논문 18-43-04-16 The Journal of Korean Institute of Communications and Information Sciences '18-04 Vol.43 No.04
https://doi.org/10.7840/kics.2018.43.4.730

DLL/API 통계적 분석을 통한 Feature 추출 및 Machine Learning 기반 악성코드 탐지 기법

하 지 희◆, 김 수 정*, 이 태 진°

Feature Extraction using DLL/API Statistical Analysis and Malware Detection based on Machine Learning

Ji-hee Ha◆, Su-jeong Kim*, Tae-jin Lee°

요 약

최근에 들어 랜섬웨어를 비롯해 다양한 형태의 신종/변종 악성코드가 하루에도 수십만 개 이상 등장하고 있으며, 직접적인 금전탈취 목적의 악성코드가 갈수록 증가하고 있는 추세이다. 이와 같이 대량으로 출현하는 악성코드를 적은 자원을 소모하면서 빠르게 처리하기 위해서는 정적기반 분석기법이 필수적으로 요구된다. 본 논문에서는 정적 분석 기법 중 Imported DLL과 API를 이용해 신속하고 Light한 Feature로 구성해 악성코드를 탐지하고자 한다. 8만여개 파일의 Imported DLL/API를 전수 조사하여 악성코드와 정상파일간의 출현 비율에 따른 경향성을 파악하고 Machine Learning결과를 비교분석해 Feature 선별 정책을 검증하여 다른 악성코드분석과 연계할 가능성을 제공하고 DLL/API를 기반으로 악성코드 분석의 정확도를 향상시키는 데 기여하리라 예상한다. 실험을 통해 산출된 주요한 DLL/API 정보들은 악성코드 변종탐지와 악성코드 그룹분류, 악성코드 유형 분류 등에 광범위하게 사용할 수 있으며 여러 가지 악성 분석 연구에 공통의 기반으로 활용가능하다.

Key Words : Malware Detection, Static Analysis, DLL, API, Deep Neural Network

ABSTRACT

In recent years, more than hundreds of thousands of new and variant malicious codes have appeared in various forms, including Ransomware, Malicious code for direct money takeover is increasing. Static based analysis techniques are required to dispose of malicious codes rapidly while consuming a small number of resources. In this thesis, we want to detect malicious code by composing a quick and light-like feature using the Imported DLL and API among static analysis techniques. By Whole investigation through the Imported DLL / API of more than 80,000 files,Identifying the Trends of Malicious Codes and Normal files, It provides the possibility to compare the machine learning results to verify the Feature Selection Policy and to link it other malicious code analysis, It is expected to contribute to improving the accuracy of malicious code analysis based on the DLL/API. The main DLL/API information produced through the experiment can be widely used for malicious code variant detection, malicious code group classification, malicious code type classification, and can be used as a common base for various malicious analysis studies.

※ 본 연구는 2017년도 호서대학교의 재원으로 학술연구비 지원을 받아 수행되었습니다. (2017-0053)
◆ First Author : (ORCID:0000-0003-0159-3912)Hoseo University Department of Information Security, hjh1590@gmail.com, 정회원
° Corresponding Author :(ORCID:0000-0003-3078-3459)Hoseo University Department of Information Security, kinjecs0@gmail.com, 정회원
* (ORCID:0000-0003-0471-8995)Hoseo University Department of Information Security, sjkim395@gmail.com
논문번호 : KICS2018-02-043, Received February 26, 2018; Revised April 3, 2018; Accepted April 4, 2018

Ⅰ. 서 론

보안업체 카스퍼스키 랩(Kaspersky Lab)에 따르면 2017년 기준 하루 평균 36만개의 신규 악성코드가 발생되는 것으로 나타났다. 2017년 발생한 대표적인 악성코드 유형을 살펴보면 데이터의 몸값을 요구하는 랜섬웨어와 금융정보를 타깃으로 한 정보유출 악성코드, 이외에도 다운로더, 드롭퍼, 백도어 등 다양한 악성코드 유형이 존재한다. 과거의 악성코드는 주로 정보유출에 목적을 두었지만, 최근에는 직접적인 금전탈취 목적의 악성코드가 갈수록 증가하고 있는 추세이며, 악성코드 공격 위협은 PC와 스마트폰, 태블릿PC, 노트북과 같은 다양한 기기를 감염시킬 수 있다. 따라서 이러한 공격에 대응하기 위한 방법이 필요하다.

악성코드를 분석하는 방법 중 하나인 정적 분석은 파일을 실행하지 않고도 파일이 가지고 있는 구조적인 정보만을 가지고 악의적인 여부를 판단하는 방식이다. 그러나 이러한 방식은 악성코드에 패킹이나 난독화와 같은 기법을 사용할 경우 악성여부를 판단하기 위한 구조적인 정보를 분석하기에는 어려움이 따르기 때문에 취약점이 존재하지만, 다른 분석방법은 시간이 많이 소요되는 작업이므로 본 논문에서는 PE 구조에서 악성행위와 관련된 주요부분의 특징인 DLL과 API를 이용해 악성코드 탐지에 신속하고 Light한 Feature로 구성한다.

본 논문에서는 악성코드가 윈도우에서 실행되기 위해 필요한 정보인 Imported DLL/API를 이용하여 정상파일과 악성코드간의 출현 비율과, 출현 빈도 차를 비교분석해 악성코드 탐지에 유의미한 Feature인지 Machine Learning 기술을 이용해 검증하여 DLL/API를 기반으로 악성코드를 탐지하는 방법을 제안하고자 한다.

