인공지능을 활용한
악성파일 탐지 시스템

팀 명 : 네카라쿠배
지도 교수 : 양환석 교수님
팀 장 : 고준성
팀 원 : 김영호
김지원
서정우
양경모

2021. 11.
중부대학교 정보보호학과

목 차

4. 결론

5. 별첨

1. 서론

1.1 연구 배경

ICT(Information and Communication Technologies)의 융합으로 이루어지는 차세대 산업혁명인 4차 산업혁명. AI(Artificial Intelligence), IoT(Internet of Things), 로봇 기술, 드론, 자율주행차, VR(Virtual Reality) 등의 여러 분야들이 4차 산업혁명을 주도하고 있다. '통신과 이종 간의 융합'으로 정의되는 4차 산업혁명이 실현되는 동안 통신이 적용되는 범위가 넓어지면서 그에 따른 사이버 공격 영역이 크게 확장되었다. 또한 비대면 사회로 디지털 접점이 증가하면서 기존 보안체계로 감당하기 어려운 신·변종 위협의 증가로 사이버보안 영역에 대한 인공지능 기술 도입 필요성의 인식들이 확대되고 있다. 4차 산업혁명의 핵심 기술이며 가장 중심에 놓여있는 AI(Artificial Intelligence)는 핵심적인 산업으로 부상되면서, 세계 주요국들과 많은 기업들이 인공지능에 대한 집중적인 투자와 연구개발이 이루어지고 있다. 다양한 분야에서 인공지능 기술이 연구 및 투자되는 만큼 정보보안 산업에서도 인공지능 기술이 접목되어 연구 개발이 진행 중에 있다. 이러한 트렌드에 맞춰서 다양해지고 정교해지는 사이버 공격 중 악성코드 탐지에 인공지능을 접목시켜서 효과적으로 대응하고자 한다.

IT >

KISA, "AI와 빅데이터 기술 활용해 사이버 위협 방어"

이원태 KISA 원장 기자간담회

박지영 기자

입력 2021.05.23 13:44

이원태 한국인터넷진흥원(KISA) 원장이 취임 후 처음으로 가진 기자간담회에서 사이버 위협 방어를 위해 인공지능(AI)과 빅데이터 기술을 활용해 예방 시스템을 구축하겠다는 계획을 밝혔다.

악성코드란 시스템에 접근하여 파괴하거나 정보를 유출하는 등의 악의적인 행위를 하는 프로그램을 말한다. 대표적으로 바이러스, 트로이목마, 웜 등으로 전자 메일과 같은 경로를 통해서 악성코드를 은닉하여 시스템을 공격한다. 현대 기술이 발전함에 따라서 악성코드의 기술도 정교해지고 다양해지고는 있다. 많은 악성코드 제작자들은 금전적인 이득을 취하는 목적으로 악성코드를 유포한다. 목적을 달성하기 위해서는 최소한의 비용과 노력으로 최대의 효과를 내야하기 때문에, 정교하게 제작하기보다는 비슷한 기능을 가지는 악성 프로그램을 빠르게 찍어 낸다. 또한, 정치적인 목적의 제작자 및 기술 과시를 위한 제작자의 악성코드는 시스템의 취약점 및 제로데이 공격을 위한 목적으로 정교하게 제작되어 탐지가 불가능한 경우도 존재한다.

1.2 연구 목적

이번 연구에서 빠르고 다양한 발전 속도에 따른 개인정보 및 금전적인 피해를 막기 위해 기존의 백신에 인공지능을 도입하여 스스로 학습하여 발전하는 백신에 대해 연구한다. 또한 이를 바탕으로 실제 악성파일과 정상파일을 이용해 모델링을 한 백신 프로그램을 개발해 신규 악성코드에 대한 대비책이 될 수 있는지에 대해 연구하고자 한다.

2. 관련 연구

2.1 Python

Python은 1991년 발표된 인터프리터 방식의 프로그래밍 언어다. 문법이 매우 쉬워서 작성하기 간단하기 때문에 초보자들이 처음 프로그래밍을 배울 때 추천되는 언어이다. 학습용으로 좋은 언어인 동시에 실사용률과 생산성도 높은 강력한 언어이다. 2000년에는 Python2, 2008년에는 Python3가 나왔다. 2010년 후반에 이르러서는 풍부한 머신러닝 라이브러리와 프레임워크로 머신러닝, 딥러닝 등 인공지능 개발과 데이터 분석을 위한 언어로서 각광 받고 있다. 현재 Python2는 2.7을 마지막으로 더 이상 업데이트를 하지 않고 python3은 3.10 버전까지 나왔다. 0.1 버전마다 for문 등의 루프 연산 효율이 2배 이상 빨라지는 개선이 되기 때문에 버전 업데이트가 필수다. 본 연구에서는 필요한 라이브러리 사용을 위해 Python2.7을 사용했다. Python2만 지원하는 라이브러리도 있고, Python3도 지원하는 라이브러리가 있기 때문에 개발을 할 때 본인이 필요한 라이브러리와 맞는 버전을 사용하면 된다.

2.2 PE

PE(Protable Executable) 형식은 윈도우 운영체제에서 사용되는 실행 파일, DLL 파일, Object 코드 등을 위한 파일 형식이다. PE 구조는 유닉스의 COFF(Common Object File Format)에서 파생되어 그 형식을 따른다. COFF OBJ 파일과도 공통적으로 많은 데이터 구조와 비슷하다. 'Portable Executable'이라는 용어는 지원되는 모든 CPU에서 모든 Windows 버전에 대해 공통 파일 형식으로 사용하여 다른 운영체제와도 이식성을 좋게 만든 파일 포맷이다. PE 형식을 조금만 수정하면 64비트 윈도우에서도 확장이 가능하다. EXE 파일과 DLL 파일 등도 같은 PE 형식을 사용한다. 이러한 PE 형식의 파일들은 동일한 PE header 구조를 가지고 있다. PE header는 파일이 메모리에 적재될 때의 각 정보가 메모리의 어느 부분에 위치하는지 등의 정보를 가지고 있다. 또한, PE header는 추출이 비교적 쉽고 간단하며, 프로그램 특성에 대한 많은 정보 획득이 가능하다.

악성코드 분석을 통해 파악이 가능한 정보는 제작에 사용된 컴파일러 버전과 종류, 날짜 정보, 동작에 필요한 기능 등 대부분의 정보는 PE header 내에 담겨있다. PE header는 PE 파일 포맷의 NT header와 DOS header, Section header로 구성되어있다. 해당 부분에는 메모리 적재 방식, 실행 시작 위치, 실행에 필요한 동적 라이브러리의 위치나 Stack/Heap의 메모리 크기에 대한 정보 등이 담겨있다.

2.3 Binary Images

딥러닝을 이용하여 악성코드를 탐지 및 분류하기 위해서 전처리를 통해 적절한 입력 데이터가 필요하다. 이때 악성코드를 이미지로 표현하는 대표적인 방법이 바이너리 이미지이다. 파일을 이미지화하여 악성코드 분류에 사용한 최초의 시도는 2011년 발표된 논문인 "Malware Images: Visualization and Automatic

Classification" 에서 소개되었다. 프로그램에서 이미지의 특징을 구별하는 것이 파일의 특징을 구별하는 것보다 어렵다. 이미지란 사람에게는 인지하는 형식이지만 컴퓨터에게는 의미 없는 연속된 바이트의 모음이기 때문이다. 그렇지만 최근 딥러닝의 많은 발전으로 이미지의 특성을 구분해내는 것이 어렵지 않은 일이 되고 있다.

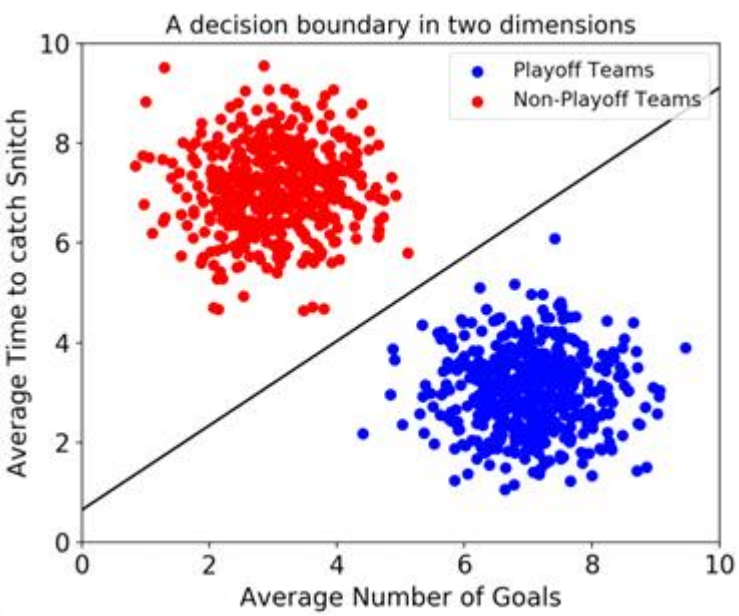
딥러닝에 이용 가능한 전처리 데이터를 만들기 위해 악성코드 바이너리에서 1바이트를 1픽셀로 변환하여 이미지 특정 정보들을 그레이스케일 이미지로 생성한다. 변환된 대부분의 이미지들은 육안으로 식별이 불가능하다. 이에 대해, 이미지 기반 딥러닝 알고리즘인 CNN을 시각화한 악성코드 분류기로 사용하여 육안으로 식별이 불가능한 이미지도 분류할 수 있도록 한다.

