Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.30, NO.4, Aug. 2020 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2020.30.4.593

Stacked Autoencoder 기반 악성코드 Feature 정제 기술 연구*

김 홍 비,[1†] 이 태 진[2‡]
[1,2]호서대학교(대학원생, 교수)

Stacked Autoencoder Based Malware Feature Refinement Technology Research*

Hong-bi Kim,[1†] Tae-jin Lee[2‡]
[1,2]Hoseo University(Graduate student, Professor)

요 약

네트워크의 발전에 따라 악성코드 생성도구가 유포되는 등으로 인해 악성코드의 출현이 기하급수적으로 증가하였으나 기존의 악성코드 탐지 방법을 통한 대응에는 한계가 존재한다. 이러한 상황에 따라 머신러닝 기반의 악성 코드 탐지 방법이 발전하는 추세이며, 본 논문에서는 머신러닝 기반의 악성 코드 탐지를 위해 PE 헤더에서 데이터의 feature를 추출한 후 이를 이용하여 autoencoder를 통해 악성코드를 더 잘 나타내는 feature 및 feature importance를 추출하는 방법에 대한 연구를 진행한다. 본 논문은 악성코드 분석에서 범용적으로 사용되는 PE 파일에서 확인 가능한 DLL/API 등의 정보로 구성된 549개의 feature를 추출하였고 머신러닝의 악성코드 탐지 성능 향상을 위해 추출된 feature를 이용하여 autoencoder를 통해 데이터를 압축적으로 저장함으로써 데이터의 feature를 효과적으로 추출해 우수한 정확도 제공 및 처리 시간을 2배 단축에 성공적임을 증명하였다. 시험 결과는 악성코드 그룹 분류에도 유용함을 보였으며, 향후 SVM과 같은 분류기를 도입하여 더욱 정확한 악성코드 탐지를 위한 연구를 이어갈 예정이다.

ABSTRACT

The advent of malicious code has increased exponentially due to the spread of malicious code generation tools in accordance with the development of the network, but there is a limit to the response through existing malicious code detection methods. According to this situation, a machine learning-based malicious code detection method is evolving, and in this paper, the feature of data is extracted from the PE header for machine-learning-based malicious code detection, and then it is used to automate the malware through autoencoder. Research on how to extract the indicated features and feature importance. In this paper, 549 features composed of information such as DLL/API that can be identified from PE files that are commonly used in malware analysis are extracted, and autoencoder is used through the extracted features to improve the performance of malware detection in machine learning. It was proved to be successful in providing excellent accuracy and reducing the processing time by 2 times by effectively extracting the features of the data by compressively storing the data. The test results have been shown to be useful for classifying malware groups, and in the future, a classifier such as SVM will be introduced to continue research for more accurate malware detection.

Keywords: autoEncoder, feature importance, malware, TF-IDF, variant

Received(03. 20. 2020), Modified(06. 19. 2020), Accepted(07. 20. 2020)
* 본 연구는 2019년도 호서대학교의 재원으로 학술연구비 지원을 받아 수행되었습니다(2019-0427)
† 주저자, tlp3a1@gmail.com
‡ 교신저자, kinjecs0@gmail.com(Corresponding author)

I. 서 론

최근 네트워크의 발전으로 인해 악성코드의 출현 속도는 매년 급격히 증가하고 있으며 모바일(mobile) 기기와 사물인터넷의 발전으로 다양한 형태의 악성코드가 증가하고 있는 추세이다. 패킹(packing)과 같은 코드(code) 난독화 기법의 대중화로 기존의 악성코드를 이용한 변종 악성코드가 제작되어 백신(Anti-Virus)가 악성코드를 탐지하는데 어려움이 더해지고 있다. 글로벌 사이버 보안 기업 시만텍의 2018년 "인터넷 보안 위협 보고서(ISTR)"에 따르면 사용자의 동의 없이 악성코드를 설치하는 다운로더 악성코드의 변종은 2017년 대비 92% 증가하였으며, 랜섬웨어(ransomware) 악성코드의 변종은 46% 증가하였다고 발표했다[1]. 새로운 변종 악성코드는 지속적으로 증가하고 있으므로 정확한 악성코드 탐지 기법이 요구된다.

악성코드 문제를 해결하기 위한 기존의 악성코드 탐지 방법 중 하나인 백신은 서명(signature) 기반의 탐지 기법으로, 악성코드의 해시값을 서명으로 사용한다. 데이터베이스(database)에서 동일한 해시(hash)값을 검색하여 악성코드를 탐지하므로 교묘히 진화하고 있는 변종 악성코드의 탐지에 있어 약한 면모를 보인다. 이러한 기존 탐지 방법의 문제점을 해결하기 위해 악성코드 탐지 기술에 대한 활발한 연구가 진행되고 있다.

악성코드는 윈도우(window)에서 실행되기 위해 윈도우에서 제공하는 라이브러리인 API(Application Programming Interface) 함수와 동적 연결 라이브러리인 DLL(Dynamic Link Library)을 호출한다. API를 통해 파일생성, 레지스트리(registry)변경, 프로세스(process)생성 등과 같은 다양한 행위가 진행되기 때문에 API를 분석하면 악성코드의 행위를 판별해 패밀리를 분류할 수 있다. 또한 악성코드가 실행되면 프로세스, 레지스트리, 네트워크, 메모리와 관련된 DLL을 Import하기 때문에 DLL 분석을 통한 악성코드탐지가 가능하다.

따라서 본 논문에서는 악성코드를 윈도우에서 실행시키기 위해 필요한 정보인 API와 DLL 등을 이용하여 feature를 산출하였으며, 더 나아가 대량의 feature를 이용하여 악성코드 탐지 시 발생할 수 있는 악성코드 탐지 모델의 성능 저하 문제를 해결하기 위해 autoencoder에서 데이터(data)를 더 잘 나타내는 feature를 한 번 더 추출하여 악성코드를 탐지하는 기술에 대한 연구를 진행하고자 한다.