Ⅱ. 본 문

2.1 관련연구

악성코드를 분석하는 방법에 있어 크게 두 가지로 나눌 수 있는데, 동적 분석은 실제 또는 가상 환경에서 파일을 실행시켜 나타나는 시스템의 변화를 분석하는 방식이다. 프로세스, 파일시스템, 레지스트리, 네트워크의 변화를 실시간으로 확인할 수 있지만 분석시간이 많이 소요되고 분석가가 직접 판단하는 것이라 놓치는 항목이 존재할 수 있다는 한계점이 있다[1]. 정적 분석은 악성코드를 실행하지 않고 바이너리 파일안의 동작, 원리, 구조적 정보를 이용해 파일의 악의적인 여부를 판단하는 방식이다. 그렇기 때문에 비교적 신속하게 분석이 가능하나, 악성코드 제작자가 탐지를 우회하기 위해서 패커를 통해 압축을 하거나 난독화, 암호화 기법을 적용한 경우, 내부의 구조적 정보를 확인하기 위해서는 언패킹과 복호화를 적용해야 분석이 가능해 진다[2].

악성코드 행위와 관련하여 악성코드 분류와 탐지를 위한 다양한 연구가 진행되고 있다. Hiran V. Nath는 악성코드와 정상파일에서 사용하는 API를 추출하여 해당 API가 악성코드에서 발생하는 확률에 따라 가중치를 산정하여 악성코드를 분류하는 연구를 수행하였다[3]. Wu Liu는 악성행위를 실행하기 위해서 다양한 API를 호출하는 특징을 이용하여 API 시퀀스에 따라 악성행위 그룹을 파일 Action, 프로세스 Action, 네트워크 Action, 레지스트리 Action으로 분류하고, 파일의 API 시퀀스에 따라 수식을 이용하여 행위별로 유사한 패턴이 존재할 때 위험도를 산출하여 악성코드를 탐지하고 분류를 수행하였다[4]. Y Zhong은 바이너리 파일 내부의 함수들의 특징을 이용하여 악성코드를 분류한 연구를 진행하였는데, API 호출 정보와, 사용한 String 문자열에 관한 정보, 함수의 basic block정보를 N-gram을 이용하여 빈도수를 추출하였고, 추출한 정보를 Support Vector Machines 알고리즘에 적용시켜 악성코드를 분류하였다[5]. Sung-tae. Yu는 악성코드와 정상파일에서 Native API 함수를 추출하고 해당 Native API가 악성코드에서 발생하는 확률에 따라서 F-measure 실험을 통해 가중치의 합을 결정하고, 최종적으로 가중치를 이용하여 워드 클라우드에서 텍스트의 크기로 표현되는 기법을 이용해 악성코드를 탐지하였다[6]. 이밖에도 K. Rieck는 악성코드의 행위에 대하여 공유 패턴을 이용하여 학습기반으로 악성코드를 자동 분류하는 연구가 진행되었는데 다양한 악성코드 샘플로부터 행위를 추출하기 위해 동적 분석인 샌드박스 환경을 이용하여 모니터링 하였으며 Data Acquiring, Behavior Monitoring, Feature Extraction, Learning and Classification의 4가지 단계별로 실험을 진행하였다[7].

이러한 연구 동향들은 PE파일에서 추출 가능한 다양한 정보들을 대상으로 악성코드 분석에 활용하고 있음을 알 수 있다. 이 중에서도 Imported된 DLL 및 API는 악성코드 분석에 매우 중요한 정보를 제공하여 관련된 다양한 연구들이 많이 진행중이다. 그러나 연구를 진행할 때마다 수만개에 이르는 DLL/API 우선순위에 대한 분석이 이루어지지 않았다. 효율적인 연구들을 진행할 수 있도록 비교분석 연구가 필요로 하

다. 본 논문에서는 8만여개의 악성코드와 정상파일을 대상으로 DLL 및 API 전수 분석을 통해, 출현 비율과 빈도차이 값을 산출해 통계적으로 분석하고 Machine Learning 기술을 이용해 검증하였다. 다음 절에서는 자세한 제안모델을 제시한다.

2.2 제안 모델

PE파일이 윈도우에서 실행되기 위해 필요한 정보는 PE구조의 헤더와 섹션에 존재한다. 때문에 악성코드를 실행하지 않고도 PE구조에서 악성행위와 관련된 정보를 얻을 수 있으며, PE- Header내의 구조적 정보인 IAT(Import Address Table)를 이용해 어떤 DLL(Dynamic Link Library)이 로드되는지 파악할 수 있으며, 해당 DLL에서 어떠한 함수를 사용하는지 확인할 수 있다[8]. DLL은 Microsoft의 Window에서 구현된 동적 라이브러리로 악성코드가 실행이 되면 일반적으로 프로세스, 메모리, 레지스트리, 네트워크와 관련된 DLL을 Import한다. Imported된 DLL은 시스템에 영향을 끼치는 특징을 가지는 함수를 사용하게 되는데, 이 때 악성코드와 정상파일을 구분할 수 있는 프로그램의 특징으로서 Imported된 DLL과 사용되는 API 목록을 이용해 전체파일에 대한 출현 비율과 악성과 정상사이에서의 출현 빈도 차를 비교하여 악성코드를 탐지하는 방법을 제안하고자 한다. 본 논문에서 제안하는 모델은 Fig.1과 같다.

