Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.32, NO.2, Apr. 2022 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2022.32.2.381

기계학습 기반 악성코드 검출을 위한 이미지 생성 방법

전 예 진,[1†] 김 진 이,[1‡] 안 준 선[2]
[1,2]한국항공대학교 (학생, 교수)

Image Generation Method for Malware Detection Based on Machine Learning

YeJin Jeon,[1†] Jin-e Kim,[1‡] Joonseon Ahn[2]
[1,2]Korea Aerospace University (Undergraduate student, Professor)

요 약

기계학습 이미지 인식 기술의 발전에 따라 이를 악성코드 검출에 적용하는 방법이 연구되고 있다. 그 대표적인 접근법으로 악성코드 파일을 이미지로 변환하고 이를 CNN과 같은 딥러닝 네트워크에 학습시켜 악성코드 검출과 분류를 수행하는 연구가 진행되어 의미 있는 결과가 발표되고 있다. 본 연구에서는 기계학습을 사용한 악성코드 검출에 효과적인 이미지 생성방법을 제시하고자 한다. 이를 위하여 이미지 생성의 여러 선택 요소에 따른 악성코드 검출의 성능을 실험하고 분석하였으며, 그 결과를 반영하여 명령어 흐름의 특성을 좀 더 명확하게 나타낼 수 있는 선형적 이미지 생성방법을 제시하고 이 방법이 악성코드 검출의 정밀도를 높일 수 있음을 실험을 통하여 보였다.

ABSTRACT

Many attempts have been made to apply image recognition based on machine learning which has recently advanced dramatically to malware detection. They convert executable files to images and train deep learning networks like CNN to recognize or categorize dangerous executable files, which shows promising results. In this study, we are looking for an effective image generation method that may be used to identify malware using machine learning. To that end, we experiment and assess the effectiveness of various image generation methods in relation to malware detection. Then, we suggest a linear image creation method which represents control flow more clearly and our experiment shows our method can result in better precision in malware detection.

Keywords: malware detection, static analysis, machine learning, image recognition

I. 서 론

악성코드는 컴퓨터 시스템의 주요 보안 위험 중 하나로 최근에는 랜섬웨어, 암호화폐 불법 채굴 등 침해의 형태가 확대되고 있으며 그 피해 규모도 계속 증가하고 있다[1]. 이러한 악성코드를 사전에 검출하여 침해를 차단하기 위하여 동적 분석 및 정적 분석과 실행 시간 탐지 등 다양한 방법이 사용되고 있으나, 그 회피 기법 역시 고도화되고 있어 이에 대한 지속적인 대응이 요구되고 있다.

최근에는 급속히 발전하고 있는 기계학습 기술을 악성코드 검출에 적용하는 방법이 새로운 대안으로 주목받고 있다. 특히 실행 파일을 이미지로 변환한 후 여기에 이미지 인식 기술을 적용하여 악성 여부를 판별하거나 악성코드를 분류하는 연구가 진행되어 의미 있는 결과가 발표되고 있다. 최신의 발전된 심층학습(deep learning) 기술은 많은 양의 입력 데이터와 다단계의 학습 네트워크를 활용한 이미지의 특징 추출에 우수한 성능을 보여, 이를 악성코드로 생성한 이미지에 적용할 경우 적은 노력으로 효과적인

Received(02. 04. 2022), Modified(03. 18. 2022), Accepted(03. 22. 2022)
† 주저자, jn70425@kau.kr
‡ 교신저자, jini1651@kau.kr(Corresponding author)

검출 및 분류를 할 수 있을 것으로 기대되고 있다.

본 연구에서는 기계학습 기반의 이미지 인식 기술을 악성코드 검출에 적용하는 데 효과적인 이미지 생성방법을 제시하고자 한다. 이를 위하여 이미지의 해상도, 실행파일 구조의 반영, 컬러 이미지 표현의 사용, 역어셈블(disassemble)을 사용한 코드 의미 반영을 이미지 생성을 위한 주요 인자로 설정하고, 각 인자가 악성코드 검출의 성능에 주는 영향을 분석하였다. 이를 통해 이미지 표현이 명령어 실행 흐름의 패턴이나 특정 API의 호출 순서와 같은 악성코드 검출에 필요한 특성을 잘 나타내면서 불필요한 잡음 정보는 최소화하여야 검출의 성능을 높일 수 있다는 결론을 도출하였으며, 이러한 결과에 기반하여 악성코드 검출에 효과적인 선형적 이미지 생성방법을 제시하였다. 제시된 방법은 명령어의 제어 흐름과 관계없는 이미지 정보의 생성을 줄이고 필요한 코드의 특징을 이미지에 효과적으로 표현하는 특징을 가지며, 벤치마크 집합을 사용한 실험 결과 악성코드 검출의 정밀도를 높이는 것으로 분석되었다.

본 논문의 구성은 다음과 같다. 2절에서는 악성코드 검출을 위한 기존의 연구 결과를 개괄적으로 기술한다. 3절에서는 악성코드 검출을 위한 이미지 생성에 있어 고려해야 할 요소들을 제시하며, 4장에서는 각 요소의 영향에 대한 실험 및 분석 결과와 함께, 악성코드 검출에 효과적인 선형적 이미지 생성방법을 제시하고, 5장에서 결론으로 맺는다.