본 논문의 구성은 다음과 같다. 2장에서는 기존의 feature 추출 방법을 이용한 머신러닝(machine larning) 기반의 악성코드 탐지 및 autoencoder와 관련된 연구에 대한 내용을 설명한다. 3장에서는 본 논문에서 진행한 autoencoder 기반의 악성코드 탐지 기술에 대해 설명하고 4장에서는 k-NN을 이용하여 549feature, autoencoder를 통해 산출된 feature, 549feature에서 ExtraTreesClassifier를 통해 산출한 feature importance 및 feature importance와 autoencoder의 feature를 결합한 feature의 정확도를 비교한다. 5장에서는 도출된 결과를 토대로 본 논문의 결론 및 향후 연구 방향을 제시한다.

II. 관련 연구

2.1 AI 기반 악성코드 분석 기술 동향

악성코드의 탐지 및 분류를 위해 이전에 탐지된 악성코드의 특징을 파악하여 분석한다[2]. 기존의 악성코드 분석 방법에는 정적 분석과 동적 분석이 있으며 정적 분석은 악성코드의 실제 실행 없이 그 자체가 갖고 있는 내용들에 기반하여 악의적인 여부를 진단하는 방법이며, 동적 분석은 악성코드를 직접 실행시키고 동작을 살펴봄으로써, 프로그램 내에 악성코드의 포함 여부를 판단하는 기법이다[3][4]. 최근 이와 같은 정적 분석 및 동적 분석을 통해 추출된 feature를 이용한 머신러닝 기반의 악성코드 탐지 기술 연구가 활발히 진행되고 있다.

정적 분석은 악성코드의 PE(Portable Executable) 구조를 분석함으로써 탐지하는 것이 일반적이다. PE 구조란 윈도우 실행 파일 형식으로서, 크게 헤더(header)와 섹션(section)으로 구성되어 있다. 헤더에는 파일을 실행할 때 필요한 전반적인 정보들을 담고 있으며, 섹션에는 실제 프로그램 구성에 대한 정보를 담고 있다. 따라서 악성코드의 실행 없이도 PE 구조에서 악성행위를 탐지할 수 있는 정보를 얻을 수 있다. 악성 행위를 판별할 수 있는 정보가 담긴 API는 PE 헤더내의 구조적 정보인 IAT(Import Address Table)에 존재하며 DLL에 대한 정보 또한 확인 가능하다. Chen은 실시간 응답과 낮은 오 탐지율을 가진 악성 파일과 정상 파일을 비교함으로써 알려진 악성코드와 알려지지 않은

악성코드를 식별하기 위해 데이터 마이닝 기법을 이용한 정적 악성코드 탐지 시스템을 제안했다[5]. k-NN (k-Nearest Neighbors), 의사결정 트리(Decision Tree), Extreme Gradient Boosting(XGBoost) 등 다양한 머신러닝 알고리즘을 선택하여 파일의 악성 여부를 판단하였고, XGboost을 사용하여 탐지한 결과 99.56%의 탐지 정확도를 검증하였다. Ha 등은 정적 분석을 이용해 악성파일에서 문자열을 추출하여 텍스트 마이닝 기법 중 하나인 TF-IDF(Term Frequency-Inverse Document Frequency)을 이용한 악성코드를 분류를 제안하였다[6]. TF-IDF 기법을 적용해 특징값을 가공한 후 이를 Cosine-Similarity 유사성 측정기법을 활용해 유사도 측정값을 도출하였다. 그 후 k-NN 기법을 적용시켜 악성코드를 분류하였고 최종적으로 83%의 정확도를 도출하였다. Table 1. 은 악성코드로부터 추출된 문자열을 TF-IDF 기법을 이용하여 가공한 벡터에 대한 표이다.

동적 분석은 실제 또는 가상머신과 같은 제어 가능한 환경에서 파일을 직접 실행시켜 나타나는 행위를 바탕으로 악성코드를 분석하여 시스템의 변화나 프로세스, 네트워크 변화를 실시간으로 확인 가능하다[7]. Kakisim 등은 다양한 동적 feature를 분석하여 악성코드 실행 파일을 탐지할 수 있는 보다 차별적인 동적 feature를 찾는 연구를 진행하였다[8]. API-call, 시스템 라이브러리 및 오퍼레이션과 같은 동적 분석에서 feature를 추출하였고, SVM(Support Vector Machine), HMM(Hidden Markov Model) 등과 같은 다양한 머신러닝 알고리즘을 사용하여 악성코드 탐지 연구를 진행하였다. Toshiki 외 3인은 동적 분석을 통해 악성코드로부터 수집한 feature를 RNN(Recurrent Neural Network)이용해 악성코드 분석 방법을 제안하였다[9]. 높은 정확도 달성을 위해 통신 목적을 나타내는 feature를 추출하였으며 결과적으로 분석 시간을 67.1%로 줄였다.

2.2 feature 정제 기술 연구 동향

Autoencoder란 비지도 신경망(unsupervised neural network)으로 입력층(input layer)의 뉴런과 출력층(output layer)의 뉴런을 같은 개수로 두고, 입력과 출력 값이 같은 값이 되게끔 학습시키는 신경망이다[10]. 인코딩 과정에서 입력된 데이터의 핵심 feature 정보만 은닉층(hidden layer)에서 학습하고 나머지 정보는 손실시키는 원리로 출력값이 입력값과 최대한 같아지도록 학습하면서 feature를 잘 추출할 수 있게 하는 방법으로, autoencoder는 항상 입력을 내부 표현으로 바꾸는 인코더(encoder)와 내부 표현을 출력으로 바꾸는 디코더(decoder) 두 부분으로 구성되어 있다[11]. Fig.1.은 두 개의 뉴런으로 구성된 하나의 은닉층(인코더)와 세 개의 뉴런으로 구성된 출력층(디코더)를 가진 autoencoder를 나타내는 그림이다.