2.4 N-gram

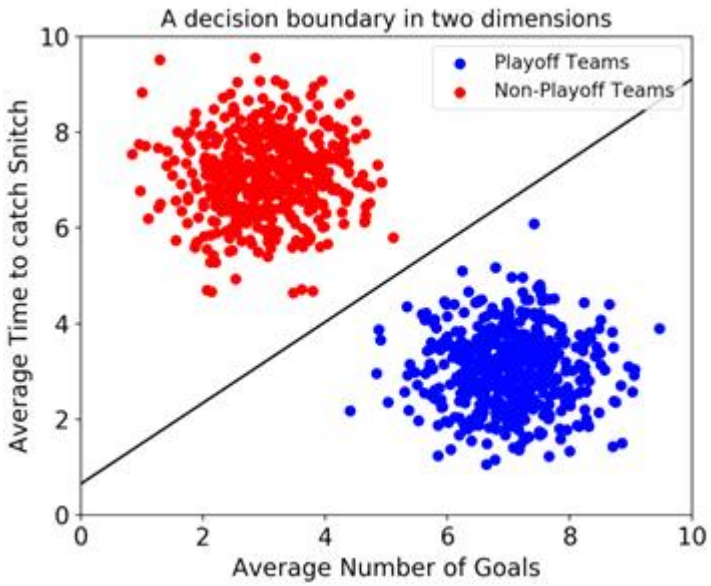
N-Gram 방식은 카운트에 기반을 두는 통계적 접근을 사용한다. 모든 단어를 고려하는 것이 아닌 일부의 단어를 고려한다. N-gram은 n개의 연속적인 단어 나열을 의미한다. N개의 단어로 이루어진 묶음들을 각각 하나의 토큰으로 본다. 본 연구에선 우선 프로그램의 Code Section 내용을 가져와 해당 내용을 어셈블리어로 변환했다. 어셈블리어 명령어를 4개씩 끊어 묶음을 만드는 4-Gram 방식을 사용했다. 묶음을 만든 후엔 각각의 빈도수를 카운트한다. 정상파일에서 자주 등장하는 묶음, 악성파일에서 자주 등장하는 묶음을 특징값으로 사용했다. 악성코드인지 판별을 원하는 파일을 입력받으면 해당 파일의 Code Section을 변환해 자주 등장하는 어셈블리 명령어를 확인해 악성파일일 확률을 계산한다.

2.5 SVM

SVM은 Support Vector Machine으로서 분류에 사용할 수 있는 강력한 머신러닝 모델이다. 간단하게 말하면 '분류를 위한 최적의 기준선을 정의하는 모델' 이다. 선을 결정하고 이후에 새로운 점이 나타나면 선을 기분으로 경계의 어느 쪽에 속하는지 분류할 수 있는 모델이다.



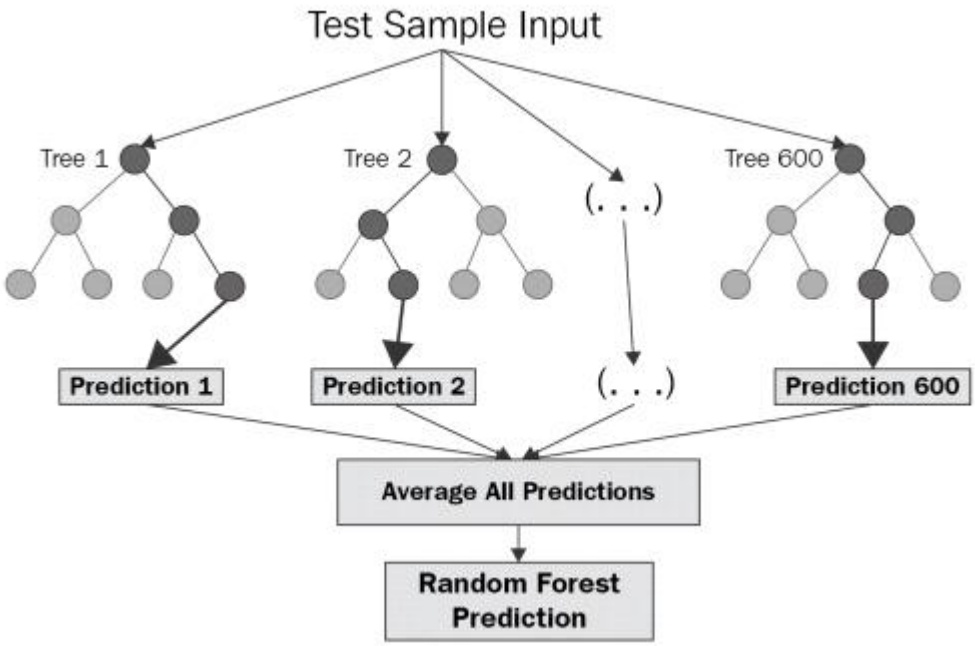
해당 그림처럼 경계선을 적절히 정하기 위한 모델이라고 생각하면 된다. Support Vector는 결정경계와 가장 가까이 있는 데이터 포인터를 의미한다. 마진이란 결정경계와 Support Vector 사이의 거리를 의미한다.



결정 경계가 데이터군으로부터 최대한 멀리 떨어졌을 때 그것을 최적의 경계라고 한다. 따라서 최적의 결정 경계는 마진을 최대화한다. SVM은 Support Vector만 잘 골라내면 나머지 쓸모없는 데이터 포인터들을 무시할 수 있어 속도가 빠르다는 장점이 있다.

2.6 Random Forest

랜덤 포레스트는 분류, 회귀, 분석 등에 사용되는 학습 방법으로, 훈련 과정에서 다수의 결정트리로부터 분류 또는 회귀분석을 하는 앙상블 방식을 사용한다. 여러 개의 결정트리를 만들고 새로운 데이터 포인트를 각 트리에 통과시킨다. 각 트리가 분류한 결과에서 투표를 실시해 가장 많은 득표를 한 결과를 분류 결과로 선택하게 된다. 일부 트리는 overfitting 될 수 있지만 많은 수의 트리를 생성함으로써 일부의 overfitting은 다수의 트리가 예측을 하는데 큰 영향을 미치지 못하도록 예방한다. 이렇게 다수의 트리를 사용하는 기법을 앙상블 기법이라고 부른다.



랜덤 포레스트에서 트리를 생성할 때 Bagging이라는 프로세스를 사용한다. 학습 데이터 셋에 1000개의 행이 있다고 생각했을 때 임의로 100개의 행을 선택에 결정 트리를 만드는 것을 Bagging이라고 한다. 이때 중복을 허용한다. 트리를 만들 때는 모든 feature에서 일부 선택하고 정보 획득량이 가장 높은 것을 기준으로 데이터를 분할하게 된다. 결국 분석에 사용되는 변수를 랜덤하게 추출하고 훈련 단계에서 훈련의 목적 함수를 최대로 만드는 매개변수 최적값을 도출하게 된다.

2.7 Naive Bayes

베이즈 정리를 이용한 확률적 기계학습 알고리즘이다. 대표적인 확률론적 분류 모델이며 특징들 사이에 독립성을 가정으로 하는 베이즈 정리를 토대로 데이터를 분류한다. 사전 확률에 기반을 두고 사후 확률을 추론하는 확률적 예측을 하는데, 이때 모든 사건이 독립사건이라는 순진한 가정을 하고 있다. 각 특징에 대한 확률값들은 베이즈 정리를 통해 최종 결과를 얻어낸다. 특징의 확률을 구하는 방법은 가우시안(Gaussian), 다항분포, 베르누이로 총 3가지의 방식이 존재한다. 특성들 사이

의 독립적 가정을 기반으로 텍스트 문서를 여러 범주로 분류하는 대표적인 분류기로 사용되고 있다. 나이브 베이즈 알고리즘은 학습 데이터의 양이 적은 상황에서도 좋은 성능을 보여준다는 최대 장점을 가지고 있다. 또한 지도학습 환경에서 매우 효율적으로 훈련될 수 있고, 적절한 전처리 과정이 있다면 서포트 벡터 머신이나 랜덤 포레스트와 같이 진보된 알고리즘에도 뒤지지 않는 성능을 보여준다. 이 밖에도 '차원의 저주(Cuse of Dimensionality)'라는 많은 특징을 사용할 때 발생하는 문제에 영향을 받지 않는다.

2.8 DNN

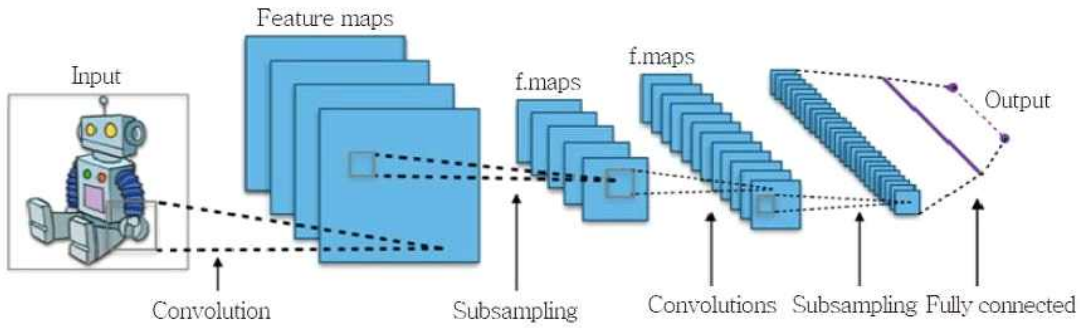
ANN(Artificial Neural Network) 기법의 여러 문제가 해결되면서 모델 내 은닉층을 많이 늘려서 학습의 결과를 향상시키는 방법이 등장했고, 이를 DNN(Deep Neural Network) 또는 심층 신경망이라고 한다. DNN은 자신의 각 레이어로부터 나온 특징들을 기존의 hand-crafted 알고리즘이 아닌 데이터를 통해 자동으로 학습하도록 하여 컴퓨터 비전, 자연어 처리 등 여러 분야에 우수한 성능을 보여준다. DNN은 입력층, 출력층 그리고 여러 개의 은닉층으로 구성되어있다. 각 노드는 이전 층의 노드와 특정한 가중치(Weight)를 가지고 연결되어 있다. DNN의 각 노드는 이전 층에서 주어지는 입력값과 각 연결의 가중치, 그리고 편향값(Bias)을 이용하여 출력값을 계산하며 이 과정에서 입력층에서 출력층으로 순차적으로 진행하여 최종적으로 DNN의 출력값을 계산한다. 컴퓨터가 스스로 분류레이블을 만들어 내고 공간을 왜곡하고 데이터를 구분 짓는 과정을 반복하여 최적의 구분선을 도출해낸다. DNN의 각 연결의 가중치와 편향값을 결정하는 과정은 소프트웨어 개발에서 핵심이라고 볼 수 있다. 이외에도 DNN은 많은 데이터와 반복학습, 사전학과 오류역전파 기법을 통해 많이 사용되고 있다.

