Radare2, “radare2” https://rada.re/r/, Jan. 2019.
〔20〕 Windows API Index, “Windows API Index” https://docs.microsoft.com/en-us/windows/desktop/apiindex/windows-api-list, Jan. 2019.

〈저자소개〉

이 현 종 (Hyunjong lee) 정회원
2017년 2월: 단국대학교 컴퓨터과학과 졸업
2019년 2월: 단국대학교 소프트웨어학과 석사
2019년 3월~현재: 케이사인 보안기술연구소 주임 연구원
〈관심분야〉 기계학습, 병렬처리, 영상처리

어 성 율 (Seongyul Euh) 정회원
1997년 2월: 아주대학교 컴퓨터공학과 졸업
1999년 2월: 아주대학교 컴퓨터공학과 석사
2018년 3월~현재: 단국대학교 소프트웨어학과 박사과정
〈관심분야〉 기계학습, 병렬처리, 위협 인텔리전스

황 두 성 (Doosung Hwang) 정회원
1986년 2월: 충남대학교 계산통계학과 졸업
1990년 2월: 단국대학교 전자계산학과 석사
2003년 2월: Wayne State Univ. 컴퓨터과학과 박사
2003년 3월~현재: 단국대학교 소프트웨어학과 교수
〈관심분야〉 기계학습, 병렬처리, 영상처리