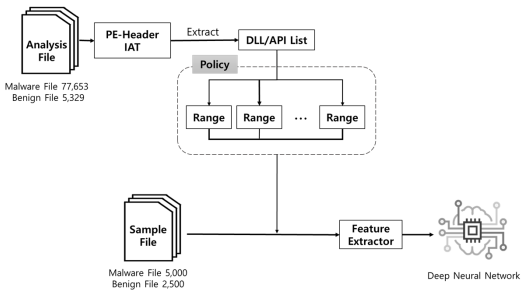

그림 1. 악성코드 탐지를 위한 DLL/API Feature 추출 및 검증 모델
Fig. 1. DLL/API Feature Extraction and Validation model for malware detection

2.2.1 DLL/API Feature 추출

악성코드를 실행하기 위해 필요한 DLL과 API를 추출하여 정보가 담긴 목록을 획득하기 위해서는 PE OPTIONAL Header내에 존재하는 구조적 정보인 IMAGE DIRECTORY ENTRY IMPORT를 이용해 Import된 DLL명과 API명 목록을 추출한다[9]. 본 논문에서는 추출된 1,622개의 DLL명으로 구성된 목록과 65,300개의 API명으로 구성된 목록을 이용해 분석대상 파일별로 해당 DLL/API의 존재유무에 따라 1과 0으로 체크하여 Feature의 값으로 사용하게 된다. 다음 Fig. 2는 추출된 DLL/API List의 일부이다.

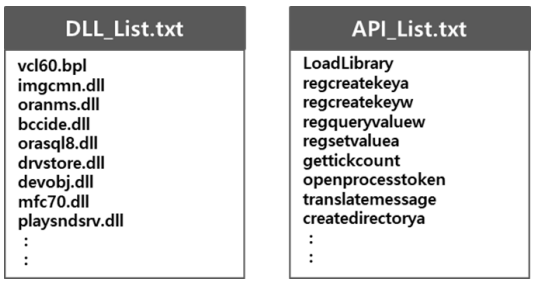

그림 2. Imported DLL/API 예시
Fig. 2. Imported DLL / API Example

그 중에서 악성코드와 정상파일 사이에 유의미한 차이를 나타내는 Feature를 선정하기 위해 전체파일에 대한 출현 비율 평균과 악성과 정상에서의 출현빈도차를 이용해 통계적 분석을 하였다.

2.2.2 Feature 선별정책

추출된 DLL 1,622개와 API 65,300개의 Feature들 중 유의미한 Feature를 선별하기 위해서 정상파일에서의 출현 비율과 악성코드에서 출현 비율을 가공하여 Range 정책을 선정하고자 한다. 정상과 악성에서 각각의 출현 비율은 수식 (1)과 수식(2)와 같이 계산할 수 있다.

$$\text{Rate of Occurrence in Normal File} = \text{Number of Occurrences in Normal Files} / \text{Number of Normal Files} \quad (1)$$

$$\text{Rate of Occurrence in Malware File} = \text{Number of Occurrences in Malware Files} / \text{Number of Malware Files} \quad (2)$$

수식(3)은 수식(1)과 (2)를 이용하여 정상과 악성사이의 출현 비율 평균 나타내며, 악성과 정상사이의 출현 빈도 차이를 나타내는 수식(4)는 100%에 가까울수록 악성의 비율이 높고 0%에 가까울수록 정상의 비율이 높으며, 50%라면 악성과 정상의 비율이 1:1이라는 것을 의미한다. 따라서 50에 가까울수록 악성과 정상에서의 차이가 없다.

$$X = (\text{Rate of Occurrence in Normal File} + \text{Rate of Occurrence in Malware File}) / 2 \quad (3)$$

Y = Rate of Occurrence in Malware File / (Rate of Occurrence in Normal File + Rate of Occurrence in Malware File) (4)

Fig.3과 Fig.4와 같이 정상과 악성사이의 출현 비율 평균을 X축으로 하고 악성과 정상사이에서의 출현 빈도 차이를 Y축으로 하여 악성여부를 탐지하기 위해 어느 축의 Range에 가중치를 더 크게 부여해야하는지 Range 정책을 결정한다.

이를 위해 X축과 Y축을 특정 비율마다 Feature로 선정하여 해당 Range별로 Machine Learning 결과를 이용해 비교한다. Range를 나눈 영역은 A부터 G까지 존재하며 A를 기준으로 B, C, D는 Y축, E, F, G는 X축에 대한 변화를 포함한다. Fig.5는 수식(3)에 따른 주요 영역 중 하나인 E영역의 DLL./API List 일부와 수식(4)에 따른 주요 영역 중 하나인 B영역의 DLL./API List 일부이다.