II. 관련 연구

일반적인 악성코드 검출 방법은 코드를 안전한 환경에서 수행하면서 그 특징을 분석하는 동적 분석(dynamic analysis)과 코드를 수행하지 않고 코드의 악성 여부를 판별하는 정적 분석(static analysis)으로 구분된다. 이미지 인식에 기반한 악성코드 검출은 일종의 정적 분석 방법으로서 악성코드 수행 부담이 없을 뿐 아니라, 일반적인 정적 분석의 역어셈블 과정과 복잡한 의미 기반 분석 없이도 유용한 악성코드 검출 성능을 얻을 수 있으므로 많은 관련 연구가 진행되었다.

Nataraj 등의 연구는 이미지 인식 기반의 악성코드 분석 연구를 제시한 초기의 사례로서 악성코드를 흑백 이미지로 표현하고 이미지의 질감(texture)을 구분하는 특징 벡터에 k-nn(k-nearest neighbor) 알고리즘을 적용하여 악성코드를 자동으로 분류하는 방법을 제시하였다[2]. 이후 이미지 인식에 심층학습의 CNN(Convolutional Neural Network) 모델을 활용하는 연구가 진행되어, 악성코드 검출과 악성코드 분류를 위한 연구 결과가 발표되었다[3,4,5]. 또한, Bhodia 등은 악성코드 분류에 있어서 심층학습의 CNN 모델과 특징 벡터를 사용한 k-nn 알고리즘의 성능을 비교하고, 제로데이 악성코드에 대한 검출에서 심층학습을 사용한 방식의 성능이 더 뛰어남을 제시하였다[6].

최근에는 효과적인 이미지 생성을 통해 그 성능을 높이고자 하는 연구가 다양하게 진행되었다. He 등은 RGB 컬러 채널의 이미지가 CNN 기반 악성코드 탐지에 어떤 영향을 미치는지 실험하여 RGB 이미지 방식이 그레이스케일 이미지 방식보다 전체적인 정확도는 높지만, 중복 API 주입에 대한 탐지 능력은 더 낮아지는 결과를 보고하였으며[7], Zhang 등은 R 채널에는 바이너리로 코드의 바이트를 그대로 저장하고 G 채널에는 PE 파일의 섹션 정보, B 채널에는 문자 코드 정보를 저장하여 멀웨어 검출 성능을 높이고자 하였다[8].

바이너리 실행 파일을 그대로 이미지로 사용하는 방법 대신, 악성코드를 역어셈블 하여 코드의 의미적 특성을 이미지 생성에 반영하는 연구도 진행되었다. Liu 등은 역어셈블한 코드를 제어 흐름 그래프에 따라 재배열하여 바이너리 코드 이미지를 생성함으로써 난독화와 코드 변조에 대응하는 방법을 제시하였고[9], Ni 등은 명령어 바이트열의 유사성을 유지하는 해쉬 알고리즘으로 주요 명령어 5가지의 순열을 인코딩하여 이미지를 생성하는 방법을 제시하였다[10]. Yan 등은 실행파일의 흑백 이미지에 대한 CNN 분석과 역어셈블한 명령어 열에 대한 LSTM(Long Short-Term Memory) 분석을 함께 수행하여 악성코드를 분류하는 방법을 제안했으며[11], 정성민 등은 기본 블록(basic block) 내의 명령어들로 하나의 바이트열을 만들고 이러한 바이트열들의 순열에 피쳐 해싱(feature hashing)을 적용하는 방법을 제시하여 명령어들의 순서 정보가 중요함을 보였다[12].

Table 1은 관련 연구를 표로 정리한 것이다. 특성 벡터에 대하여 k-nn 등의 분석 모델을 적용한 초기의 연구 이후에는 대부분 흑백 이미지에 대한 CNN 학습 모델이 사용되었다. 악성코드 여부를 판단하는 이진 분석에 대해서는 95% 내외의 정확도를 보이며 악성코드 분류의 경우에는 상대적으로 더 높

Table 1. Summary of Related Works on Image based Malware Classification and Identification

reference	image color (size)	analysis model	classification / detection	data set	performance		
					recall	precision	accuracy
Nataraj [2]	gray (various)	k-nn	classification	Anubis analysis system			98-99%
Choi [3]	gray (256x256)	CNN	detection	Hauri, Kaist			95.6%
Bojan [4]	gray (various)	CNN (MalConv)	detection	self-collected data	84.68± 11.71%	92.83 ±5.56%	
Seok [5]	gray (256x256)	CNN	detection	Microsoft	92.1%	91.7%	96.1%
				VXHeaven	76.9%	80.8%	82.9%
Bhodia [6]	gray (various)	CNN (Resnet)	classification	Malimg			98.39%
				Malicia			97.61%
			detection	Malimg			94.8%
				Malicia			92.93%
He [7]	gray(various)	CNN (Resnet)	classification	andro-dumpsys	79.1%	46.8%	
	RGB(various)				70.3%	54.2%	
Zhang [8]	RGB(various)	CNN (VGGnet)	classification	Malimg	97.2%	97.2%	98.9%
Liu [9]	gray(512x512)	Selective Ensemble, K-means	classification	self-collected data			98%
Ni [10]	(binary data)	CNN	classification	Microsoft			98.8%
Yan [11]	gray(64x64), instr. seq.	CNN, LSTM (Ensemble)	detection	Microsoft, online SW providers,			99.88%
Jeong [12]	instr. seq.	CNN (1-D Conv.)	classification	self-collected data			92%
			detection				94%

은 정확도를 보인다.