Autoencoder의 종류 중 하나인 스택 autoencoder(SAE, Stacked Autoencdoer)는 autoencoder를 층으로 쌓는 방식으로, 주로 네트워크를 학습할 때 초기 가중치를 초기화에 이용된다. 스택 autoencoder는 여러 개의 은닉층을 가지는 autoencoder이며, 레이어를 추가할수록 autoencoder가 더 복잡한 코딩(부호화)을 학습할 수 있다[11]. 스택 autoencoder는 가운데의 은닉층을 기준으로 대칭인 구조를 가진다. Omid E. David와 Nathan S. Netanyahu는 자동 악성코

Table 1. Example of TF-IDF based vector using string

Word Filename	command	copy	createdirectorya	createfilea	delete	dll
0411c3e0f1fe2cc38d1031523492e475.vir	0.002	0.001	0.001	0.002	0.009	0.004
0428a4234d7585b476fd2f02702c2c32.vir	0	0.036	0.038	0.031	0	0.127
042b57e3495a82d5009efe1b2fc7c8d9.vir	0.029	0.000	0.006	0	0.007	0.791

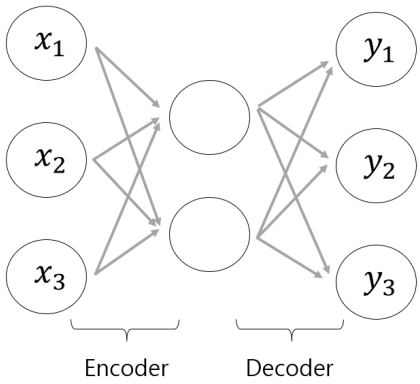

Fig. 1. Architecture of autoencoder

드 서명 생성 및 분류를 위한 새로운 Deep Learning 기반 방법을 제시하였다[12]. 악성코드 데이터셋이 제공되면 샌드박스에서 텍스트 파일을 생성 후 구문 분석을 통해 비트 스트링으로 변환하여 여러 개의 autoencoder를 쌓는 방식으로 구현된 Deep Belief Network(DBN)를 이용해 학습한다. DBN에는 8개의 레이어를 사용하였으며, 이를 통해 악성코드 동작의 변하지 않는 컴팩트한 표현을 생성하고 이를 기반으로 supervised 분류기를 훈련시켜 제안 방법 악성코드 탐지에서 매우 성공적이라는 것을 증명하였다. Fig.2.는 샌드박스에서 부터 서명생성까지의 전체적인 과정을 보여주는 그림이다. Tanuwidjaja 등은 새로운 악성코드의 정확한 탐지를 위해 DFES(Deep Abstraction and Weighted Feature Selection)를 기반으로 하는 수정된 feature 학습 방법을 제안했다[13]. 특징 추출을 위해 스택 autoencoder를 사용하였으며, 특징 선택 및 분류를 위해 ANN(Artificial Neural Network)을 사용했다. 최적의 설정을 위해 원본 데이터의 feature 수 조정 및 modified

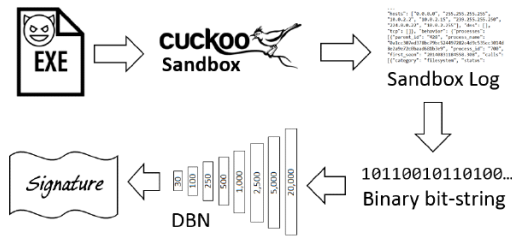

Fig. 2. Overall signature generation process

DFES 적용 등의 다양한 실험을 하였으며, 원본 feature에 대해 기본 DFES가 탐지율등에서 우수한 성능을 제공하며, modified DFES는 기본 DFES 보다 featrue 선택 과정에서 소요되는 시간이 더 빠르다는 것을 검증하였다. Fig.3은 제안 모델 중 하나를 나타낸 그림이다.

원래의 입력에 노이즈를 추가하여 이 입력을 원본으로 만드는 신경망인 노이즈 제거 autoencoder(DAE, Denoising autoencoder)는 입력층에서 은닉층으로 갈 때 노이즈를 추가하는 방식으로 진행된다. 노이즈 추가는 입력에 가우시안(gaussian) 노이즈를 추가하거나, 드롭아웃(dropout)처럼 랜덤하게 일부 노드를 삭제하여 발생시킬 수 있다. Kwon 외 2인은 프로파일링 공격뿐만 아니라 논프로파일링 환경에서도 부채널 분석을 수행할 수 있는 노이즈 제거 autoencoder를 적용한 기법을 제안하였다[14]. 기존의 노이즈 제거 autoencoder와 다르게 노이즈가 제거된 라벨을 이용하여 실제 데이터의 노이즈를 학습하는 방법으로 기존의 기법들과의 비교 실험을 통해 제안 기법의 우수한 성능을 입증하였다. Pascal Vincent 외 4인은 노이즈 제거 autoencoder를 기반으로 딥 네트워크를 구축하기 위한 전략을 탐구하였다[15]. 제안된 모델은 잘 알려진 기본 autoencoder의 변형으