2.9 CNN

CNN(Convolution Neural Network)은 다양한 분야에서 많이 활용되는 대표적인 딥러닝 알고리즘 중 하나이다. 복잡하고 많은 정보를 담고 있는 이미지에서 인간이 놓치기 쉬운 많은 부분을 특징화해준다. 이미지에서 자동으로 특징 추출이 가능하기 때문에 다양한 심층 신경망 모델 중 특히 이미지 분야에서 뛰어난 성능을 보인다. 자율 주행, 이미지와 영상처리 분야에서 적용되고 있다.

CNN은 이미지의 특징을 추출하고 이미지의 이동이나 왜곡에 대한 향상성을 유지하기 위한 과정을 거쳐 해당 이미지를 구분하기 위한 분류작업을 진행한다. 또한 이미지의 특징을 추출하기 위해 특정한 구조를 설계하여 이용한다. 특징을 추출 시 크게 추출하는 부분과 추출한 특징을 사용해 판단을 내리는 부분으로 나눈다. 특징을 추출하는 부분을 'Convolutional Layer'라고 부른다. Convolution 레이어는 다양한 필터를 사용해 이미지 전체를 스캔하듯이 사진을 찍는 방식을 사용한다. 필터를 이용하여 부분 스캔하듯이 사진을 찍어 그 사진들을 하나의 이미지로 간주한다. 그리고 또 다른 필터를 이용하여 사진을 찍어 나가는 방식으로 특징을 추출한다. 여러 개의 Convolutional 레이어를 거치게 되면 세밀한 부분까지 명확히 잡아내는 이

미지가 생성된다. 마지막으로 추출된 이미지의 특징으로 Fully Connected 레이어에서 분류 작업을 하게 된다.



3. 본론

3.1 시스템 구성

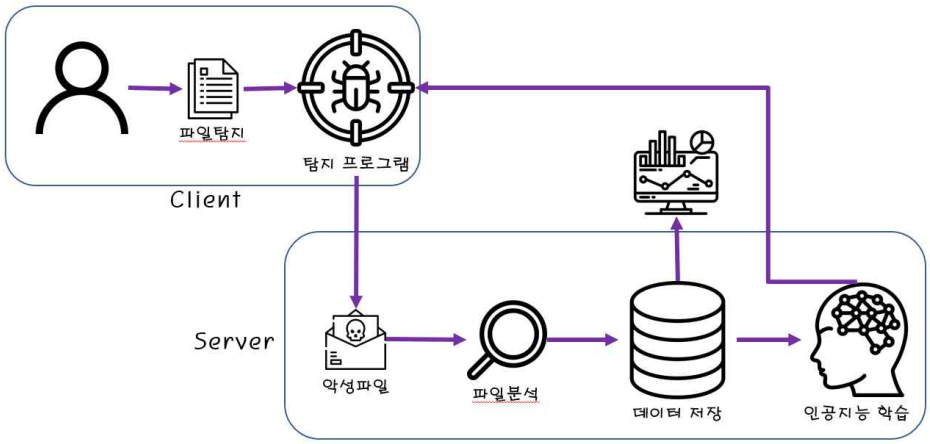


그림 6 시스템 구성도

3.2 데이터 셋

악성파일과 정상파일 데이터셋은 '정보보호산업진흥포털'에서 제공해주는 '정보보호 R&D 데이터셋'을 사용했다. 대용량 정상, 악성파일 1~5까지 대략 160,000개의 데이터를 신청해서 사용했다. 데이터셋은 각각 MD5로 이름이 지정되어 있고 함께 첨부된 csv 파일 안에 각 파일이 악성인지 정상인지 작성되어 있다.

악성코드 데이터셋	대용량 정상, 악성파일 1 (2017 예선)	한국인터넷진흥원, 하우리, 세인트시큐리티	"악성코드 탐지 2017" 예선에 활용된 15,000개의 Training Set
	대용량 정상, 악성파일 2 (2017 본선)	한국인터넷진흥원, 하우리, 세인트시큐리티	"악성코드 탐지 2017" 본선에 활용된 15,000개의 Test Set
	대용량 정상, 악성파일 3 (2018)	한국인터넷진흥원, 안랩, 이스트시큐리티, 하우리, 세인트시큐리티	"AI기반 악성코드 탐지 2018" 대회에 활용된 50,000개 정상, 악성코드
	대용량 정상, 악성파일 4 (2019)	한국인터넷진흥원, 안랩, 이스트시큐리티, 하우리, 세인트시큐리티	"AI기반 악성코드 탐지 2019" 대회에 활용된 40,000개 정상, 악성코드
	대용량 정상, 악성파일 5 (2020)	한국인터넷진흥원	"AI기반 악성코드 탐지 2020" 대회에 활용된 40,000개 정상, 악성코드

그림 7 정보보호산업진흥포털의 악성코드 데이터셋

3.3 특징 추출

3.3.1 PE 헤더

악성코드 분석을 통해 파악이 가능한 정보는 제작에 사용된 컴파일러 버전과 종류, 날짜 정보, 동작에 필요한 기능 등 대부분의 정보는 PE 헤더 내에 담겨있다. PE 파일에서 기능을 추출하기 위해서 Python의 pefile 라이브러리를 사용한 ClaMP의 오픈소스에서 도움을 받아서 특징을 추출했다. 제작자의 방법과 동일하게 PE 헤더에서 DOS 헤더 영역에서 6개, NT_HEADER에 속해있는 FILE 헤더에서 17개, OPTIONAL 헤더에서 37개의 총 60개의 RAW 특징과 필드 값을 한 번 더 가공한 Derived 특징 9개 총 69개의 특징을 추출하였다.

filename	MD5	e_cblp	e_cp	e_cparhdr	e_maxalloc	e_sp	e_lfanew	NumberOf	CreationYe	FH_char0	FH_char1	FH_char2	FH_char3	FH_char4	FH_char5	FH_char6
d7f8d89e1	d7f8d89e1	144	3	4	65535	184	128	3	1	0	1	0	0	0	0	0
d952e206l	d952e206l	0	1	2	17744	332	12	3	1	0	1	1	1	0	0	0
bce82de8	bce82de8	144	3	4	65535	184	200	4	1	1	1	0	0	0	0	0
13C9E872I	13c9e872f	144	3	4	65535	184	264	5	1	0	1	0	0	0	0	0
BE0E0BBA	be0e0bba	144	3	4	65535	184	216	5	0	0	1	0	0	0	0	0
9ebaf9f71	9ebaf9f71	80	2	4	65535	184	256	8	1	1	1	1	1	0	0	1
ebd2813e	ebd2813e	80	2	4	65535	184	256	8	1	0	1	1	1	0	0	1
1d6d1570	1d6d1570	80	2	4	65535	184	256	3	1	0	1	1	1	0	0	1
D7CD00A	d7cd00a1	80	2	4	65535	184	256	9	0	0	1	1	1	0	0	1
d39fdfe34	d39fdfe34	144	3	4	65535	184	128	5	1	1	1	1	0	0	0	0
5502856b	5502856b	144	3	4	65535	184	128	3	0	0	1	0	0	0	0	0
fe5ab4c19	fe5ab4c19	144	3	4	65535	184	128	4	1	0	1	0	0	0	0	0
89f3a3fb9	89f3a3fb9	144	3	4	65535	184	232	5	1	1	1	1	1	0	0	0
e37d3914	e37d3914	80	2	4	65535	184	256	3	1	1	1	1	1	0	0	1
8163C7A4	8163c7a4	80	2	4	65535	184	256	8	1	0	1	1	1	0	0	1
6eb468e4	6eb468e4	80	2	4	65535	184	512	8	0	0	1	1	1	0	0	0
f2cd4cd14	f2cd4cd14	144	3	4	65535	184	216	4	0	0	1	0	0	0	0	0
41f24c96d	41f24c96d	144	3	4	65535	184	264	5	0	0	1	0	0	0	0	0

그림 8 PE 전처리

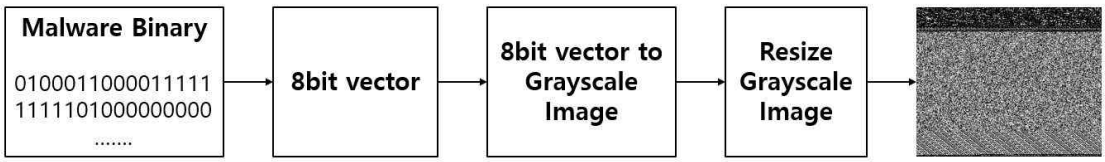
3.3.2 Binary Image

악성코드의 이진 파일을 이미지화로 시각화하여서 이미지 기반 딥러닝을 진행하였다. 이미지 기반 딥러닝 알고리즘인 CNN을 시각화한 악성코드 분류기로 사용하기 위하여 입력 데이터를 이미지화시키고 동일한 크기로 조정할 것이다. 이를 진행하기 위해서 악성코드를 분류하기 위해 전처리를 실행하여 해당 파일을 적절한 입력 데이터로 만들어준다.

입력 데이터는 1바이트를 1픽셀로 변환시키는 방법을 사용한다. 악성코드 파일의 바이너리를 8비트 부호 없는 정수의 벡터로 읽어 들인다. 각 바이트를 이미지의 한 픽셀로 변환하여 그레이스케일 이미지를 만든다. 파일 크기에 따라 달라지는 이미지의 너비와 높이를 CNN 분류기에 적용하기 위해서 256x256으로 동일하게 조정한다. 이미지의 크기 조정 방식은 파일 크기 width를 측정하고 한 변의 길이가 $\lceil \sqrt{\ln} \rceil$ 인 $\lceil \sqrt{\ln} \rceil * \lceil \sqrt{\ln} \rceil$ 의 사각형 이미지를 만든 후, 256x256 크기가 아닐 경우에만 사이즈를 조정하는 방식을 사용하여 진행하였다.