기존 연구와 비교하여 본 연구는 다음과 같은 차별점을 갖는다. 악성코드 검출을 위한 기존 연구의 경우 고안한 이미지 기법의 효과를 증명하기 위하여 서로 다른 테스트 데이터를 사용하고 있으므로, 다양한 기법들의 효과를 서로 비교하기 어려운 한계가 있다. 본 연구에서는 이미지 생성 시 고려되는 다양한 요소들을 같은 테스트 데이터에 적용하여 실험함으로써 이러한 요소들이 검출 효과에 미치는 영향에 대한 실제적인 정보를 제공하고자 하였다.

또한, 기존의 연구들이 모두 정방형 또는 직사각형 형태의 이미지를 사용하고 있는 것에 대하여, 실험 데이터의 결과 분석을 기반으로 악성코드 검출에 효과적인 선형적 이미지 생성 기법을 제시하고 이에 대한 성능 평가를 수행하였다.

III. 이미지 인식 기반 악성코드 판별 모델

3.1 시스템 개요

이미지 인식에 기반한 악성코드 식별은 이미지 생성과 학습 과정으로 이루어진다. 이미지 생성 과정은 입력 실행 파일을 하나 또는 복수의 이미지 파일로 변환하는 과정이며, 이를 위하여 입력 파일에 대한 분석, 역어셈블, 인코딩(encoding) 등 다양한 작업이 적용될 수 있다. 학습 과정에서는 생성된 악성코드와 정상코드로부터 생성된 이미지를 심층학습 신경망으로 학습하여 악성코드를 검출하게 된다.

생성되는 이미지는 악성코드의 특성을 잘 보존하면서도, 악성코드와 무관한 특성과 악성코드 특성의 간섭이 최소화되는 형태이어야 한다. 악성코드와 관련 없는 정보가 불필요하게 포함된다면 악성코드 검출의 성능을 떨어뜨릴 수도 있다. 이에 관하여 본 연구에서는 이미지 생성을 위한 다양한 요소를 설정하고 악성코드 검출에 미치는 영향을 분석한 후에 악성코드 검출에 효과적인 이미지 생성 방법을 제시하고자 하였다.

악성코드 식별을 위한 이미지 학습 모델로 다수의 관련 연구에서 사용된 Resnet34를 사용하였다[6,7]. Resnet34는 CNN 기반의 심층학습 네트워크로서, 복잡하고 정교한 학습 성능을 보이며 건너뛴 연결(skip connection)을 사용하여 기울기 소실

(gradient vanishing)과 성능 저하(degration) 문제를 완화하는 장점이 있다.

3.2 이미지 생성을 위한 고려 요소

본 연구에서는 이미지 생성의 고려 요소로 이미지의 크기, 컬러 이미지 사용의 효과, 실행 파일 구조의 반영, 역어셈블을 통한 코드 의미 반영 등을 설정하였다.

3.2.1 이미지 해상도

실행 파일은 다양한 크기를 가지므로, 학습의 부하를 줄이고 악성코드를 식별할 수 있는 적절한 크기를 선택하는 것이 필요하다. 이미지 크기가 너무 크면 작은 실행 파일의 경우 임의로 채워지는 부분이 늘어나게 되고, 학습의 부하가 필요 없이 증가할 수 있다.

이미지를 생성하기 위해서는 실행 파일의 앞에서부터 필요한 만큼의 바이너리 값을 추출하여 이미지상의 픽셀에 할당하게 된다. 악성코드의 크기가 이미지 크기보다 클 경우 코드 파일의 뒷 부분이 버려지며, 악성코드가 작을 경우에는 이미지의 뒷부분은 기본값인 0으로 채워진다. 기존 관련 연구들은 128x128에서 256x256 정도 해상도의 이미지를 대부분 사용하고 있다.

3.2.2 실행 파일의 섹션 구분 반영

실행 파일은 여러 섹션(section)들로 구성되어 있으며, 각각의 섹션은 다른 종류의 정보를 가진다. 본 연구에서 대상으로 하는 PE(Portable Executable) 파일의 경우 실행 명령어를 저장하는 .text 섹션과 전역 변수의 값을 저장하는 .data 섹션 그리고 임포트(import)와 익스포트(export)되는 주소값을 저장하는 .idata 및 .edata 섹션 등으로 구성된다. 악성코드 판별에 유의미한 섹션을 파악하여 이를 이미지 생성에 반영하면 악성코드 검출의 성능을 높일 수 있을 것이다.