Fig.3는 DLL을 기반으로 한 Feature range를 나타내고, Fig.4은 API을 기반으로 한 Feature range 나타낸다. Fig.3는 DLL 기준으로 보았을 때 악성과 정상

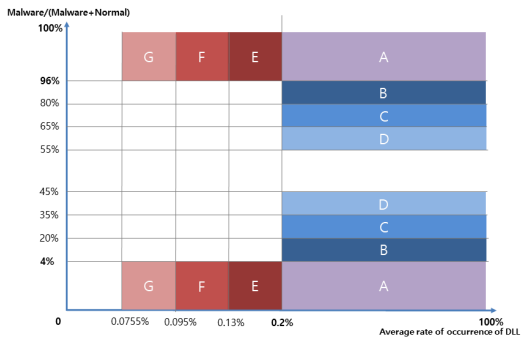

그림 3. DLL을 기반으로 한 Feature range 영역
Fig. 3. Feature coverage area based on the DLL

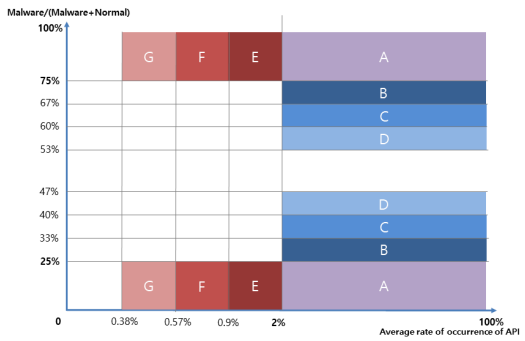

그림 4. API을 기반으로 한 Feature range 영역
Fig. 4. Feature coverage area based on the API

파일간의 출현 비율 평균(X축)이 0.2%이상이고, 악성과 정상사이 출현 빈도 차이(Y축)가 96%이상과 4% 이하의 Range를 가진 Feature를 Range 정책 결정을 위한 실험에서 기준점 A로 정했다. Fig.4는 Fig.3과 같은 방식의 정책을 사용했으며 해당하는 Range에는 차이가 있다. Range의 적정선을 파악하기 위해서 A를 기준으로 하여 X축을 고정한 B, C, D영역과 Y축을 고정한 E, F, G영역의 실험을 각각 진행한다. B, C, D 영역은 정상파일 샘플과 악성코드 샘플 사이에서 DLL/API의 빈도차이에 변화를 주었으며 E, F, G영역은 DLL/API의 출현 비율 값에 변화를 주었다. Feature 개수가 ML결과에 영향을 미치게 됨으로 Range를 나눌 때, A영역의 Feature 개수와 다른 영역의 개수가 일정하게 유지되도록 B, C, D, E, F, G로 영역을 나눈다.

Fig. 6는 2번째 실험인 DNN모델 성능비교 실험에서 사용하는 DLL, API의 H와 I에 대한 Feature Range 영역을 나타낸다. Table 1은 Fig. 3과 Fig.4, Fig.6에서 나타내는 DLL/API 영역별 X축, Y축 기준이고, 각 영역별 Feature 개수는 Table 2와 같다.

Range 정책 결정 실험은 A를 기준으로 B, C, D와 E, F, G 영역의 ML 실험결과를 비교하여 악성코드 탐지에 유의미한 Feature로 사용가능한 Threshold를 선정한다. Machine Learning을 이용하여 결과를 산출할 때 절대적인 Feature 개수를 증가시키면 (Overfitting이 일어나지 않은 선에서) 결과가 더 개선되는 경향이 존재하지만 무한적으로 개선되지는 않는다. 따라서 제안한 모델의 도메인 내에서 최적의 결과를 산출하기 위한 Range 정책 결정 실험을 통한 최적

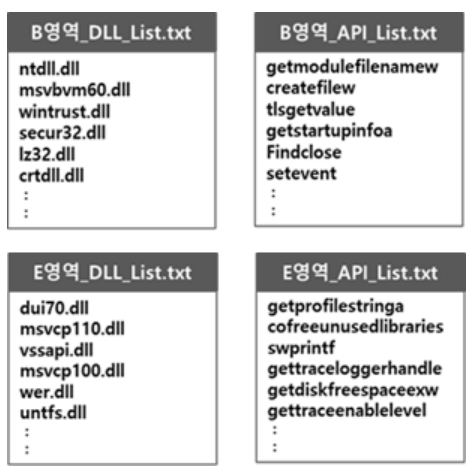

그림 5. E Range, B Range DLL/API 목록
Fig. 5. E Range, B Range DLL/API List

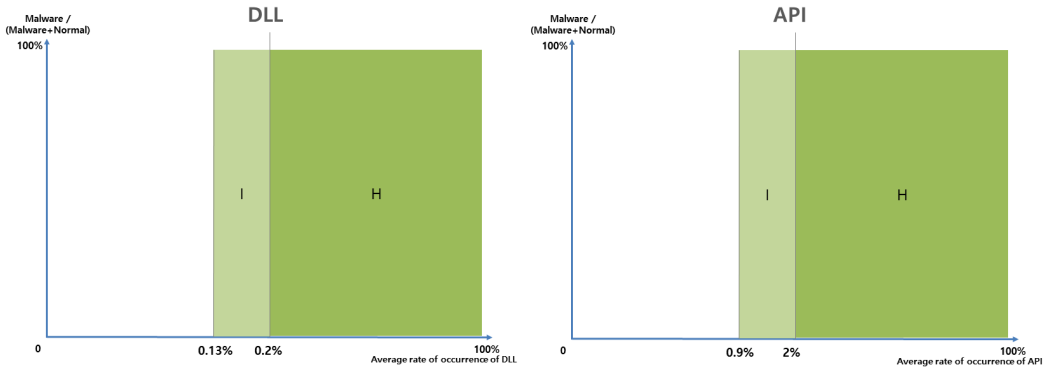

그림 6. DLL,API에서의 DNN모델 성능비교 실험을 위한 Feature range 영역
Fig. 6. Feature range for DNN model performance comparison experiment in DLL and API

의 Range 선정을 제안한다.