전처리 이후 [그림 10], [그림 11]과 같은 바이너리 이미지가 생성되었다. 같은 패밀리의 악성코드는 사용하는 라이브러리 등 유사한 특징을 가지고 있어 악성코드를 이미지로 변환했을 때 유사한 형태의 이미지가 생성된다. 변환된 이미지의 유사도

가 높을 경우에는 육안으로 분류 가능한 경우도 존재한다.



1바이트를 1픽셀로 변환하는 과정

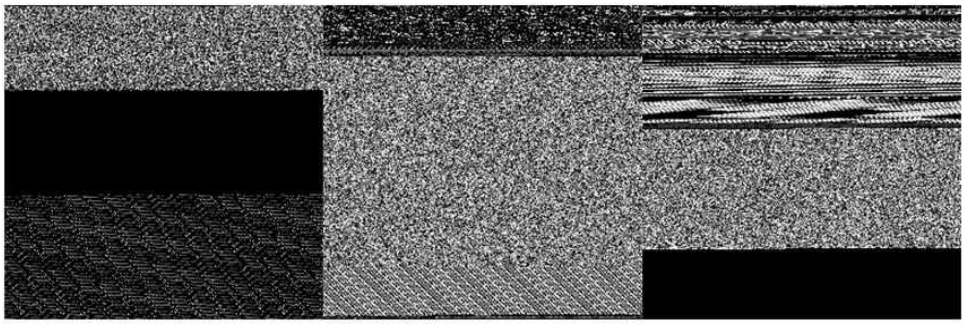


그림 10 바이너리 이미지로 변환된 악성파일

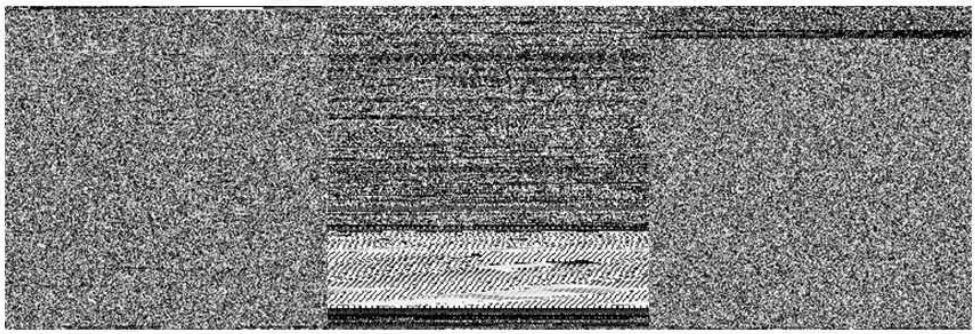


그림 11 바이너리 이미지로 변환된 정상파일

3.3.3 4-Gram

본 연구에서는 4-Gram 방식을 사용했다. Code Section을 어셈블리로 불러온 뒤 명령어를 4개씩 끊어 묶음을 만들었다. 각 묶음들의 빈도수를 확인한 것을 특징으로 사용했다. 예를 들어, 악성파일에는 [MOV MOV ADD MOV] 라는 명령어가 많이 나오고 정상 파일에는 [MOV SUB DIV MUL] 명령어가 많이 나온다고 가정해보자. 악성파일인지 검출하고 싶은 파일을 4-Gram을 통해 묶음을 만들어주고 각 명령어의 빈도수를 확인했을 때, [MOV MOV ADD MOV] 라는 명령어가 자주 등장한다면 악성파일일 확률이 높은 것이다. 단순히 하나의 묶음만 확인하는 것이 아니라 상위 100개의 명령어 묶음의 빈도수를 특징값으로 사용해 정확도를 높여줬다.

3.4 Anti Virus

본 연구에서는 PE Header에서 추출한 특징을 통해 악성코드인지 아닌지 예측하는 모델을 Kicom Anti Virus에 탑재했다.

Kicom Anti-Virus은 멀웨어를 탐지하고 치료할 수 있도록 설계된 오픈소스 바이러스 백신 엔진이다. 최초 버전은 C/C++로 작성되었으며 현재는 파이썬으로 새롭게 작성되어 누구나 쉽게 개발에 참여할 수 있다. 2018년 2월에는 OPSWAT 백신 엔진으로 채택되어 사용 중이다. 기본적으로 파일 및 폴더 검사, 로그 파일 생성, 악성파일 삭제, 바이러스 목록 출력 등의 다양한 기능이 있지만 본 연구에서는 백신의 기본이 되는 바이러스 탐지, 해당 바이러스 파일 삭제기능을 사용하였다.

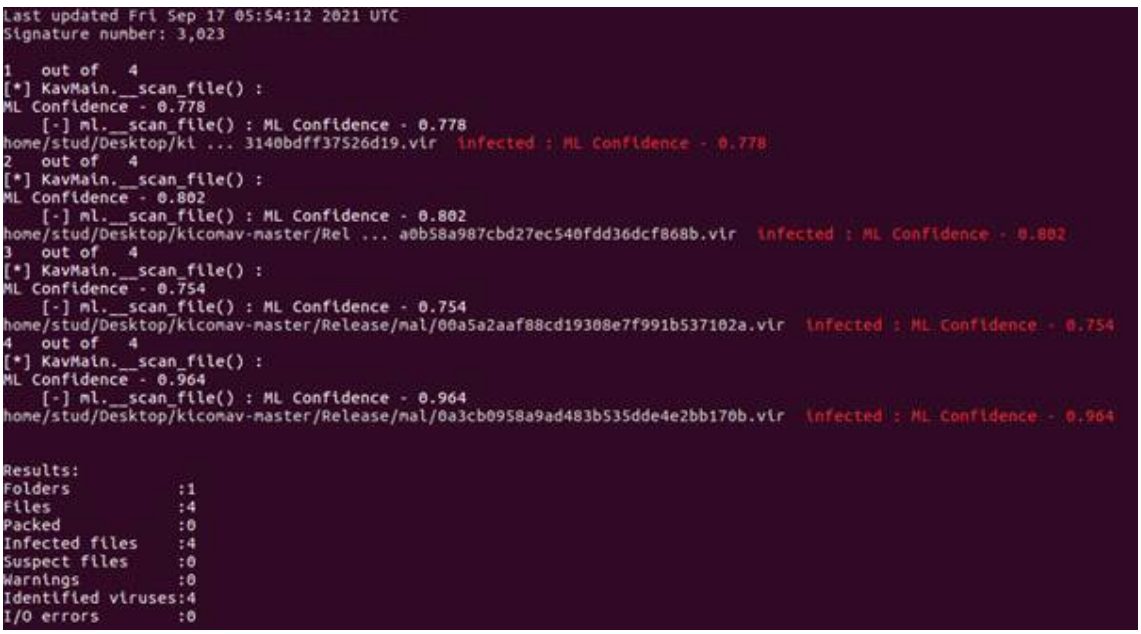


그림 13 백신에 ML 적용한 모습

악성파일 탐지 실제 동작 화면이다. 자신이 원하는 폴더 혹은 파일을 선택하면 직접 탑재한 모델 기반으로 악성파일 확률을 출력하고 악성파일인지 아닌지 탐지한다. 모든 파일에 대한 검사를 마치면 최종 결과에 대한 요약을 출력하고 종료한다.

악성파일 탐지 후 삭제 실제 동작 화면이다.

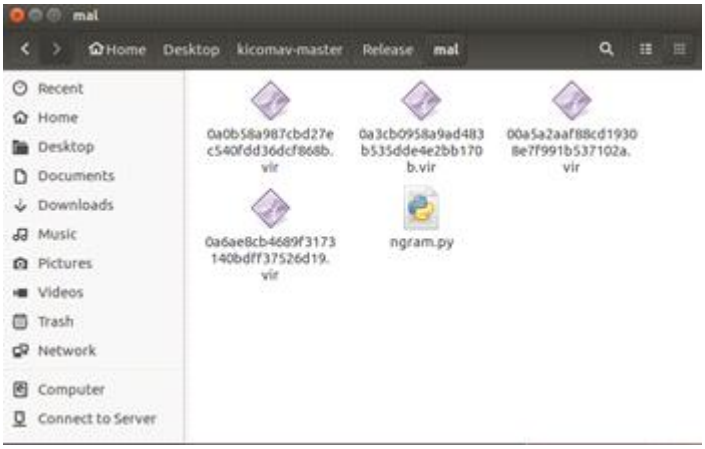


그림 14 악성파일 4개와 정상파일 1개

해당 폴더에서 악성파일 4개와 정상파일 1개를 넣어놓았다.

```
1   out of   5
[*] KavMain.__scan_file() :
ML Confidence - 0.778
    [-] ml.__scan_file() : ML Confidence - 0.778
home/stud/Desktop/kicomav-master/Release/mal/0a6ae8cb4689f3173140bdff37526d19.vir  infected : ML Confidence - 0.778
home/stud/Desktop/kicomav-master/Release/mal/0a6ae8cb4689f3173140bdff37526d19.vir  deleted
2   out of   5
[*] KavMain.__scan_file() :
ML Confidence - 0.802
    [-] ml.__scan_file() : ML Confidence - 0.802
home/stud/Desktop/kicomav-master/Release/mal/0a0b58a987cbd27ec540fdd36dcf868b.vir  infected : ML Confidence - 0.802
home/stud/Desktop/kicomav-master/Release/mal/0a0b58a987cbd27ec540fdd36dcf868b.vir  deleted
3   out of   5
[*] KavMain.__scan_file() :
4   out of   5
[*] KavMain.__scan_file() :
ML Confidence - 0.754
    [-] ml.__scan_file() : ML Confidence - 0.754
home/stud/Desktop/kicomav-master/Release/mal/00a5a2aaf88cd19308e7f991b537102a.vir  infected : ML Confidence - 0.754
home/stud/Desktop/kicomav-master/Release/mal/00a5a2aaf88cd19308e7f991b537102a.vir  deleted
5   out of   5
[*] KavMain.__scan_file() :
ML Confidence - 0.964
    [-] ml.__scan_file() : ML Confidence - 0.964
home/stud/Desktop/kicomav-master/Release/mal/0a3cb0958a9ad483b535dde4e2bb170b.vir  infected : ML Confidence - 0.964
home/stud/Desktop/kicomav-master/Release/mal/0a3cb0958a9ad483b535dde4e2bb170b.vir  deleted


Results:
Folders           :1
Files             :5
Packed            :0
Infected files    :4
Suspect files     :0
Warnings          :0
Identified viruses:4
Deleted files     :4
I/O errors        :0
```

그림 15 백신에서 악성4개 탐지 후 삭제된 모습

넣은 파일 디렉토리를 기준으로 프로그램 실행 후 악성파일임을 탐지하고 탐지율과 함께 악성파일임이 확인되면 자동으로 삭제가 된다.