3.2.3 컬러 이미지 표현 사용

대부분의 기존 연구는 흑백 이미지를 사용하였으나, RGB로 구성된 컬러 이미지를 생성한 연구 결과도 일부 발표되었다[7,8]. 컬러 이미지를 사용한다면, 다른 섹션을 색으로 구분하거나, 하나의 명령어를 구성하는 명령어 코드(OP code)와 피연산자를 색깔 요소에 구분하여 저장함으로써 추가적인 의미를 이미지에 부여할 수 있다. 또한, RGB 이미지를 사용하여 해상도는 유지하면서 추가적인 정보를 세 배 더 포함시키는 연구도 수행되었다.

3.2.4 역어셈블을 사용한 코드 의미 반영

실행 파일의 코드 부분을 이미지로 생성하는 데 있어, 각각의 바이트를 이미지의 픽셀로 단순 매핑시킬 경우, 명령어 코드의 길이와 피연산자의 바이트 길이가 가변적이므로 명령어가 다양한 길이의 픽셀에 매핑될 수 있으며, 하나의 명령어가 두 줄에 나뉘어 저장될 수도 있어 이미지에 다르게 나타나게 된다. 또한, 다른 의미의 명령어 코드와 피연산자가 같은 비트 패턴을 가질 수도 있다.

이에 대하여 각각의 명령어를 일정한 크기의 바이트에 인코딩하여 표현함으로써 이미지가 수행 코드의 명령어 특성을 명확하게 나타낼 수 있다[13]. 본 연구에서는 연속되는 3개의 바이트에 하나의 명령어를 저장하도록 하였으며, RGB와 흑백의 두 가지 이미지 생성 방법을 실험하였다. RGB 이미지의 경우 명령어 코드를 R(red) 과 G(green)요소 부분에 저장하였으며, B(blue) 요소에 피연산자를 인코딩하여 저장하였다. 흑백 이미지의 경우 연속되는 세 개의 픽셀이 같은 방식으로 사용되었다.

다양한 길이의 명령어를 3바이트에 저장하기 위한 방법은 다음과 같다. 실행 파일 명령어의 명령어 코드는 일반적으로 2바이트에서 6바이트 길이인데 이 중에서 주로 사용되는 2^{16}가지의 명령어 코드를 선정하고 이에 식별값을 부여하여 처음 2바이트에 저장하였다. 피연산자는 1바이트에 요약하여 저장하는 방법을 사용하였으며 Fig. 1.은 피연산자 부분의 저장 방법을 나타낸다. 피연산자의 종류를 상수(immediate value), 레지스터, 메모리로 구분하여 처음 세 비트는 각각의 형식의 피연산자가 명령어에 포함되었는지를 표시하며, 나머지 5비트는 상수

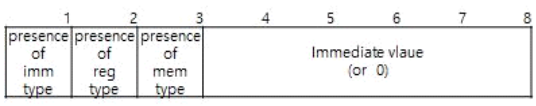


Fig. 1. Operands Encoding Method

가 사용되었을 경우 낮은 자리 비트를 저장하였다.

3.3 악성코드 검출을 위한 선형적 이미지 사용

프로그램의 코드는 여러 명령어의 연속으로 구성되어 있다. 악성코드 여부를 판별하기 위해서는 프로그램을 구성하는 명령어들의 분포와 함께 각 명령어들이 수행되는 순서의 특징을 파악하는 것이 중요하다. 따라서 이러한 순차적인 명령어의 수행 패턴을 이미지가 명확하게 표현하도록 이미지를 생성하는 것이 매우 중요하다. 관련 연구 사례[9, 10]에 의하면 실제 프로그램의 수행 흐름을 분석 이미지에 적절히 반영함으로써 악성코드 분류의 성능이 향상됨을 제시하였다. 또한, 연속적인 흐름 정보를 나타내기 위한 대표적인 방법으로 RNN(Recurrent Neural Network)이 있으나 악성코드 판별을 위한 실행 파일 바이트열의 길이가 매우 길어질 수 있으므로, RNN을 적용하는데 어려운 한계가 존재한다.

이에 관하여 본 연구에서는 명령어 열의 순차 정보를 최대한 이미지에 반영하기 위한 이미지 생성 방법을 제시하였다. 기존의 연구들이 실행 파일의 비트 패턴을 일반적인 직사각형 크기의 이미지에 반영했다면, 본 연구에서는 각 명령어 하나의 이진 코드를 이미지의 한 줄에 할당하여 위아래 방향의 긴 이미지를 생성함으로써 명령어의 흐름 정보가 이미지에 명확히 나타나도록 하였다. 기존의 방법은 이미지 생성을 위한 최종 바이너리 데이터를 왼쪽 상단부터 오른쪽 하단까지 256바이트와 같은 정해진 폭으로 채우는 방식으로 생성되어 명령어의 수행 흐름과는 무관한 위아래 줄 명령어 간의 연관성이 이미지 모양에 반영된다. 따라서 상하좌우 4방향으로 분석하는 2d-convolution 수행 시에 분석에 혼란을 줄 수 있는 단점이 있다. 즉 같은 명령어 흐름을 가진 코드들도 해당 코드의 시작 위치에 따라 상하 방향의 패턴은 같은 다른 형태가 될 수 있어 같은 종류의 악성코드인데도 다르게 분석될 가능성이 있다. 선형적인 이미지는 위아래 방향으로 이미지를 길고 얇게 생성하여 위의 단점들을 보완하고, 명령어의 흐름 정보 특성을 좀 더 직접적으로 나타낼 수 있다.