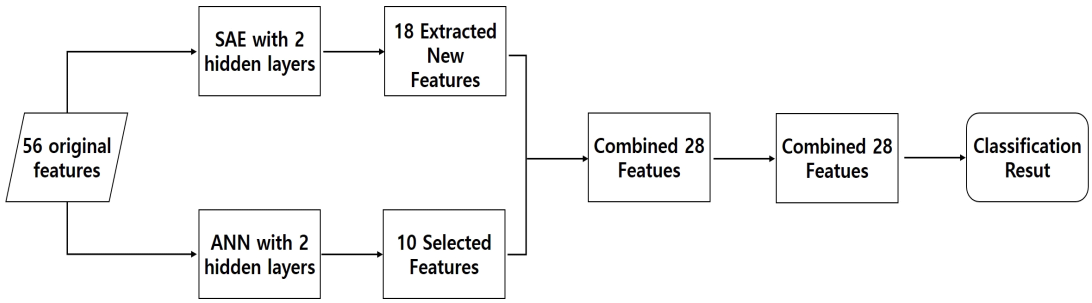

Fig. 3. Dataset consists of 56 features with modified DFES

로서 약간의 노이즈 조정을 통해 높은 성능을 달성함을 보였다.

III. 제안 모델

3.1 시스템 모델

악성코드의 분류를 위해 PE 포맷에서 Imported DLL과 사용되는 API 목록을 추출하고 유의미한 feature를 선별해 통계적 분포에 따른 range 작업으로 feature 가공을 진행한다. 이후 대량의 feature를 이용한 악성코드 탐지 시에 발생할 수 있는 악성코드 탐지 모델의 성능 저하 문제를 방지하기 위해 feature 가공 후 autoencoder를 진행하여 feature 수를 줄인다. Autoencoder를 통한 feature 축소의 유의미함을 검증하기 위해 다양한 방안의 autoencoder 모델에 대한 연구를 진행한다. Table 2.는 Imported DLL과 API 예시이며, Fig.4.는 시스템 모델의 전체적인 과정을 나타낸 그림이다.

Table 2. Imported DLL and API example

Imported DLL	Imported API
KERNEL32.dll	transiatemessage
MSVCRT.dll	createdirectorya
kerne132.dll	gettickcount
Lab07-03.dll	regsetvaluea
WS2_32.dll	regcreatekeya
…	…

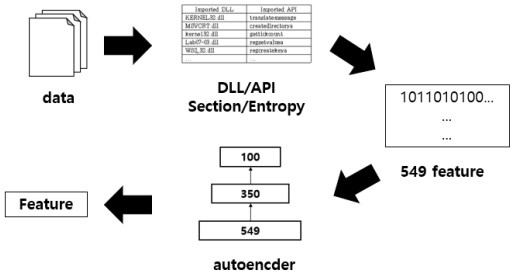

Fig. 4. System configuration

3.2 정적기반 악성코드 Feature 추출

악성코드가 파일을 실행하기 위해서는 다양한 정보들이 필요하며, 이와 같은 다양한 정보들은 PE 파일을 통해 파악 가능하다. PE 파일에는 파일/메모리에서의 시작 위치, 크기, 접근 권한 등의 정보가 담겨 있다. 따라서 악성 행위의 판별을 위해 PE 파일에서 추출 가능한 모든 정보들을 추출하였고 정상과 악성 파일간의 통계적 분석을 진행하였다. SizeOfStackReserve, DllCharacteristics, MajorLinkerVersion, CheckSum, VirtualSize, Image Base, e_lfanew, Subsystem 등 37개 feature를 선별하였고 악성과 정상파일 사이의 통계적 분포에 따라 가공하였다.

Table 3.은 KISA 주관 2018 R&D Data challenge 대회의 "악성코드 탐지" 트랙에서 제공한 Dataset 중 악성 및 정상으로 분류되어 있는 약 20,000개의 데이터를 활용하여 파일 내에 존재하는 섹션을 이용해 특징 값으로 가공한 예시이다. train에는 9,972개의 데이터를 사용하였고 test에는 train에 사용되지 않은 나머지 10,000개의 데이터를 사용하였다. 섹션의 범위가 0~4인 경우 정상 파일의 비중이 더 큰 것을 확인할 수 있고 섹션의 범위가 6을 제외한 5 이상인 경우 악성코드 파일에서 더 많이 나타남을 알 수 있다. 대체적으로 악성코드 파일의 경우 정상파일 보다 섹션을 많이 사용하는 특징이 있으며, 이 특징을 활용하기 위해 파일이 가지고 있는 구조적 데이터를 이용하여 악성 여부를 판단하기 위해 range 가공 작업을 진행하여 feature 값으로 사용하였다.

악성코드의 다양한 행위들은 DLL/API를 통해 이루어지기 때문에 DLL/API 분석으로 행위 판별이 가능하며 특정 유형의 악성코드를 탐지할 수 있다[16]. 따라서 DLL/API 함수를 추출하면 악성코드 파일과 정상파일의 차이점을 알 수 있으며 해당 정보를 추출하여 악성코드 탐지를 진행하였다. DLL/API 정보를 이용하여 악성코드 탐지를 하기 위해 우선 PE파일의 IAT를 통해 로드된 DLL 및

Table 3. Distribution of the number of section features

Number	Range	Normal	Malware
1	0 ~ 4	67.95%	40.80%
2	5	16.89%	34.41%
3	6	8.74%	5.94%
4	7 ~ 12	4.17%	7.77%
5	12 ~ 40	2.25%	11.08%

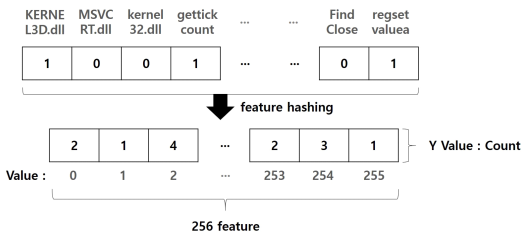

Fig. 5. Feature hashing technique example

사용할 API 목록을 수집하였다. 수집된 DLL/API 명을 해시하여 Mod 256 연산을 진행하였고, 문자열을 0~255 사이의 값으로 매핑하여 256개의 feature로 가공하였다. Fig.5.는 feature hashing 기법 적용 예시를 나타낸다.