DNN모델 성능 비교 실험은 Range에 따라 어떤 DNN모델을 이용해야 적절한 실험이 가능한지를 보여준다. DNN모델의 구축을 위해 Google의 Tensorflow를 활용하였다. Range에 따른 Hidden Layer와 Node의 적정 개수를 파악하여 실험에 사용되는 수준의 Data에서 적합한 DNN모델을 선정을 제안한다[10,11].

표 1. 영역별 X축, Y축 기준
Table 1. Range Based on X axis and Y axis

Category	Range	X-axis(%)	Y-axis(%)
DLL	A	0.2<=X<100	0<=Y<4, 96<=Y<100
	B	0.2<=X<100	4<=Y<20, 80<=Y<96
	C	0.2<=X<100	20<=Y<35, 65<=Y<80
	D	0.2<=X<100	35<=Y<45, 55<=Y<65
	E	0.13<=X<0.2	0<=Y<4, 96<=Y<100
	F	0.095<=X<0.13	0<=Y<4, 96<=Y<100
	G	0.0755<=X<0.095	0<=Y<4, 96<=Y<100
	H	0.2<=X<100	0<=Y<100
API	I	0.13<=X<0.2	0<=Y<100
	A	2<=X<100	0<=Y<25, 75<=Y<100
	B	2<=X<100	25<=Y<33, 67<=Y<75
	C	2<=X<100	33<=Y<40, 60<=Y<67
	D	2<=X<100	40<=Y<47, 53<=Y<60
	E	0.9<=X<2	0<=Y<25, 75<=Y<100
	F	0.57<=X<0.9	0 < = Y < 2 5 , 75<=Y<100
	G	0.38<=X<0.57	0 < = Y < 2 5 , 75<=Y<100
	H	2<=X<100	0<=Y<100
	I	0.9<=X<2	0<=Y<100

표 2. 영역별 Feature 개수
Table 2. Number of Feature by range

	API	DLL
A	241	30
B	207	20
C	219	21
D	183	21
E	200	20
F	209	21
G	205	21
H	938	103
I	1450	130

Ⅲ. 실 험

본 논문에서 제안하고 있는 PE파일의 DLL, API의 사용여부를 Feature로 활용한 악성탐지 기법의 성능 검증을 위해 PE파일에서 사용되고 있는 DLL, API의 비율을 조사하여 Range를 결정하는 정책과 Feature의 성능을 확인하기 위한 Machine Learning의 DNN(Deep Neural Network)모델 별 결과에 대해 비교하는 실험을 통해 적절한 Range 정책과 Feature별로 적합한 DNN모델을 선정하고 결과에 대해 분석한다.

3.1 DLL, API Range 정책 선정 실험

DLL, API의 Feature를 결정하기 위해 적정한 Range 판별 및 Feature 선별을 위한 실험을 수행하였다. 적정한 Feature를 선별을 위한 실험이기 때문에 사용되는 샘플은 별도의 Test set 없이 Training set 악성샘플 5,000개, 정상 2,500개로 선정하였다. Feature의 Range를 지정함에 있어서 Y축의 변화가 Machine Learning 결과에 미치는 영향을 확인하기 위해 B, C, D영역의 각각의 결과를 비교하고, B, B+C, B+C+D와 같이 영역이 증가함에 따른 결과의 변화를 확인하였다. X축의 변화를 확인하는 E, F, G영역도 각각의 결과를 비교하고 영역이 커질 때 정확도의 변화를 측정하였다.

Table 3은 A, B, C, D, E, F, G의 각 영역별로 Feature를 파싱하여 Machine Learning을 수행한 결과를 정리한 것이다. A영역의 정확도와 X축을 고정하고 Y축에 변화를 준 B, C, D영역의 정확도를 비교했을 때, DLL, API의 악성코드와 정상파일 사이의 빈도 차이가 작아지더라도 결과에 부정적인 영향을 끼치지 않는다는 것을 확인할 수 있다. Y축을 고정하고 DLL, API의 출현 비율을 나타내는 X축에 변화를 준 E, F, G영역을 A영역과 정확도를 비교했을 때, 출현 비율이 낮아지면 결과에 부정적인 영향을 미치게 되므로 출현 비율이 높아야만 하는 것을 확인할 수 있다.

Fig.7은 Table 3 실험에 대한 ROC Curve를 그린 그림으로 그래프 아래의 면적을 구한 AUC값이 클수

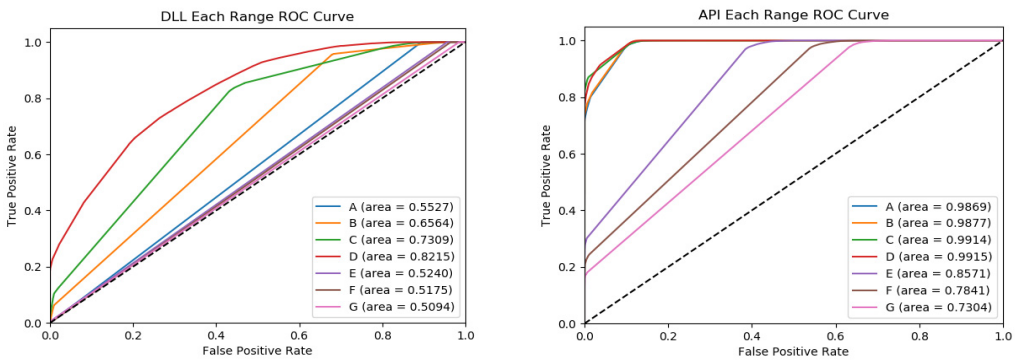

그림 7. A, B, C, D, E, F, G 각 영역별 ROC Curve 및 AUC
Fig. 7. ROC Curve and AUC for each range of A, B, C, D, E, F and G