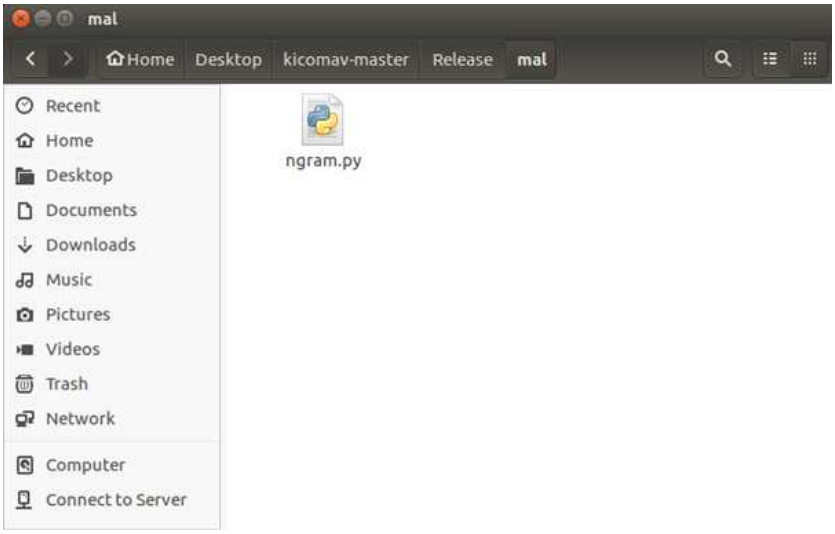


그림 16 악성파일 4개가 삭제된 모습

다시 디렉토리로 와서 확인해보면, 악성파일들이 삭제되고 정상파일 1개만 남은 것을 확인할 수 있다.

3.5 Client

GUI 제작에는 python 라이브러리인 tkinter을 사용했다. tkinter는 파이썬에서 GUI 프로그램을 할 때 사용하는 모듈이다. Python 설치 시 기본적으로 내장되어있는 파이썬 표준 라이브러리로 쉽고 간단하게 GUI 프로그램을 만들 수 있다는 장점이 있다.

본 연구에선 tkinter를 활용해 폴더를 선택, 악성파일 분석, 분석된 악성파일 삭제, 사용방법, 제작자를 보여주는 기능이 있는 GUI를 개발했다. 아래는 실제 동작 모습이다.

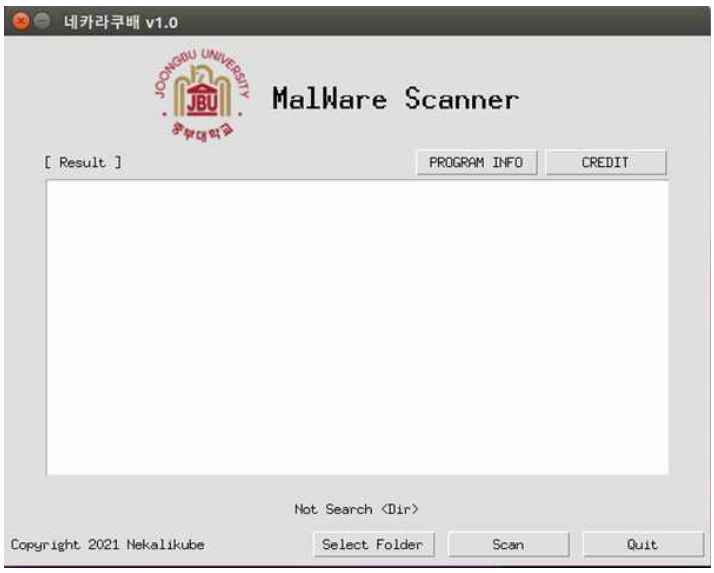


그림 17 프로그램 실행 화면

[버튼 설명]

Select Folder : 검사하고자 하는 파일이나 폴더를 선택
Scan : 선택된 파일이나 폴더에 대한 악성코드 검사
Quit : 프로그램 종료
Program Info : 프로그램에 대한 전반적인 정보와 사용법 기재
Credit : 제작자 정보
Select Folder 버튼 클릭 시 동작 모습이다. 검사하고자 하는 파일이나 폴더를 선택할 수 있는 윈도우가 나온다. 해당 윈도우에서 선택하면 된다.

그림 18 디렉토리 선택

파일 선택 완료 후 모습
Selected 아래에 현재 지정된 폴더명이 나타난다.

그림 19 디렉토리가 선택된 모습

Scan 버튼 클릭 시 동작 모습이다. 폴더 안의 파일들에 대한 검사를 진행하고 악성코드가 몇 개인지 정상파일이 몇 개인지 출력해준다.

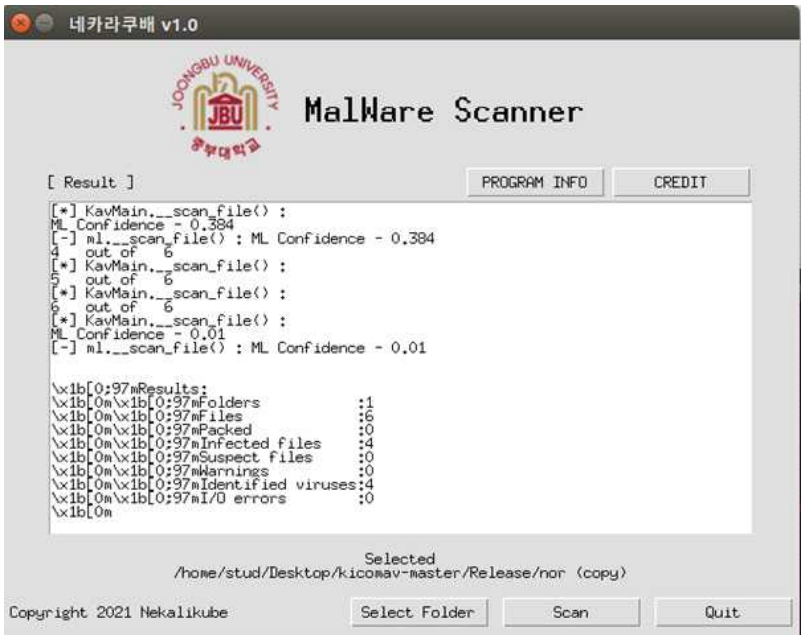


그림 20 Scan 동작 모습

검사 완료 후에는 Delete 여부를 물어보는데 이것은 악성파일로 판명된 파일들을 삭제할 것인지 질의하는 것이다. 만약 파일 삭제를 원한다면 Delete 버튼을 누르고 삭제를 원하지 않는다면 Cancle 버튼을 눌러 이전으로 돌아갈 수 있다.

그림 21 삭제 상호작용 화면

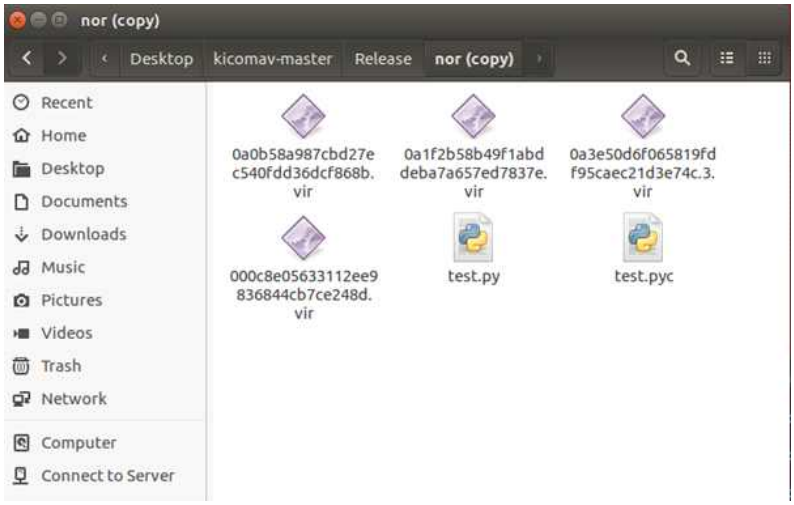


그림 22 정상파일 2개와 악성파일 4개

기존의 예시 파일 6개중 파이썬 2개는 정상 파일 나머지 4개는 악성 파일이다. 만약 Delete 버튼을 누른다면 정상 파일들은 제외하고 악성이라고 판단된 파일들이 삭제되는 것을 확인할 수 있다.