이미지를 생성에는 본 연구의 실험에서 악성코드 판별에 효과적이었던 .text 섹션과 .data 섹션을 사용하였다. 분석 대상 프로그램들을 구성하는 기계어 명령어들의 길이 분포가 4바이트 이하가 42%, 5~8바이트가 57%로 대부분 8바이트 이하인 것을 고려하여 이미지의 폭을 8바이트 설정하여 이미지를 생성하였다. Fig. 2는 선형적 이미지의 예를 보여주며, 8바이트보다 짧은 명령어는 0으로 채워져 이미지에서 오른쪽 부분이 검은색으로 나타난다.

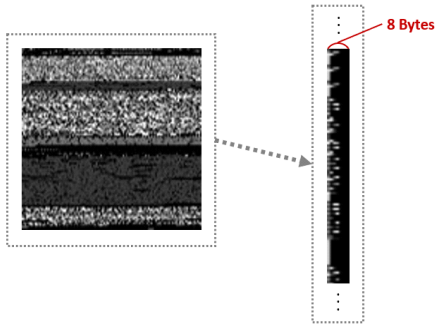


Fig. 2. Example of a Linear Binary Image

IV. 실 험

4.1 실험 환경 및 데이터 셋

실험을 위한 이미지 추출 및 학습 작업은 Ubuntu 18.04.5 LTS 운영체제에서 cuda version 11.2와 python 3.7.10 및 Google colab을 사용하였다.

학습 모델 구현 및 학습을 위해 'fastai' 파이썬 패키지를 사용하였으며 PE 파일의 구조 분석 및 주소 추출에는 'pefile' 패키지를 사용하였다. 또한 실행 파일의 역어셈블 및 명령어 코드 분류에는 'Capstone' 프레임워크를 사용하였다.

악성코드 및 정상 코드 판별을 위한 데이터 셋으로 한국인터넷진흥원 주최 2019 정보보호 R&D 데이터 챌린지의 데이터셋을 사용하였다. 이 데이터 집합은 약 4만개의 32/64bit PE 파일들로 구성되어 있으며 정상코드와 악성코드의 비율은 약 4:6이다. 본 연구에서는 전체 파일의 90%를 학습에 사용하였고, 10%를 테스트에 사용하였다.

4.2 이미지 생성에 따른 악성코드 판별 성능 분석

3장의 다양한 이미지 생성방법의 요소가 악성코드 검출에 끼치는 영향을 실험하였다. 성능 지표는 정밀도(precision), 재현율(recall), 정확도(accuracy), F1 점수(F1 score)를 사용하였다. 정밀도는 악성코드 보고를 하였을 경우 실제 악성코드일 확률을 나

타내며, 재현율은 악성코드를 악성코드로 보고할 확률을 나타낸다. 정확도는 악성코드 보고 또는 일반 코드 보고가 정확할 확률을 나타낸다.

$$\text{- 정밀도} = \frac{TruePositives}{TruePositives + FalsePositives}$$

$$\text{- 재현율} = \frac{TruePositives}{TruePositives + FalseNegatives}$$

$$\text{- 정확도} = \frac{TruePositives + TrueNegatives}{All}$$

$$\text{- F1 점수} = 2 \times \frac{\text{Precision} \times Recall}{\text{Precision} + Recall}$$

4.2.1 이미지 크기

Table 2는 실행 파일을 다양한 크기의 흑백 이미지로 생성하여 악성코드 검출의 성능을 분석한 결과이다. 이미지의 크기가 커질수록 전반적인 성능이 향상되었다. 256x512에서 256x256에 비해 대폭 향상된 정밀도 값을 나타냈으며 256x1024에서는 정밀도는 낮으나 가장 우수한 정확도와 F1 값을 얻을 수 있었다.

Table 2. Effect of Image Size

size	Preci.	Recall	Accu.	F1
256*256	0.9222	0.9401	0.9193	0.9311
256*512	0.9395	0.9290	0.9232	0.9342
256*1024	0.9378	0.9411	0.9290	0.9394

4.2.2 섹션별 데이터 판별 성능 분석

Table 3은 실행파일의 섹션별 바이너리 데이터가 악성코드 판별에 유용한 정도를 분석한 결과이다. 개별적으로 사용할 경우 .text 섹션과 .data 섹션이 악성코드 판별에 유의한 특징을 가지고 있는 것으로 파악되었으며, 상대적으로 .idata와 .edata 섹션을 사용한 경우는 판별 성능이 낮게 나타났다. 또한 .text와 .data 섹션을 함께 같은 비율만큼 포함시켜 생성한 이미지를 사용하였을 경우 가장 우수한 성능을 보였으며, .idata 및 .edata까지 포함시키면 오히려 성능이 떨어지는 것으로 나타났다.