표 3. A, B, C, D, E, F, G 영역 조합별 Machine Learning 결과
Table 3. Machine learning results for combinations of A, B, C, D, E, F, and G range

Category	Range	True Positive Rate(%)	False Positive Rate(%)	Accuracy(%)
DLL	A	99.90	89.36	70.15
	B	95.70	68.16	74.41
	C	85.40	46.80	74.67
	D	91.56	49.04	78.03
	E	99.92	95.56	68.09
	F	99.94	96.44	67.81
	G	100.00	98.52	67.16
API	A	98.56	10.64	95.49
	B	99.82	11.96	95.89
	C	98.68	10.72	95.55
	D	99.42	11.20	95.88
	E	98.68	41.24	85.37
	F	98.54	55.80	80.43
	G	98.54	64.28	77.60

록 좋은 결과라고 볼 수 있다. DLL에서는 D영역, API에서는 B, C, D영역이 눈에 띄게 좋은 것을 확인할 수 있다. Fig.7과 같은 ROC Curve는 sklearn 라이브러리의 roc_curve 함수와 matplotlib 라이브러리를 활용하여 그림으로 표현하였다[11].

A+B, A+B+C, A+B+C+D와 같은 조합을 이용하여 영역을 확장시켜 Feature의 개수를 늘린 결과는 Table 4에서 나타난다. DLL/API의 Range를 나누어 실험한 결과를 확인할 수 있었다. DLL과 API는 전체 Feature의 개수와 전체적인 정확도가 다르기 때문에 결과에서 차이를 보이는데 DLL의 경우 각 영역의 Feature개수가 API에 비해 적기 때문에 정확도의 변화 폭이 크게 되고, API의 경우 정확도 값이 95%라는 비교적 높은 수치에서 시작하여 Feature의 개수가 증가함에 따라 정확도가 차츰 증가하는 추세임을 확인할 수 있다. Y축이 전반적으로 정확도가 더 높고 영역이 커질 때 정확도 값의 증가폭 또한 X축보다 큰 것으로 보아 악성과 정상사이 출현 빈도 차이를 나타내는 Y축의 Range를 넓게 정하는 것이 악성 탐지에 좋은 것으로 판단된다.

Fig.9의 우측은 좌측 ROC Curve에 대해 x축을 0에서 0.14사이의 값, y축을 0.65에서 1.05사이 값의 좌표영역을 확대한 것이다. Fig.8을 보면 DLL에서 A부터 D까지의 영역을 조합한 Range에 해당하는 Feature로 학습한 결과가 가장 좋으며, Fig.9에서도 API에서 가장 좋은 결과가 DLL과 같은 Range인 A부터 D까지의 영역임을 통해 Fig.3과 Fig.4의 Y축인 악성과 정상사이 출현 빈도 차이를 나타내는 Range가 증가하여 Feature의 개수를 증가시킬 때 결과가 좋은 것을 확인할 수 있다.

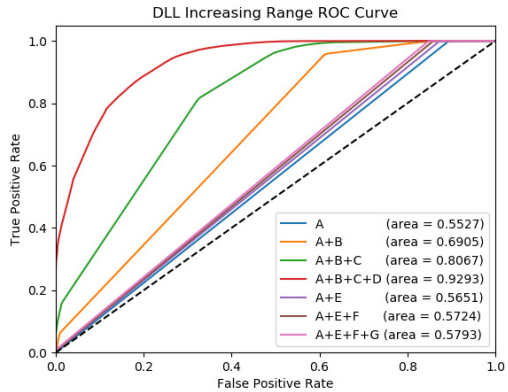

그림 8. DLL에 대한 A, B, C, D, E, F, G 영역 조합별 ROC Curve 및 AUC
Fig. 8. ROC Curve and AUC for each combination of A, B, C, D, E, F and G by DLL

3.2 DNN모델 성능 비교 실험

앞선 DLL, API Range 정책 선정 실험의 결과에서 보는바와 같이 악성과 정상 사이 출현 빈도 차이를 나타내는 Y축의 Range를 넓게 정하는 것이 악성 탐지에 효과적이라는 것을 확인했다. 이러한 경향에 따라서 Fig. 6이 나타내는 H와 I영역으로 DNN모델 성능 비교 실험을 위한 Range를 선정하였다. DNN모델의

표 4. A, B, C, D, E, F, G 영역 조합별 Machine Learning 결과
Table 4. Machine learning results for combinations of A, B, C, D, E, F, and G range

Category	Range	True Positive Rate(%)	False Positive Rate(%)	Accuracy(%)
DLL	A	99.90	89.36	70.15
	A+B	95.72	61.36	76.69
	A+B+C	96.34	50.04	80.88
	A+B+C+D	95.82	29.16	87.49
	A+E	99.84	87.24	70.81
	A+E+F	99.78	85.72	71.28
	A+E+F+G	99.78	84.68	71.63
API	A	98.56	10.64	95.49
	A+B	99.26	8.40	96.71
	A+B+C	99.88	9.08	96.89
	A+B+C+D	99.82	8.08	97.19
	A+E	99.52	11.68	95.79
	A+E+F	99.56	10.72	96.13
	A+E+F+G	98.94	9.08	96.27