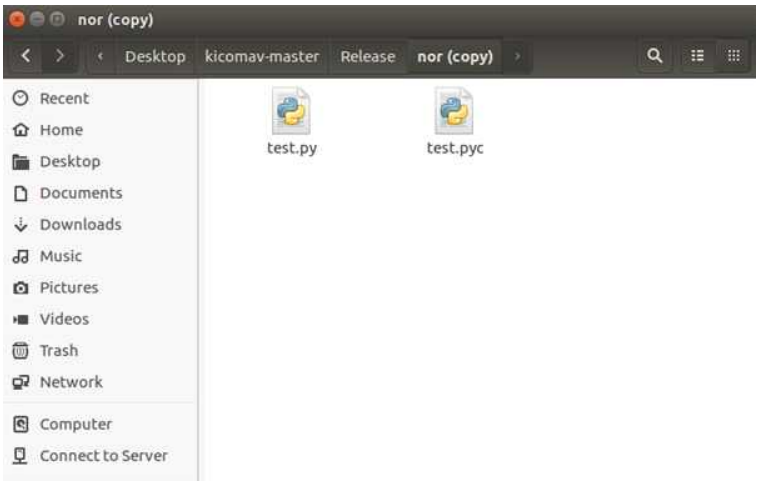


그림 23 정상파일만 남은 것을 확인

4. 결론

4.1 결론

본 프로젝트에서는 인공지능을 활용하여 오픈소스 백신 프로그램에 플러그인으로 모델을 올려 PE 구조의 파일들을 검사하고 악성파일을 잡아내었다. 바이너리 이미지와 CNN 알고리즘을 이용한 방법은 기술적인 문제와 이번 프로젝트에서의 정확도 문제로 제외하였다. N-gram에서 4-gram을 이용한 방법과 PE 정보를 이용한 방법을 비교하여 정확도를 확인하였을 때는, PE 정보를 가지고 하는 PE 정보만을 가지고 하는 방법, PE 정보와 Packer(패커)를 확인하여 두 가지 정보를 가지고 하는 방법, PE에서 상위 20가지 정보를 이용하여 4가지의 분류 알고리즘에 돌려서 정확도를 확인하였다.

	svm	randomforest	naivebayes	dnn	avg
pe	0.942222	0.987302	0.151111	0.061587	0.535556
pe_packer	0.942222	0.988571	0.151111	0.061587	0.535873
pe_top20	0.942222	0.986667	0.196825	0.061587	0.546825

위와 같이 결과가 나왔고, 패커와 PE 정보를 이용한 방법과 Random Forest 알고리즘을 사용한 정확도가 가장 높은 결과를 보여주었다. 이를 이용한 방법을 이용해서 모델로 사용하여 진행하였다. [3.4], [3.5]의 백신과 클라이언트에서 결과를 확인해 볼 수 있다. 이 모델을 백신에 플러그인으로 적용하여 탐지를 시도해보았고, 꽤 높은 탐지율로 악성파일을 잡아내는 결과를 도출해 내었다.

4.2 기대효과

악성코드는 기술 발전과 함께 다양해지고 정교한 형태로 발전하고 있다. 머신러닝 기반의 악성코드 탐지 기술은 대상 파일이 악성코드일 확률로 악성 여부를 정해준다는 특징이 있다. 따라서 위 기술은 아직 알려지지 않은 악성코드를 쉽게 잡아낼 수 있는 만큼 백신 및 프로그램에서 악성이 아닌 잡지 말아야 할 파일까지 탐지해내는 위험성이 존재하고 이를 감수해야 한다고 생각한다. 앞으로 많은 시도와 기술이 발전함에 따라서 더 나은 결과를 도출해 낼 수 있을 것이라 생각하고 보안에서의 인공지능과 같은 기술이 보다 더 안전한 기술 환경을 만들어 줄 수 있을 것이라 생각이 된다.

4.3 향후계획

시간 및 기술적인 문제로 기존에 구상했던 시스템 구성에서 서버 부분이 빠졌다. 현재 프로젝트에서의 백신을 이용해서 서버와 연동하고 모델을 재학습 시키는 과정을 진행하여 완성시켜볼 계획이다.

5. 별첨

5.1 소스코드

image.py

```
import numpy as np
import os, array
import scipy.misc
import glob
from PIL import Image

class IMAGE_feature():

    def __init__(self, in_path, out_path):

        self.in_path = in_path
        self.out_path = out_path

    def get_image(self, path, file):

        filename = path + file

        f = open(filename, 'rb')
        ln = os.path.getsize(filename)

        width = int(ln**0.5)
        rem = ln % width

        a = array.array("B")
        a.fromfile(f, ln-rem)
        f.close()

        g = np.reshape(a, (int(len(a)/width), width))
        g = np.uint8(g)

        fpng = self.out_path + file + ".png"
        scipy.misc.imsave(fpng, g)

        outfile = self.out_path + 'thumb/' + file + "_thumb.png"
        print(outfile)
        size = 256, 256
```

```
            img = Image.open(fpng)
            img_size = img.size

            if img_size != size:
                img_size = img.resize(size)
                img_size.save(outfile, "PNG")
            else:
                img.save(outfile, "PNG")

    def get_all(self):
            path = self.in_path

            for file in os.listdir(path):
                self.get_image(path, file)

def main():

    mal_path = '/PATH/200mal/'
    nor_path = '/PATH/100nor/'

    mal_out_path = '/PATH/200mal/'
    nor_out_path = '/PATH/100nor/'

    im1 = IMAGE_feature(mal_path, mal_out_path)
    im1.get_all()

    im2 = IMAGE_feature(nor_path, nor_out_path)
    im2.get_all()

if __name__ == '__main__':
    main()

print('COMPLETE')
```

image_set.py

```
import os
import glob
import sys
import numpy as np

from PIL import Image
```

```
from sklearn.model_selection import train_test_split
from keras.utils import np_utils

data_dir = '/PATH/dataset_image'

categories = ['normal', 'malware']
nb_classes = len(categories)

image_w = 28
image_h = 28
pixels = image_w * image_h * 1

X = []
Y = []

for idx, m in enumerate(categories):
    data_dir_detail = data_dir + "/" + m
    files = glob.glob(data_dir_detail + "/thumb/*thumb.png")

    for i, f in enumerate(files):
        try:
            img = Image.open(f)
            img = img.convert("L")
            img = img.resize((image_w, image_h))
            data = np.asarray(img)

            X.append(data)
            Y.append(idx)

            if i % 20 == 0:
                print(m, " : ", f)

        except:
            print(m, str(i) + "error")

X = np.array(X)
Y = np.array(Y)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3,
random_state = 0)

XY = (X_train, X_test, Y_train, Y_test)
```

```
np.save('/PATH/binary_image_data.npy', XY)
print('COMPLETE')
```

label_classify.py

```
import numpy
import pandas
import os
import shutil

file_root = 'C:\\PATH\\졸작\\label.csv'
dir_path ='C:\\PATH\\test_folder'

csv_data = pd.read_csv(file_root,header=None)
file_list = os.listdir(dir_path)

csv_name_list = set(csv_data[0])
for i in file_list:
    now = i[:-4]
    if now in csv_name_list:
        # i 에 이름값 num 에 label값
        num = int(csv_data[csv_data[0]==now][1].values)
    if num == 1:
        move_src = 'C:\\PATH\\malware\\'
        shutil.move(dir_path+'\\'+i,move_src+i)
    elif num == 0:
        move_src = 'C:\\PATH\\normal\\'
        shutil.move(dir_path+'\\'+i,move_src+i)

print('work it')
```

ngram.py

```
import os
import pefile
import pydasm
import operator
import csv
import hashlib

from itertools import chain
from capstone import *
from capstone.x86 import *
```

```
class NGRAM_FEATURE:

    def __init__(self,output_file):
        self.output_file = output_file
        self.grams = dict()

    def get_asm(self,mode,file):
        asm = []
        pe = pefile.PE(file)
        bytes = []

        ep = pe.OPTIONAL_HEADER.AddressOfEntryPoint
        end = pe.OPTIONAL_HEADER.SizeOfCode

        for section in pe.sections:
            addr = section.VirtualAddress
            size = section.Misc_VirtualSize

            if ep > addr and ep < (addr+size):
                ep = addr
                end = size
        data = pe.get_memory_mapped_image()[ep:ep+end]
        offset = 0

        temp = data.encode('hex')
        temp = [temp[i:i+2] for i in range(0,len(temp),2)] #slice 2

        if mode:
            return temp

        md = Cs(CS_ARCH_X86,CS_MODE_32)
        md.detail = False

        for i in md.disasm(data,0x401000):
            asm.append(i.mnemonic)
        return asm

    def gen_list_n_gram(self,n,asm_code):
        for i in range(0,len(asm_code),n):
            yield asm_code[i:i+n]
```

```
    def n_grams(self,n,asm_code,check):
        if check == 1:
            grams = self.grams
        elif check == 0:
            grams = dict() #make new dict

        gen_list = self.gen_list_n_gram(n,asm_code)

        for i in gen_list:
            i = ' '.join(i)
            try:
                grams[i] += 1
            except:
                grams[i] = 1

        return grams

    def write_csv_header(self,csv_header):
        file_path = self.output_file
        HASH = ['filename','MD5']
        label = ['label']
        header = HASH+csv_header+label

        file = open(file_path,'wa')
        write = csv.writer(file,delimiter = ',')
        write.writerow(header)
        file.close()

    def count_n_gram(self,grams,header,label):
        grams_count = list()

        for asm in header:
            try:
                grams_count.append(grams[asm])
            except:
                grams_count.append(0)
        grams_count.append(label)

        return grams_count

    def calc_file_hash(self,file_path):
        f = open(file_path,'rb')
```

```
        data = f.read()
        hash = hashlib.md5(data).hexdigest()
        return hash

    def write_csv_data(self,data):
        file_path = self.output_file
        file = open(file_path,'a')
        write = csv.writer(file,delimiter=',')
        write.writerow(data)
        file.close()
```