Table 3. Effect of Considering Sections

	size	Preci.	Recall	Accu.	F1
.text	256*256	0.9334	0.9282	0.9300	0.9308
.data	256*256	0.9008	0.9166	0.8868	0.9086
.i/edata	256*256	0.7807	0.7721	0.7835	0.7764
.text + .data	256*256	0.9487	0.9210	0.9395	0.9346
	256*512	0.9548	0.9586	0.9511	0.9567
	256*1024	0.9601	0.9634	0.9570	0.9617
.text + .data + .i/edata	256*256	0.9484	0.9254	0.9366	0.9368
	256*512	0.9529	0.9508	0.9458	0.9518
	256*768	0.9553	0.9483	0.9455	0.9518
	256*1024	0.9520	0.9530	0.9464	0.9525

4.2.3 컬러 이미지 사용의 효과

Table 4는 컬러 이미지를 사용한 악성 코드 검출의 결과를 나타낸다. 첫 번째 실험에서는 .text, .data 섹션과 .idata 및 .edata 섹션을 시각적으로 구분하기 위하여 각각의 섹션에 대하여 RGB의 색깔 요소 중 각 하나씩만 사용하고, 나머지 요소는 0으로 패딩하였다. 즉, .text 섹션은 Red 색깔 요소로만 표시되며, .data 섹션은 Green 색깔 요소로 표시된다. 결과적으로 Table 3의 가장 마지막 행의 흑백 이미지와 동일한 양의 데이터를 포함하지만 컬러이미지를 사용하였으므로 파일 용량은 세 배가 되며, 각 섹션들이 빨강, 초록, 파랑으로 보여지게 된다. 실험 결과를 Table 3의 마지막 행과 비교하면, 같은 양의 정보를 포함하고, 각 섹션에 대한 색깔 구분을 하였음에도 불구하고 오히려 낮은 성능이 나타났는데, 이는 각 섹션에서 사용되지 않는 다른 두 색깔 요소를 0으로 패딩 한 것이 학습 과정에서 오히려 잡음으로 작용한 것으로 판단된다.

두 번째 실험에서는 단순히 연속적인 세 개의 바이트를 하나의 픽셀에 저장하는 방법으로 컬러 이미지를 생성하였으며, 결과적으로 같은 해상도를 기준으로 할 때 흑백 이미지보다 세 배의 데이터를 포함하게 되고 파일 크기도 세 배가 된다. 256x256 컬러 이미지를 사용한 성능을 보면 Table 3의 256x1024 해상도 흑백 이미지와 비슷하여 컬러 이미지 사용을 통한 유의한 효과는 발견되지 않았다. 256x512 컬러 이미지의 경우 오히려 일부 지표가 나빠졌으며, 이는 실행 파일의 크기가 384KB보다 작은 경우가 60% 정도 되어 부족한 부분에 대한 기

Table 4. Effect of using RGB Image

	size	Preci.	Recall	Accu.	F1
test 1	256*1024 (768K)	0.9364	0.9488	0.9335	0.9426
test 2	256*256 (192K)	0.9652	0.9501	0.9503	0.9576
	256*512 (384K)	0.9585	0.9528	0.9558	0.9556

- test 1: using different color for each section
- test 2: allocating 3 bytes to each pixel

본값 패딩이 분석에 악영향을 준 것으로 판단된다.

4.2.4 명령어 코드와 피연산자 정보 고려

역어셈블을 수행하여 명령어 코드와 피연산자 정보를 파악한 후 이를 이미지 생성에 활용한 방식은 코드의 의미를 이미지에 좀 더 직접적으로 반영하는 방법이라 할 수 있다. 이를 위하여 3.2.4의 방법대로 각 명령어의 명령어 코드와 피연산자 정보를 3바이트로 인코딩하고, 이를 컬러 또는 흑백 이미지로 생성하였으며, 추가로 흑백 이미지에 .data 섹션을 50% 포함하는 방법을 실험하였다. Table 5에 정리된 결과를 보면 예상과는 다르게 실제 성능은 오히려 단순 이미지 생성보다 전반적으로 낮은 성능을 보이고 있다. 명령어 구조를 반영한 이미지 생성은 결과적으로 악성코드의 특징적인 명령어의 순서를 좀 더 잘 반영하나 악성코드 검출의 효과가 적은 것으로 판단되었다. 즉, 이미지 인식의 관점에서 악성코드를 판별하는 경우 명령어 열을 2차원 이미지로 변환하고 이에 대하여 컨볼루션(convolution)을 수행하는 과정에서 실제 코드에서는 멀리 떨어진 상하 명령어 간의 관계가 이미지 인식에 반영되고, 명령어 코드 자체의 구조에 의한 정보도 희석됨으로써 명령어 순열 반영의 효과가 의도대로 활용되지 못한 것으로 분석되었다.

Table 5. Effect of applying OP code and operand information from disassembled code.

	size	Preci.	Recall	Accu.	F1
RGB .text	256*256	0.9395	0.8998	0.9075	0.9192
RGB .text + .data	256*256	0.9531	0.9231	0.9284	0.9379
gray .text	256*256	0.9209	0.8951	0.8974	0.9078
	256*768	0.9218	0.8922	0.8965	0.9068
gray .text + .data	256*256	0.9513	0.9174	0.9288	0.9358
	256*512	0.9547	0.9174	0.9288	0.9357

4.2.5 선형적 이미지의 사용

행 방향의 명령어의 순열 정보에 대하여 열 방향 명령어 간의 관계가 악성코드 인식의 잡음으로 작용하는 문제를 해결하기 위해 역어셈블한 실행 파일의 명령어들을 일차원적으로 나열한 선형적 이미지를 사용하여 악성코드의 판별 성능을 실험하였다.