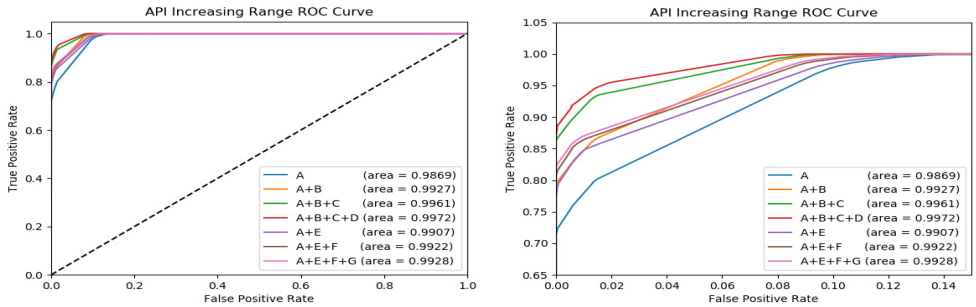

그림 9. API에 대한 A, B, C, D, E, F, G 영역 조합별 ROC Curve 및 AUC
Fig. 9. ROC Curve and AUC for each combination of A, B, C, D, E, F and G by API

성능비교 실험에서 Training set은 Range 판별 및 Feature 선별 실험과 동일한 set을 사용하였고, Training set과 다른 샘플로 구성된 악성코드 샘플 5000개, 정상 샘플 2500개인 Test set을 이용하여 H와 I영역을 대상으로 비교 실험을 수행하였다. Feature 개수는 DLL/API의 Range 실험 결과에 따라 성능 검증을 위한 실험에 사용될 샘플로 Table 2와 같이 Feature의 개수를 일정 수준 유지하였다. DNN모델은 Hidden Layer와 Node의 수를 다르게 하여 모델 별로 차이에 따른 실험을 진행하였다.

Table 5는 악성탐지 기법에 사용되는 DNN모델의 성능을 비교하기 위해 test set에 대해 여러 DNN 모델에서의 실험을 비교한 True Positive Rate, False Positive Rate, Accuracy에 대한 결과를 정리한 것이다. DLL과 API 각각 2가지 Range에 대해 총 6번의 실험을 진행하여 API에서 91%이상, DLL에서 86%이상의 정확도를 확보하였다.

X축이 A이고 Y축은 전체인 Range로 파싱한 Feature로 실험한 결과는 Table 5의 H Range와 같으며 Hidden Layer 6개와 Node 150개인 경우의 정확도가 Hidden Layer 4개에서 5개로, Node 50개

에서 100개로의 증가폭 보다 크게 증가한 것을 확인할 수 있다. 이에 따라 X축 영역을 A로 range하는 정책을 적용하여 파싱한 Feature가 다른 모델들보다 Hidden layer6, Node150인 DNN 모델에 적합하다고 판단된다. Table 5의 I Range는 X축이 A+E이고 Y축이 전체인 Range로 파싱한 Feature로 실험의 결과이다. Hidden layer 4개, Node 100개일 때의 정확도가 가장 높으므로 해당 DNN모델이 가장 적합하다고 판단된다. Hidden Layer 5개, Node 150개에서는 정확도가 감소하는 것과 같이 Hidden Layer와 Node의 개수가 늘어나는 것이 정확도가 항상 상승하는 것이 아님을 알 수 있다.

표 5. DNN 모델에 따른 Machine Learning 결과
Table 5. Machine learning results according to DNN model

Category	Range	DNN Model	True Positive Rate(%)	False Positive Rate(%)	Accuracy(%)
API	H	Hidden Layer4, Node50	95.00	15.60	91.47
		Hidden Layer5, Node100	94.50	14.24	91.59
		Hidden Layer6, Node150	95.08	13.48	92.23
	I	Hidden Layer4, Node50	94.86	15.12	91.53
		Hidden Layer5, Node100	95.02	13.84	92.07
		Hidden Layer6, Node150	94.88	14.56	91.73
DLL	H	Hidden Layer4, Node50	91.88	23.00	86.92
		Hidden Layer5, Node100	93.24	25.64	86.95
		Hidden Layer6, Node150	93.86	25.44	87.43
	I	Hidden Layer4, Node50	92.02	24.32	86.57
		Hidden Layer5, Node100	92.00	22.92	87.03
		Hidden Layer6, Node150	93.04	24.68	87.13