대다수의 악성코드들은 백신을 우회하기 위해 패킹 및 난독화를 수행하여 섹션 구분이 어려워지게 된다. 이에 본 논문에서는 이를 해결하기 위해 정보의 양을 나타내는 척도로 사용되는 엔트로피(entropy) 값을 사용한다[17].

패커들은 각각 나름의 규칙을 이용하여 섹션을 만들기 때문에 변종 파일 간 섹션명이 같거나 섹션별 엔트로피 값이 유사할 가능성이 높다. 이러한 특징을 이용하여 X축에는 악성코드와 정상파일간에 존재하는 섹션명으로, Y축에는 섹션별 엔트로피 값으로 구성하였다. 문자열 형태인 섹션명을 정수로 변환하기 위해 MD5 해시 알고리즘을 이용하여 해시를 진행해 16진수로 표현 후 16진수를 다시 10진수로 변환하여 Mod 16을 취해 0~15 사이의 값으로 변환하였다. Fig.6.은 섹션명을 feature 해싱 차원축소 기법을 이용하여 X값을 추출하는 과정을 나타낸 그림이다.

엔트로피값은 0에서 8사이의 값을 가지게 된다.

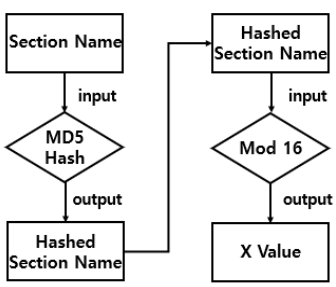

Fig. 6. Extraction of x-values using section name feature hashing dimension technique

본 논문에서는 이 엔트로피값을 0.5단위로 나누어 Y축의 범위를 0~8이 아닌 0~15으로 분석하여 더 세밀한 분석이 가능하게 하였다. 섹션명에 해당하는 X축의 범위가 0~15, 엔트로피에 해당하는 Y축의 범위가 0~15로, (0,0)에서 (15,15)까지 256개의 격자가 존재하며, 격자별 발생횟수가 나타난다. 데이터가 중복일 경우 카운트 값을 의미하게 되고, 16x16의 2차원 배열을 1차원 배열로 변환하여 256개의 feature 값으로 사용한다.

3.3 SAE 기반 feature 정제

autoencoder는 2장에서 설명한 바와 같이 입력과 출력이 같은 값이 되게끔 학습시키는 신경망으로서, 은닉층에서 데이터를 압축적으로 저장하여 feature를 효과적으로 추출할 수 있게 하는 방법이다. 이는 고차원의 데이터가 있는 경우 발생 할 수 있는 feature 분산 문제를 방지하기 위한 방안이며, 본 논문에서는 이러한 autoencoder의 특징을 이용하여 악성코드 feature를 효과적으로 추출하여 악성코드를 탐지하는 방법으로 접근하였다. autoencoder는 feature 수, autoencoder의 층의 개수, 드롭아웃(dropout) 비율에 따라 약간씩 다른 방식으로 접근하였으며 기본적인 autoencoder 진행 방식은 SAE 방식에 기반하여 진행되었다. 진행한 SAE 접근 방식의 모든 입력으로 PE 파일로부터 추출된 549개의 feature를 사용하였다. 첫 번째 방식인 feature 수는 autoencoder에서 생성되는 최종 feature 수를 다르게 하는 방식으로서, 총 2개의 autoencoder를 이용해 학습하였다. 두 번째 방식인 autoencoder 층의 개수는 첫 번째 방식과 달리 autoencoder에서 생성되는 최종 feature 개수는 동일하지만 feature 추출 과정인 autoencoder 학습 과정에서 autoencoder의 stack의 개수를 다르게 진행한 방식이다. 기본 autoencoder부터 세 개의 autoencoder를 쌓아 올려 만든 SAE까지 다양한 방식으로 접근하였다. 마지막으로 드롭아웃 비율을 다르게 적용한 방식은 최종 생성되는 feature와 autoencoder 개수는 동일하지만 autoencoder에 feature가 입력이 되었을 때, 이 입력을 은닉층으로 축소하는 과정에서 적용하는 드롭아웃 비율을 조정한 방식이다. 첫 번째와 두 번째 방식에서는 드롭아웃을 적용하지 않았으며 세 번째 방식에서 드롭아웃이 미

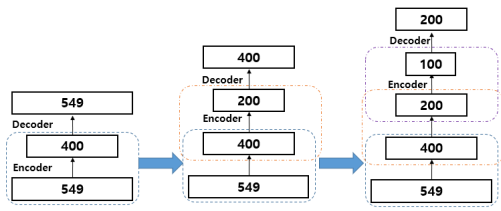

Fig. 7. Three-layer(549-400-200-100) autoencoder

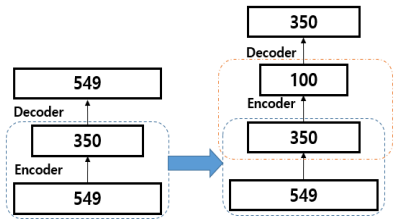

Fig. 8. Two-layer(549-350-100) autoencoder

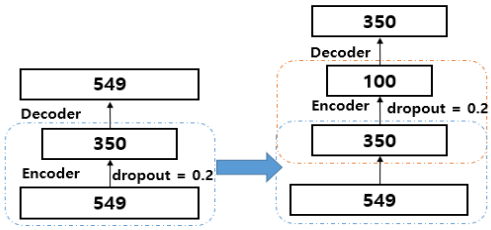

Fig. 9. Autoencoder with Dropout(0.2)

치는 영향을 알아보기 위해 드롭아웃을 적용하여 진행하였다. Fig.7~9.는 각각의 방식에 대한 과정을 나타낸 것이다.