model.py

```
# 4가지 알고리즘에 대한 코드

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
import tensorflow as tf

import numpy as np
import pandas as pd


class Classifiers():

        def __init__(self, X, Y):

                self.x_train, self.x_test, self.y_train, self.y_test = \
                train_test_split(X, Y, test_size=0.2, random_state=0)


        def do_svm(self):
                clf = SVC()
                clf.fit(self.x_train, self.y_train)
                y_pred = clf.predict(self.x_test)

                return accuracy_score(self.y_test, y_pred)
```

```
def do_randomforest(self, mode):

        clf = RandomForestClassifier()
        clf.fit(self.x_train, self.y_train)

        if mode == 1:
                return clf.feature_importances_
        y_pred = clf.predict(self.x_test)

        return accuracy_score(self.y_test, y_pred)

def do_naivebayes(self):
        clf = GaussianNB()
        clf.fit(self.x_train, self.y_train)
        y_pred = clf.predict(self.x_test)

        return accuracy_score(self.y_test, y_pred)


def do_dnn(self):

        if "Series" in str(type(self.y_train)):
                self.y_train = self.y_train.to_frame()
                self.y_test = self.y_test.to_frame()
                input_len = len(self.x_train.columns)
        else:
                self.y_train = self.y_train.reshape(len(self.y_train), 1)
                self.y_test = self.y_test.reshape(len(self.y_test), 1)
                input_len = np.size(self.x_train, 1)

        learning_rate = 0.001
        batch_size = 128
        training_epochs = 50
        keep_prob = 0.7

        x_train = self.x_train
        y_train = self.y_train

        X = tf.placeholder(tf.float32, [None, input_len])
        Y = tf.placeholder(tf.float32, [None, 1])
```

```
                W1 = tf.Variable(tf.random_normal([input_len, 128]),
name='weight1')
                b1 = tf.Variable(tf.truncated_normal([128]), name='bias1')
                L1 = tf.sigmoid(tf.matmul(X, W1) + b1)

                W2 = tf.Variable(tf.random_normal([128, 1024]), name='weight2')
                b2 = tf.Variable(tf.truncated_normal([1024]), name='bias2')
                L2 = tf.sigmoid(tf.matmul(L1, W2) + b2)
                L2 = tf.nn.dropout(L2, keep_prob)

                W3 = tf.Variable(tf.random_normal([1024, 1024]), name='weight3')
                b3 = tf.Variable(tf.truncated_normal([1024]), name='bias3')
                L3 = tf.sigmoid(tf.matmul(L2, W3) + b3)
                L3 = tf.nn.dropout(L3, keep_prob)

                W4 = tf.Variable(tf.random_normal([1024, 128]), name='weight4')
                b4 = tf.Variable(tf.truncated_normal([128]), name='bias4')
                L4 = tf.sigmoid(tf.matmul(L3, W4) + b4)

                W5 = tf.Variable(tf.random_normal([128, 1]), name='weight5')
                b5 = tf.Variable(tf.truncated_normal([1]), name='bias5')

                output = tf.sigmoid(tf.add(tf.matmul(L4, W5), b5))

                cost = -tf.reduce_mean(Y * tf.log(output) + (1 - Y) * tf.log(1 -
output))
                train =
tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

                predicted = tf.cast(output > 0.5, dtype=tf.float32)
                accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y),
dtype=tf.float32))

                with tf.Session() as sess:
                        sess.run(tf.global_variables_initializer())

                        for epoch in range(training_epochs):
                                avg_cost = 0
                                total_batch = int(len(x_train) / batch_size)

                                for i in range(total_batch-1):
                                        batch_xs =
```

```
x_train[i*batch_size:(i+1)*batch_size]
                                        batch_ys                                        =
y_train[i*batch_size:(i+1)*batch_size]

                                        _ , c =sess.run([train, cost], feed_dict={X:
batch_xs, Y: batch_ys})

                        acc =   sess.run(accuracy, feed_dict={X: self.x_test, Y:
self.y_test})

                return acc


        def do_all(self):
                rns = []

                rns.append(self.do_svm())
                rns.append(self.do_randomforest(0))
                rns.append(self.do_naivebayes())
                rns.append(self.do_dnn())

                return rns
```

pe_raw_feature.py

```
import csv
import os
import pefile
import math
import hashlib



IMAGE_DOS_HEADER                                                                =
['e_magic','e_cblp','e_cp','e_crlc','e_cparhdr','e_minalloc','e_maxalloc','e_ss','e_sp'
,'e_csum',

'e_ip','e_cs','e_lfarlc','e_ovno','e_res','e_oemid','e_oeminfo','e_res2','e_lfanew']



F       I       L       E       _       H       E       A       D       E       R       =
["Machine","NumberOfSections","TimeDateStamp","PointerToSymbolTable",
```

```
                "NumberOfSymbols","SizeOfOptionalHeader","Characteristics"]

OPTIONAL_HEADER                                                                 =
["Magic","MajorLinkerVersion","MinorLinkerVersion","SizeOfCode","SizeOfInitializedD
ata",\
"SizeOfUninitializedData","AddressOfEntryPoint",\
"BaseOfCode","BaseOfData","ImageBase","SectionAlignment","FileAlignment",\
"MajorOperatingSystemVersion","MinorOperatingSystemVersion",\
"MajorImageVersion",\
"MinorImageVersion",\
"MajorSubsystemVersion",\
"MinorSubsystemVersion",\
"SizeOfImage",\
"SizeOfHeaders",\
"CheckSum",\
"Subsystem",\
"DllCharacteristics",\
"SizeOfStackReserve",\
"SizeOfStackCommit",\
"SizeOfHeapReserve",\
"SizeOfHeapCommit",\
"LoaderFlags",\
"NumberOfRvaAndSizes"]



def extract_image_dos_header(pe):
    IMAGE_DOS_HEADER_data = [0 for i in range(19)]
    try: #Dos_header 안의 내용 호출
        IMAGE_DOS_HEADER_data = [
            pe.DOS_HEADER.e_magic,
            pe.DOS_HEADER.e_cblp,
            pe.DOS_HEADER.e_cp,
            pe.DOS_HEADER.e_crlc,
            pe.DOS_HEADER.e_cparhdr,
            pe.DOS_HEADER.e_minalloc,
            pe.DOS_HEADER.e_maxalloc,
            pe.DOS_HEADER.e_ss,
            pe.DOS_HEADER.e_sp,
            pe.DOS_HEADER.e_csum,
            pe.DOS_HEADER.e_ip,
            pe.DOS_HEADER.e_cs,
```

```
            pe.DOS_HEADER.e_lfarlc,
            pe.DOS_HEADER.e_ovno,
            pe.DOS_HEADER.e_res,
            pe.DOS_HEADER.e_oemid,
            pe.DOS_HEADER.e_oeminfo,
            pe.DOS_HEADER.e_res2,
            pe.DOS_HEADER.e_lfanew]
    except Exception , e:
        print e
    return IMAGE_DOS_HEADER_data


def extract_file_header(pe):
    FILE_HEADER_data = [0 for i in range(7)]
    try:
        FILE_HEADER_data=[
            pe.FILE_HEADER.Machine,
            pe.FILE_HEADER.TimeDateStamp,
            pe.FILE_HEADER.PointerToSymbolTable,
            pe.FILE_HEADER.NumberOfSymbols,
            pe.FILE_HEADER.SizeOfOptionalHeader,
            pe.FILE_HEADER.Characteristics]
    except Exception,e:
        print e
    return FILE_HEADER_data



def extract_optional_header(pe):
    OPTIONAL_HEADER_data = [0 for i in range(29)]

    try:
        OPTIONAL_HEADER_data = [pe.OPTIONAL_HEADER.Magic,
                pe.OPTIONAL_HEADER.MajorLinkerVersion,
                pe.OPTIONAL_HEADER.MinorLinkerVersion,
                pe.OPTIONAL_HEADER.SizeOfCode,
                pe.OPTIONAL_HEADER.SizeOfInitializedData,
                pe.OPTIONAL_HEADER.SizeOfUninitializedData,
                pe.OPTIONAL_HEADER.AddressOfEntryPoint,
                pe.OPTIONAL_HEADER.BaseOfCode,
                pe.OPTIONAL_HEADER.BaseOfData,
                pe.OPTIONAL_HEADER.ImageBase,
```

```
                    pe.OPTIONAL_HEADER.SectionAlignment,
                    pe.OPTIONAL_HEADER.FileAlignment,
                    pe.OPTIONAL_HEADER.MajorOperatingSystemVersion,
                    pe.OPTIONAL_HEADER.MinorOperatingSystemVersion,
                    pe.OPTIONAL_HEADER.MajorImageVersion,
                    pe.OPTIONAL_HEADER.MinorImageVersion,
                    pe.OPTIONAL_HEADER.MajorSubsystemVersion,
                    pe.OPTIONAL_HEADER.MinorSubsystemVersion,
                    pe.OPTIONAL_HEADER.SizeOfImage,
                    pe.OPTIONAL_HEADER.SizeOfHeaders,
                    pe.OPTIONAL_HEADER.CheckSum,
                    pe.OPTIONAL_HEADER.Subsystem,
                    pe.OPTIONAL_HEADER.DllCharacteristics,
                    pe.OPTIONAL_HEADER.SizeOfStackReserve,
                    pe.OPTIONAL_HEADER.SizeOfStackCommit,
                    pe.OPTIONAL_HEADER.SizeOfHeapReserve,
                    pe.OPTIONAL_HEADER.SizeOfHeapCommit,
                    pe.OPTIONAL_HEADER.LoaderFlags,
                    pe.OPTIONAL_HEADER.NumberOfRvaAndSizes]
    except Exception,e:
        print
    return OPTIONAL_HEADER_data



def extract_features(pe):
    IMAGE_DOS_HEADER_data = extract_image_dos_header(pe)
    FILE_HEADER_data = extract_file_header(pe)
    OPTIONAL_HEADER_data = extract_optional_header(pe)
    return IMAGE_DOS_HEADER_data + FILE_HEADER_data +
OPTIONAL_HEADER_data




def main():
    source_path = r'C:\PATH\malware'
    output_file = r'C:\PATH\result.csv'
    label = ['clean']
    f = open(output_file,'wt')
    writer = csv.writer(f)
```

```
writer.writerow(IMAGE_DOS_HEADER+FILE_HEADER+OPTIONAL_HEADER+['label'])


    for subdir,dirs,files in os.walk(source_path):
        for file in files:
            input_file = source_path + '/' + file
            try:
                pe = pefile.PE(input_file)
            except Exception, e:
                print "Exception while loading file : " ,e
            else:
                try:
                    features = extract_features(pe)
                    writer.writerow(features+label)
                except Exception,e:
                    print "Exception while opening and write CSV file : " , e
    f.close()
    print 'It work !!'

main()
```