이미지의 크기는 앞선 실험에서 가장 좋은 성능을 보인 Table 3의 256x1024 해상도 이미지를 기준으로 설정하였다. 해당 이미지의 .data 섹션 크기를 감안하여 8x16384 크기의 .data 부분을 사용하였으며, .text 부분은 8바이트 미만의 명령어의 경우 패딩이 채워지는 것을 감안하여, 8x8192에서 8x32768 크기까지 세 가지 이미지 크기를 사용하여 실험을 진행하였다.

또한, 선형적 이미지 생성을 3.2.4의 인코딩 기법에 적용하기 위하여 인코딩한 명령어 하나를 한 줄에 배치한 이미지를 사용한 실험도 진행하였다. 명령어 하나의 인코딩에 3바이트를 사용하므로 한 행의 마지막 바이트를 0으로 패딩하였으며, 4x16384와 4x32768 해상도의 이미지에 대하여 50%씩을 .text 섹션을 인코딩한 결과와 .data 섹션에 할당하였다.

Table 6의 실험 결과를 보면 전체적으로 정밀도에서 우수한 성능을 나타내고 있다. 선형적인 이미지의 경우 .text 섹션의 명령어 순열의 특징을 이미지 인식에서 더 잘 반영하게 되므로 이러한 부분이 정밀도의 향상에 영향을 준 것으로 판단된다. 그러나 재현률의 경우 중간 크기 이미지에서 가장 높았으며,

Table 6. Effect of applying linear image generation

	size	Preci.	Recall	Accu.	F1
.text+.data	8*8192+8*16384	0.9636	0.9556	0.9549	0.9595
	8*16384+8*16384	0.9640	0.9526	0.9536	0.9583
	8*32768+8*16384	0.9695	0.9163	0.9372	0.9422
(op + oper)+.data	4*8192+4*8192	0.9684	0.9594	0.9596	0.9638
	4*16384+4*16384	0.9755	0.9228	0.9448	0.9484

Table 3의 실험보다 상대적으로 낮은 성능을 보였다. 재현율과 정밀도는 일반적으로 상충되는 관계를 가지며(모든 코드를 악성코드로 판별할 경우 재현율은 100%가 된다), 테스트에서 사용한 악성, 정상 비율이 6:4로 악성코드가 더 많은 것을 고려했을 때, 이 비율을 일반코드가 대부분인 실제 상황으로 맞춘다면 제안된 방법의 정확도가 상대적으로 더 우수할 것으로 기대된다.

선형적 이미지 생성을 인코딩 기법에 적용할 경우에는 일반적 이미지 생성에서는 인코딩 기법이 효과가 없었던 것과 다르게 선형적 이미지 생성에서는 가장 높은 정밀도를 얻을 수 있었다. 이는 선형적 이미지 생성이 명령어 제어 흐름의 특징 인식에 유용함을 나타내는 것으로 분석된다.

V. 결 론

본 연구에서는 이미지 인식에 기반한 악성코드 검출 방법에서 이미지 생성 시 선택해야 하는 다양한 인자가 악성코드 검출의 성능에 주는 영향을 알아보기 위하여, 약 40000개의 악성 및 정상코드를 사용한 실험을 수행하고 이를 포괄적으로 분석한 결과를 제공하였다. 또한, 분석 결과를 기반으로 악성코드 검출에 적합한 선형적 이미지 생성방법을 제시하였다.

이미지 생성방법의 인자로는 이미지 크기, PE 구조에서의 섹션 구분, 컬러 이미지의 활용, 역어셈블 정보의 사용을 설정하고, 각 인자가 악성코드 검출의 성능에 주는 영향을 실험하였다. 실험 결과에 따르면 악성코드 검출은 해상도를 크게 할수록 포함 데이터가 많아져 일정 수준까지 성능이 향상되나 필요한 수준을 넘어서면 오히려 패딩 픽셀의 증가로 오탐이 증가할 수 있음을 보였으며, PE 실행파일 형식의 경우 .text와 .data의 두 섹션을 구분하여 함께 사용하는 것이 악성코드 판별에 효과적인 것으로 나타났다. 또한, 컬러 이미지의 사용은 악성코드 판별에 유의한 효과가 없었으며 역어셈블을 통해 명령어 코드와 피연산자를 일정 바이트에 인코딩하는 기법도 직사각형 형태의 이미지 생성에서는 효과가 작은 것으로 나타났다.

악성코드의 검출을 위해서는 지역적인 명령어 실행 패턴이나 특징적인 API 호출 순서 등의 파악이 중요하며 이러한 특성은 해당 기계어 코드 부분을 이미지로 표현하였을 경우 픽셀들의 연속적인 패턴으로 나타난다. 본 연구의 실험에 의하면 악성 코드 검출의 성능을 높이기 위해서는 이미지 생성 시에 이러한 패턴을 최대한 표현하면서 동시에 잡음 정보를 최소화하는 것이 매우 중요한 것으로 분석되었다.