3.4 Importance 기반 Feature 정제

추출된 대량의 feature들 중에서 모델의 성능에 영향을 거의 미치지 않거나 또는 성능 저하의 원인이 될 수 있는 feature들이 존재할 수 있다. 이를 방지하기 위해 feature importance를 선별하는 과정이 필요하며 본 논문에서는 ExtraTreeesClassifier를 사용하여 feature importance를 추출하였다.

ExtraTreesClassifier는 기본적으로 의사 결정 트리를 기반으로 하는 앙상블 학습 방법으로 특정 의사 결정 및 데이터 하위 집합을 무작위화하여 데이터에서 과잉 학습 및 과적합을 최소화한다[18]. ExtraTreesClassifier를 이용해 PE 헤더에서 추출된 549개의 각 feature들의 importance 점수를 계산하였고, 점수가 높은 상위의 feature들 중 악성코드 탐지 성능의 향상에 가장 도움이 되는 40개를 feature로 선택하였다.

IV. 시험 결과

본 절에서는 제안 모델의 성능 검증을 위해 세 가지의 관점에서 feature 추출 후 시험을 진행하여 각각의 결과에 대한 비교를 진행한다. 비교를 진행할 세 가지 관점의 feature 중 첫 번째는 PE파일에서 추출한 549개의 feature이다. 549개의 feature는 악성코드 분석에서 범용적으로 사용되는 정보를 추출한 것으로, 기존 연구 대비 제안 모델의 우수성을 증명하기 위해 비교를 진행한다. 두 번째는 본 논문에서 제안하는 모델을 통해 추출한 feature로, 세 가지 autoencoder 방식으로 학습 모델 생성 후 이를 이용하여 추출한 feature이다. 마지막으로 세 번째는 feature importance 방식을 통해 선택된 featrue로, 549개 feature들의 importance를 추출하여 상위 40개의 feature 선택하였고 이를 autoencoder와 결합하여 시험을 진행하였다. 제안 모델의 성능 검증은 악성 코드 탐지의 정확도 및 feature 추출 후 악성코드 탐지에 소요되는 시간의 비교를 통해 진행하였다. 이때 정확도 측정 및 소요시간 측정은 지도 학습의 k-NN 알고리즘을 사용하였다. k-NN 알고리즘을 통해 정확도 산출 시 k 값은 3, 유사도 측정법은 cosine distance 방법으로 진행하였다. 검증 실험에 사용된 데이터는 KISA 주관 2018 R&D Data challenge 예선 및 본선 데이터 중 약 40,000개를 이용하여 진행하였으며, 학습 데이터로 약 30,000개, 테스트 데이터로 10,000개를 사용하였다.

4.1 시험 환경

시험을 위한 시스템 환경은 다음과 같다.

CPU : Intel(R) Core(TM) i7-8700K, 3.70 GHz

메모리 : 16G Bytes

운영체제 : Windows 10 Education

4.2 시험 결과

실험은 3.3절에서 설명한 autoencoder를 구현하여 악성코드 탐지에 있어 최적의 방안을 찾기 위해 여러 가지 방식으로 진행하였다. 제안 모델의 성능 검증을 위해 앞서 설명한 바와 같이 세 가지 관점에서 feature를 추출하였으며 해당 feature들을 이용하여 k-NN 결과를 도출하였다.

4.2.1 Original feature

PE 파일에서 정상 파일과 악성 파일을 구분할 수 있는 정보들이 담긴 정보들을 추출하여 생성한 549개의 feature를 이용하여 k-NN 알고리즘을 적용한 결과를 도출하였다. 데이터 셋을 이용하여 도출한 결과 정확도 93.39%, 시간은 총 1.07분이 소요되었으며, 한 개의 PE파일을 이용하여 549개의 feature를 생성해 k-NN 결과를 도출하는데 총 00.25초가 소요되었다. 이 결과를 autoencoder에서 생성된 feature와 비교하였다. Table 4.와 Table 5.는 추출된 feature로 변종 악성코드 또한 탐지할 수 있음을 나타낸다.

Table 4. Reference Malicious File

FileName	SHA256
00c7f12a53de78…	e20a48041f75…

Table 5. Variant Malicious File

FileName	SHA256
5d22fe56478b0…	904c7704f2011…
9ed61d61089e2…	fe7c97a611c6d…
9561d19076e43…	f382426975956…
792976741f97f2…	ab0f95cc230b6…

4.2.2 Autoencoder feature

Fig.10. 및 Fig.11.은 549개의 feature와 autoencoder를 통해 생성된 feature를 k-NN을 이용하여 정확도 및 시간을 측정한 결과에 대한 그래프이다. feature의 개수, autoencoder의 사용 개수(layer 개수) 및 드롭아웃 비율을 각기 다르게 하여 실험을 진행하였다. Autoencoder feature 추출에 대한 세 가지 방안 모두, 한 개의 PE 파일을