Ⅳ. 결 론

본 논문에서는 PE구조의 악성행위와 관련된 특징인 DLL과 API를 이용하여 악성코드를 탐지하는 방법을 제안하였다. 제안된 방법은 8만여개의 분석대상 파일에 대해 DLL/API를 전수 조사하여 목록을 추출하고, 목록에 기반한 DLL/API의 사용여부를 Feature로 활용하여 악성과 정상파일간의 출현 비율 평균을 X축으로, 출현 빈도차를 Y축으로 2차원 평면 형태로 선정하여 일정 비율마다 Range를 나누었다. 이를 통해 악성코드와 관련된 연구를 수행하는데 주요한 DLL과 API 목록을 도출하기 위해서, 악성여부 탐지 관점에서 어느 축의 Range에 가중치를 더 크게 부여해야하는지 DNN 모델의 실험 및 실험 결과를 이용해 분석하였다. 제안한 모델의 도메인 내에서 최적의 결과를 산출하기 위한 Range 정책 결정 실험을 통해 최적의 Range 선정을 제안하였고, 그 결과 악성과 정상사이 출현 빈도 차이를 나타내는 악성과 정상사이 출현 빈도 차이의 Range를 넓게 정하는 것이 악성 탐지에 효과적이라는 것을 확인했다. 두 번째 실험인 악성탐지 기법에서 사용되는 DNN 모델의 성능비교 실험을 통해 Hidden Layer와 Node의 개수를 증가시키는 것이 항상 정확도를 상승시킬 수 없음을 확인했으며, 악성과 정상 사이의 출현 빈도 차에 개의치 않고 출현 비율 평균의 상위 영역에 해당하는 Feature를 사용하여 실험을 진행하였다. 결과적으로 DLL에서 100여개, API에서 1000여개 정도의 Feature 개수를 이용할 경우 Hidden Layer가 6개, Node가 150개인 DNN모델을 사용하는 것이 적합하다는 것을 확인하였다.

정적 분석 시 패킹이나 난독화와 같은 기법을 사용할 경우 악성여부를 판단하기 위한 구조적인 정보를 분석하기에 어려움이 따르는 한계점이 존재한다. 그럼에도 불구하고 8만여개 파일의 전수분석을 통해 추출된 DLL/API 정보들을 이용하여 악성코드 탐지에 신속하고 Light한 Feature를 선정하고, 이를 이용하여 정적 분석만으로 API는 91%이상, DLL에서 86%이상의 정확도가 나타남을 확인하였다. 추출된 DLL/API 정보들을 조합해서 활용한다면 악성여부 분석에서 더 좋은 정확도 결과도 기대될 것으로 보인다. 뿐만 아니라 이 정보들은 악성코드 변종탐지와 악성코드 그룹분류, 악성코드 유형 분류 등에 광범위하게 사용할 수 있으며 여러 가지 악성 분석 연구에 공통의 기반으로 활용가능하다. 위의 실험을 통해 산출된 주요한 DLL/API 목록을 이용하여, 향후 DLL과 API의 존재유무와 악성코드 탐지 뿐 아니라 악성코드의 행위 특성들의 DLL/API 특징을 활용하여 행위의 API 호출 시퀀스 분석을 통해 패턴추출 연구를 진행할 예정이다.

References

[1] S.-B. Park, M.-S. Kim, and B.-N. Noh, "Detection method using common features of malware variants generated by automated tools," *J. Korean Inst. Inf. Technol.*, vol. 10, no. 9, pp. 67-75, Sept. 2012.

[2] J. Park, *Deep learning based malicious code detection using API features,* M. S. Thesis, Konkuk University, 2017.

[3] H. V. Nath and B. M. Mehtre, "Static malware analysis using machine learning methods," *Int. Conf. Secur. Comput. Netw. and Distrib. Syst.*, pp. 440-450, Trivandrum, India, Mar. 2014.

[4] W. Liu, P. Ren, K. Liu, and H. Duan, "Behavior-based malware analysis and detection," *IWCDM*, pp. 39-42, Nanjing, China, Sept. 2011.

[5] Y. Zhong, H. Yamaki, and H. Takakura, "A malware classification method based on similarity of function structure," *2012 12th Int. Symp. IEEE/IPSJ Appl. Internet (SAINT)*, pp 256-261, Izmir, Turkey, Jul. 2012.

[6] S. Yu and S. Oh, "Malware analysis mechanism using the word cloud based on API statistics," *J. Korea Academia-Ind. Cooperation Soc.*, vol. 16, no. 10, pp 7211-7218, Oct. 2015.

[7] K. Rieck, T. Holz, C. Willems, and P. Laskov, "Learning and classification of malware behavior," in *Int. Conf. Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 108-125, Paris, France, Jul. 2008.

[8] C.-W. Woo and K.-H. Ha, "A development of malware detection tool based on signature patterns," *J. Korea Soc. Comput. and Inf.*, vol. 10, no. 6, pp. 127-35, Dec. 2005.

[9] H.-U. Hwang and J.-H. Chae, "Memory injection technique and injected DLL analysis technique in windows environment," *J. Inf.*

and Secur., vol. 6, no. 3, pp. 59-67, Sept. 2006.

[10] J. Torres, *First Contact with Tensorflow*, Hanbit Media, 2016.

[11] A. C. Muller and S. Guido, *Introduction to Machine Learning with Python*, Hanbit Media, 2017.

김 수 정 (Su-jeong Kim)

2018년 2월 : 호서대학교 정보보호학과 졸업

2018년 3월~현재 : 호서대학교 정보보호학과 석사

<관심분야> 정보보호, 악성코드 분석, 머신러닝

하 지 희 (Ji-hee Ha)

2018년 2월 : 호서대학교 정보보호학과 졸업

2018년 3월~현재 : 호서대학교 정보보호학과 석사

<관심분야> 악성코드 분석, 정보보호

이 태 진 (Tae-jin Lee)

2003년 1월~2017년 2월 : 한국인터넷진흥원 팀장

2017년 3월~현재 : 호서대학교 정보보호학과

<관심분야> 시스템 보안, 악성코드 분석, 침해사고 대응