이러한 분석 결과를 기반으로 명령어 흐름의 특징이 명확하게 나타나는 선형적 이미지 생성방법을 제시하였다. 제시된 방법은 악성코드 검출의 정밀도 향상에 효과적인 것으로 나타났으며, 일반적인 직사각형 이미지와 다르게 명령어 인코딩 기법을 적용할 경우에 정밀도 향상을 나타내어 선형적 이미지가 명령어의 흐름을 기반으로 악성코드를 판별하는 데 효과적임을 알 수 있었다.

향후 연구로는 목적 코드를 기본 블록 단위로 파악하고 그래프 형태의 제어 구조를 반영하여 좀 더 넓은 범위의 코드의 특성을 효과적으로 표현하는 이미지 생성방법의 연구가 필요한 것으로 판단된다. 아울러 PE 파일 형식이 아닌 실행 파일에 대한 효과적인 이미지 추출 방법 연구도 수행할 예정이다.

References

[1] Mohammed N. Alenezi, Haneen Alabdulrazzaq, Abdullah A. Alshaher and Mubarak M. Alkharang, "Evolution of Malware Threats and Techniques: A Review," International Journal of Communication Networks and Information Security, Vol. 12, No. 3, pp. 326-337, Dec. 2020

[2] L. Nataraj, S. Karthikeyan, G. Jacob and B.S. Manjunath, "Malware Images: Virualization and Automatic Classification," VizSec '11: Proceedings of the 8th International Symposium on Visualization for Cyber Security, no.4, pp. 1-7, July 2011

[3] Sunoh Choi, Sungwook Jang, Youngsoo Kim and Jonghyun Kim, "Malware detection using malware image and deep learning," 2017 International Conference on Information and Communication Technology Convergence (ICTC) (pp. 1193-1195), IEEE, Oct. 2017

[4] Bojan Kolosnjaji, Ambra Demontis, Battista Biggio, Davide Maiorca, Giorgio Giacinto, Claudia Eckert and Fabio Roli, "Adversarial malware binaries: Evading deep learning for malware detection in executables," 2018 26th European signal processing conference (EUSIPCO) (pp. 533-537), IEEE, Sept. 2018

[5] Seonhee Seok and Howon Kim, "Visualized Malware Classification Based on Convolutional Neural Network," Journal of The Korea Institute of Information Security and Cryptology, Vol. 26, No. 1, pp. 197-208, Feb. 2016

[6] Niket Bhodia, Pratikkumar Prajapati, Fabio Di Troia and Mark Stamp "Transfer learning for image-based malware classification," 5th International Conference on Information Systems Security and Privacy, pp. 719-726, Feb. 2019

[7] Ke He and Dong-Seong Kim. "Malware detection with malware images using deep learning techniques," 2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE) (pp. 95-102), IEEE, Aug. 2019

[8] Ke Zhang, "Malware Variants Classification for Section Contrast RGB Image using CNN," The master's thesis, Gachon University , Dec. 2020

[9] Liu Liu, and Baosheng Wang. "Malware classification using gray-scale images and ensemble learning," 2016 3rd International Conference on Systems and Informatics (ICSAI) (pp. 1018-1022). IEEE, Nov. 2016

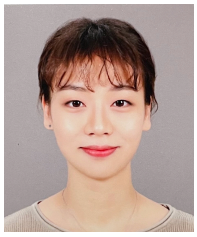
[10] Sang Ni, Quan Qian, and Rui Zhang. "Malware identification using visualization images and deep learning." Computers & Security vol. 77, pp. 871-885, Apr. 2018

[11] Jinpei Yan, Yong Qi, and Qifan Rao. "Detecting malware with an ensemble method based on deep neural network." Security and Communication Networks, vol. 2018, Mar. 2018

[12] Seongmin Jeong, Hyeonseok Kim, Youngjae Kim and Myungkeun Yoon, "V-gram: Malware Detection Using Opcode Basic Blocks and Deep Learning," Journal of KIISE vol. 46, no. 7, pp. 599-605, Jul. 2019

〈저 자 소 개〉

전 예 진 (YeJin Jeon) 학생회원
2022년 2월: 한국항공대학교 항공전자정보공학부 정보통신공학전공 졸업
〈관심분야〉 정보보호, 인공지능보안, 클라우드



김 진 이 (Jin-e Kim) 학생회원
2018년 3월~현재: 한국항공대학교 항공전자정보공학부 정보통신공학전공 학사과정
〈관심분야〉 정보보호, 인공지능보안, 컴퓨터비전

안 준 선 (Joonsen Ahn) 종신회원
1992년 2월: 서울대학교 계산통계학과 학사
1994년 2월: KAIST 전산학과 석사
2000년 8월: KAIST 전자전산학과 박사
2000년 8월~2001년 8월: KAIST 프로그램분석시스템연구단 연구원
2001년 9월~현재: 한국항공대학교 항공전자정보공학부 교수
〈관심분야〉 프로그래밍언어, 프로그램분석, 소프트웨어보안, 블록체인보안,