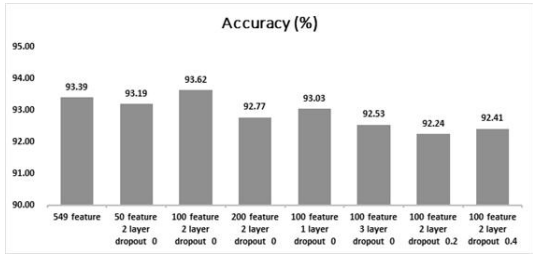

Fig. 10. 549 feature and autoencoder feature k-NN accuracy

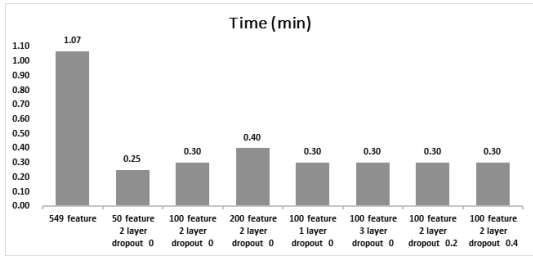

Fig. 11. 549 feature and autoencoder feature k-NN time

이용해 549개의 feature를 생성하고 autoencoder 기법을 통해 feature를 줄여 k-NN 결과를 도출하는데 약 01.16초 정도의 시간이 소요되었다. autoencoder 과정 없이 feature를 추출한 방안이 00.25초라는 것을 볼 때, autoencoder 과정을 추가하여도 성능 저하가 크지 않다는 것을 확인할 수 있다. 따라서 다양한 feature 정제 시험의 성능 비교는 feature 추출 후부터 진행되는 악성코드 탐지 정확도 및 이에 소요되는 시간에 대해서만 진행하였다.

Fig.10. 그림을 보면 미세하지만 세 번째 막대에 해당하는 2개의 autoencoder 층에서 dropout 비율 0으로 진행하여 최종적으로 100개의 feature를 생성한 방안의 정확도가 93.62%로 가장 높은 것을 확인할 수 있다. 더불어 Fig.11. 그림을 보면 549개의 feature보다 autoencoder를 통해 더욱 정제된 feature를 생성하였을 때의 feature가 549개의 feature보다 시간 면에서 매우 우수한 것을 볼 수 있다.

4.2.3 Feature importance

PE파일로부터 추출된 549 feature에서 feature importance를 추출하기 위해

ExtraTreesClassifier를 사용하였으며, 상위 40개의 feature를 추출하였다. 상위 40개의 feature를 k-NN 알고리즘을 적용한 결과는 정확도 70.12%, 시간은 22초로 정확도 측면에서는 다소 떨어지지만, 시간은 절반 이상 감소하여 feature importance를 이용한 k-NN 측정의 시간 측면에서는 우수한 성능을 보이는 것을 확인할 수 있다.

4.2.4 Autoencoder feature/feature importance 결합

549개의 feature로부터 산출된 feature importance 40개와 autoencoder의 각 방식에서 생성된 feature를 결합하여 k-NN 알고리즘을 적용해 결과를 확인하였다. Fig.12.와 Fig.13.은 feature importance 40개에 대한 결과 및 feature importance와 각 autoencoder 방식을 결합했을 경우의 k-NN의 정확도 및 시간에 대한 결과다. 정확도 측면에서는 feature importance만 추출했을 때보다 autoencoder를 결합하였을 때의 결과가 전체적으로 더욱 우수하며 그 중 하나의 autoencoder로 드롭아웃을 적용하지 않고 100개의 feature를 추출한 접근법이 가장 우수하며 시간측면에서는 feature importance 상위 40개의 k-NN 결과가 조금 더 적은 시간이 소요된 것을 확인할 수 있다. Fig.10.과 Fig.11.을 비교해 보면 전체적으로 feature importance의 방식과 autoencoder 방식을 결합한 기법은 악성코드 탐지 성능이 평균적으로 낮은 것을 확인할 수 있으며, autoencoder 기법만 사용한 경우가 시간 및 정확도 측면에서 가장 뛰어난 것을 볼 수 있다.

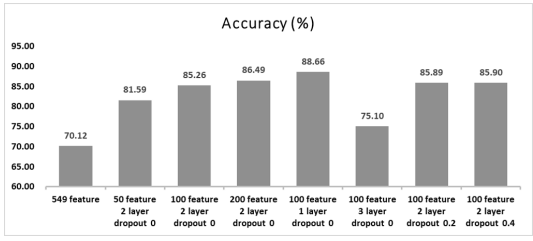

Fig. 12. Feature importance and autoencoder feature combined with important feature k-NN accuracy

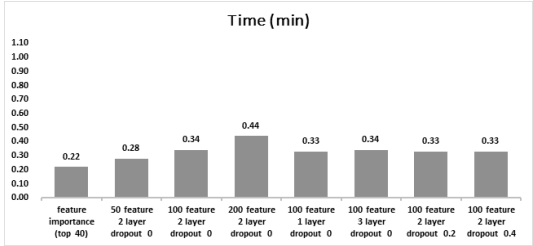

Fig. 13. Feature importance and autoencoder feature combined with important feature k-NN time

V. 결 론

정보 기술의 발전에 따라 악성코드 역시 급속도로 증가하였으며 기존의 악성코드 탐지 기법은 한계에 부딪히고 있다. 이에 따라 기존의 기법을 보완할 수 있는 새로운 방안이 요구되고 있다.

본 논문에서는 악성코드 분석에서 많이 사용되는 PE 구조에서 악성행위의 특징을 분석할 수 있는 DLL과 API 등을 정보를 이용하여 feature를 추출하였고, 이를 autoencoder의 입력으로 하여 feature를 생성하여 악성코드를 탐지하는 기법을 연구하였다. 인코딩 과정에서 입력된 데이터의 핵심 특정 정보만 은닉층에서 학습하는 autoencoder의 원리를 바탕으로 진행한 것이며, feature 수는 줄이면서 악성코드 탐지 성능은 향상시킬 수 있는 모델을 위해 다양한 접근법을 제안하여 연구를 진행하였다. 다양한 방식의 autoencoder 모델의 성능 검증을 위해 549개의 feature 및 549개 feature에서 importance를 추출하여 점수가 높은 상위 40개를 선택한 feature의 k-NN 결과를 비교하였다. 결론적으로 2개의 autoencoder를 통해 100개의 feature를 생성한 SAE 방식이 k-NN의 결과를 도출하는데 약 30초가 소요되어 autoencoder를 적용하지 않은 방식보다 시간 측면에서 우수하게 나왔으며 정확도 또한 뒤처지지 않는다는 점을 확인할 수 있었다. 이외의 다른 방식으로 autoencoder에 접근하여 추출한 feature들 역시 기존의 549개의 feature와 비교하였을 때 악성코드 탐지에 있어 유사한 정확도를 보이며 탐지 시간에 있어 더 나은 결과를 보인다는 것을 검증하였다. 이에 따라 대량의 feature를 이용한 머신러닝 기반의 악성코드 탐지 시 발생할 수 있는 문제를 해결하기 위해 본 논문에서 진행한 접근 방식들이 도움이 될 수 있음을 확인하였다.

향후, 앞선 연구들을 기반으로 여러 악성코드 탐지 기법들에 대해 연구하고 SAE에 SVM 분류기를

적용하여 더 나은 정확도를 도출할 수 있는 연구 및 정확한 변종 악성코드를 탐지가 가능한 방안에 대한 연구를 진행할 예정이다.

References

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

[2] Jae-woo Parck, Sung-tae Moon, Gi-wook Son, In-kyoung Kim, Kyoung-soo Han, Eul-gyu Im and Il-gon Kim, "An Automatic Malware Classification System using String List and APIs," Journal of Security Engineering, 8(5), pp. 611-626, Oct. 2011.

[3] Lenny Zeltser, "Mastering 4 Stages of Malware Analysis," https://zeltser.com/mastering-4-stages-of-malware-analysis/, Feb. 2015.

[4] Kyung-min Kim, "Malware analysis method," https://brunch.co.kr/@kali-km/5, Jun. 2016.

[5] Zhi-guo Chen, "A Learning-based Static Malware Detection System with Integrated Feature Selection," Ph.D. Graduate School of Konkuk University, Feb. 2019.

[6] Ji-hee Ha and Tae-jin Lee, "TF-IDF based PE File Embedding and Malware Classification," Journal of the Korean Institute of Information Security and Cryptology Summer Conference, pp. 11-14, Jun. 2019.

[7] Seong-bin Park, Min-soo Kim and Bont-nam Not, "Detecion Method Using Common Feature of Malware Variants Generated by Automated Tools," Journal of Korean institute of information technology, 10(9), pp. 67-75, Sep. 2012.

[8] A.G. Kakisim, M. Nar, N. Carkaci and I. Sogukpinar, "Analysis and Evaluation of Dynamic Feature-Based Malware Detection Methods," 11th International Conference Sec ITC, pp. 247-258, Nov. 2018.

[9] T. Shibahara, T. Yagi, M. Akiyama, D. Chiba and T. Yada "Efficient Dynamic Malware Analysis Based on Network Behavior Using Deep Learning," 2016 IEEE Global Communications Conference (GLOBECOM), pp. 1-7, Dec. 2016.

[10] Kyung-lyul Hyun and Je-hee Lee, "Motion Data Analysis with Autoencoder," Korea Computer Graphics Society Conference, pp. 133-134, Jun. 2013.

[11] A. Geron, Hands-On Machine Learning with Scikit-Learn & TensorFlow, Hanbit Media, Apr. 2018.

[12] O. David and N. S.Netanyahu, "DeepSign: Deep Learning for Automatic Malware Signature Generation and Classification," International Joint Conference on Neural Networks (IJCNN), pp. 1-8, July. 2015.

[13] H.C. Tanuwidjaja and Kwang-jo Kim, "Enhancing Malware Detection by Modified Deep Abstraction and Weighted Feature Selection," 2020 Symposium on Cryptography and Information Security, pp. 1-8, Jan. 2020.

[14] Dong-geun Kwon, Sunghyun Jin, Hee-seok Kim and Seok-hie Hong, "Improving Non-Profiled Side-Channel Analysis Using Auto-Encoder Based Noise Reduction Preprocessing," Journal of The Korea Institute of Information Security and Cryptology, 29(3), pp. 491-501, Jun. 2019.

[15] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio and P.A. Manzagol,

"Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion," The Journal of Machine Learning Research, vol. 11, no. 110, pp. 3371-3408, Dec. 2010.

〔16〕 Ji-hee Ha, Su-jeong Kim and Tae-jin Lee, "Feature Extraction using DLL/API Statistical Analysis and Malware Detection based on Machine Learning," The Journal of Korean Institute of Communications and Information Sciences, 43(4), pp. 730-739, Apr. 2018.

〔17〕 Hee-jun Kwon, Sun-woo Kim and Eul-gyu Im, "An Malware Classification System using Multi N-gram," Journal of Security Engineering, 9(6), pp. 531-542, Dec. 2012.

〔18〕 Medium, "ExtraTreesClassifier", https://medium.com/@namanbhandari/extra treesclassifier-8e7fc0502c7, Oct. 2018

〈저자소개〉

김 홍 비 (Hong-bi Kim) 정회원
2020년 2월: 호서대학교 정보보호학과 졸업
2020년 3월~현재: 호서대학교 정보보호학과 석사과정
〈관심분야〉 악성코드 분석, 정보보호

이 태 진 (Tae-jin Lee) 종신회원
2003년 2월: 포항공과대학교 컴퓨터공학과 졸업
2008년 2월: 연세대학교 컴퓨터공학과 석사 졸업
2013년 1월~2017년 2월: 한국 인터넷진흥원 팀장
2017년 2월: 아주대학교 컴퓨터공학과 박사 졸업
2017년 3월~현재: 호서대학교 정보보호학과 교수
〈관심분야〉 시스템 보안, 악성코드 분석, 침해사고 대응, AI