check_packer_pe_feature.py

```
import csv,os,pefile
import yara
import math
import hashlib

class pe_features():

    IMAGE_DOS_HEADER = [
                        "e_cblp",\
                        "e_cp", \
                        "e_cparhdr",\
                        "e_maxalloc",\
                        "e_sp",\
                        "e_lfanew"]

    FILE_HEADER= ["NumberOfSections","CreationYear"] + [ "FH_char" + str(i) for
i in range(15)]
```

```
    OPTIONAL_HEADER1 = [
                        "MajorLinkerVersion",\
                        "MinorLinkerVersion",\
                        "SizeOfCode",\
                        "SizeOfInitializedData",\
                        "SizeOfUninitializedData",\
                        "AddressOfEntryPoint",\
                        "BaseOfCode",\
                        "BaseOfData",\
                        "ImageBase",\
                        "SectionAlignment",\
                        "FileAlignment",\
                        "MajorOperatingSystemVersion",\
                        "MinorOperatingSystemVersion",\
                        "MajorImageVersion",\
                        "MinorImageVersion",\
                        "MajorSubsystemVersion",\
                        "MinorSubsystemVersion",\
                        "SizeOfImage",\
                        "SizeOfHeaders",\
                        "CheckSum",\
                        "Subsystem"]
    OPTIONAL_HEADER_DLL_char = [ "OH_DLLchar" + str(i) for i in range(11)]


    OPTIONAL_HEADER2 = [
                        "SizeOfStackReserve",\
                        "SizeOfStackCommit",\
                        "SizeOfHeapReserve",\
                        "SizeOfHeapCommit",\
                        "LoaderFlags"]  # boolean check for zero or not
    OPTIONAL_HEADER = OPTIONAL_HEADER1 + OPTIONAL_HEADER_DLL_char
+ OPTIONAL_HEADER2
    Derived_header = ["sus_sections","non_sus_sections",
"packer","packer_type","E_text","E_data","filesize","E_file","fileinfo"]

    def __init__(self,source,output,label):
        self.source = source
        self.output = output
        self.type = label
        #Need PEiD rules compile with yara
        self.rules= yara.compile(filepath='./peid.yara')
```

```
    def file_creation_year(self,seconds):
        tmp = 1970 + ((int(seconds) / 86400) / 365)
        return int(tmp in range (1980,2016))

    def FILE_HEADER_Char_boolean_set(self,pe):
        tmp = [pe.FILE_HEADER.IMAGE_FILE_RELOCS_STRIPPED,\
            pe.FILE_HEADER.IMAGE_FILE_EXECUTABLE_IMAGE,\
            pe.FILE_HEADER.IMAGE_FILE_LINE_NUMS_STRIPPED,\
            pe.FILE_HEADER.IMAGE_FILE_LOCAL_SYMS_STRIPPED,\
            pe.FILE_HEADER.IMAGE_FILE_AGGRESIVE_WS_TRIM,\
            pe.FILE_HEADER.IMAGE_FILE_LARGE_ADDRESS_AWARE,\
            pe.FILE_HEADER.IMAGE_FILE_BYTES_REVERSED_LO,\
            pe.FILE_HEADER.IMAGE_FILE_32BIT_MACHINE,\
            pe.FILE_HEADER.IMAGE_FILE_DEBUG_STRIPPED,\
            pe.FILE_HEADER.IMAGE_FILE_REMOVABLE_RUN_FROM_SWAP,\
            pe.FILE_HEADER.IMAGE_FILE_NET_RUN_FROM_SWAP,\
            pe.FILE_HEADER.IMAGE_FILE_SYSTEM,\
            pe.FILE_HEADER.IMAGE_FILE_DLL,\
            pe.FILE_HEADER.IMAGE_FILE_UP_SYSTEM_ONLY,\
            pe.FILE_HEADER.IMAGE_FILE_BYTES_REVERSED_HI
            ]
        return [int(s) for s in tmp]

    def OPTIONAL_HEADER_DLLChar(self,pe):
        tmp = [

pe.OPTIONAL_HEADER.IMAGE_DLLCHARACTERISTICS_DYNAMIC_BASE,\

pe.OPTIONAL_HEADER.IMAGE_DLLCHARACTERISTICS_FORCE_INTEGRITY,\
            pe.OPTIONAL_HEADER.IMAGE_DLLCHARACTERISTICS_NX_COMPAT ,\

pe.OPTIONAL_HEADER.IMAGE_DLLCHARACTERISTICS_NO_ISOLATION,\
            pe.OPTIONAL_HEADER.IMAGE_DLLCHARACTERISTICS_NO_SEH,\
            pe.OPTIONAL_HEADER.IMAGE_DLLCHARACTERISTICS_NO_BIND,\
            pe.OPTIONAL_HEADER.IMAGE_DLLCHARACTERISTICS_WDM_DRIVER,\

pe.OPTIONAL_HEADER.IMAGE_DLLCHARACTERISTICS_TERMINAL_SERVER_AWARE
,\

pe.OPTIONAL_HEADER.IMAGE_DLLCHARACTERISTICS_HIGH_ENTROPY_VA,\
```

```
pe.OPTIONAL_HEADER.IMAGE_DLLCHARACTERISTICS_APPCONTAINER,\
            pe.OPTIONAL_HEADER.IMAGE_DLLCHARACTERISTICS_GUARD_CF
            ]
        return [int(s) for s in tmp]

    def Optional_header_ImageBase(self,ImageBase):
        result= 0
        if ImageBase % (64 * 1024) == 0 and ImageBase in
[268435456,65536,4194304]:
            result = 1
        return result

    def Optional_header_SectionAlignment(self,SectionAlignment,FileAlignment):
        """This is boolean function and will return 0 or 1 based on condidtions
        that it SectionAlignment must be greater than or equal to FileAlignment
        """
        return int(SectionAlignment >= FileAlignment)

    def Optional_header_FileAlignment(self,SectionAlignment,FileAlignment):
        result =0
        if SectionAlignment >= 512:
            if FileAlignment % 2 == 0 and FileAlignment in range(512,65537):
                result =1
        else:
            if FileAlignment == SectionAlignment:
                result = 1
        return result

    def Optional_header_SizeOfImage(self,SizeOfImage,SectionAlignment):

        return int(SizeOfImage % SectionAlignment == 0)

    def Optional_header_SizeOfHeaders(self,SizeOfHeaders,FileAlignment):

        return int(SizeOfHeaders % FileAlignment == 0 )

    def extract_dos_header(self,pe):
        IMAGE_DOS_HEADER_data = [ 0 for i in range(6)]
        try:
            IMAGE_DOS_HEADER_data = [
                                pe.DOS_HEADER.e_cblp,\
                                pe.DOS_HEADER.e_cp, \
```

```
                                pe.DOS_HEADER.e_cparhdr,\
                                pe.DOS_HEADER.e_maxalloc,\
                                pe.DOS_HEADER.e_sp,\
                                pe.DOS_HEADER.e_lfanew]
        except Exception, e:
            print e
        return IMAGE_DOS_HEADER_data

    def extract_file_header(self,pe):
        FILE_HEADER_data = [ 0 for i in range(3)]
        FILE_HEADER_char =  []
        try:
            FILE_HEADER_data = [
                    pe.FILE_HEADER.NumberOfSections, \
                    self.file_creation_year(pe.FILE_HEADER.TimeDateStamp)]
            FILE_HEADER_char = self.FILE_HEADER_Char_boolean_set(pe)
        except Exception, e:
            print e
        return FILE_HEADER_data + FILE_HEADER_char

    def extract_optional_header(self,pe):
        OPTIONAL_HEADER_data = [ 0 for i in range(21)]
        DLL_char =[]
        OPTIONAL_HEADER_data2 = [ 0 for i in range(6)]

        try:
            OPTIONAL_HEADER_data = [
                pe.OPTIONAL_HEADER.MajorLinkerVersion,\
                pe.OPTIONAL_HEADER.MinorLinkerVersion,\
                pe.OPTIONAL_HEADER.SizeOfCode,\
                pe.OPTIONAL_HEADER.SizeOfInitializedData,\
                pe.OPTIONAL_HEADER.SizeOfUninitializedData,\
                pe.OPTIONAL_HEADER.AddressOfEntryPoint,\
                pe.OPTIONAL_HEADER.BaseOfCode,\
                pe.OPTIONAL_HEADER.BaseOfData,\
                #Check the ImageBase for the condition

self.Optional_header_ImageBase(pe.OPTIONAL_HEADER.ImageBase),\
                # Checking for SectionAlignment condition

self.Optional_header_SectionAlignment(pe.OPTIONAL_HEADER.SectionAlignment,pe
.OPTIONAL_HEADER.FileAlignment),\
```

```
                    #Checking for FileAlignment condition

self.Optional_header_FileAlignment(pe.OPTIONAL_HEADER.SectionAlignment,pe.OP
TIONAL_HEADER.FileAlignment),\
                    pe.OPTIONAL_HEADER.MajorOperatingSystemVersion,\
                    pe.OPTIONAL_HEADER.MinorOperatingSystemVersion,\
                    pe.OPTIONAL_HEADER.MajorImageVersion,\
                    pe.OPTIONAL_HEADER.MinorImageVersion,\
                    pe.OPTIONAL_HEADER.MajorSubsystemVersion,\
                    pe.OPTIONAL_HEADER.MinorSubsystemVersion,\
                    #Checking size of Image

self.Optional_header_SizeOfImage(pe.OPTIONAL_HEADER.SizeOfImage,pe.OPTIONA
L_HEADER.SectionAlignment),\
                    #Checking for size of headers

self.Optional_header_SizeOfHeaders(pe.OPTIONAL_HEADER.SizeOfHeaders,pe.OPTI
ONAL_HEADER.FileAlignment),\
                    pe.OPTIONAL_HEADER.CheckSum,\
                    pe.OPTIONAL_HEADER.Subsystem]

                DLL_char = self.OPTIONAL_HEADER_DLLChar(pe)

                OPTIONAL_HEADER_data2= [
                    pe.OPTIONAL_HEADER.SizeOfStackReserve,\
                    pe.OPTIONAL_HEADER.SizeOfStackCommit,\
                    pe.OPTIONAL_HEADER.SizeOfHeapReserve,\
                    pe.OPTIONAL_HEADER.SizeOfHeapCommit,\
                    int(pe.OPTIONAL_HEADER.LoaderFlags == 0) ]
            except Exception, e:
                print e
            return OPTIONAL_HEADER_data + DLL_char + OPTIONAL_HEADER_data2

        def get_count_suspicious_sections(self,pe):
            result=[]
            tmp =[]
            benign_sections                                                    =
set(['.text','.data','.rdata','.idata','.edata','.rsrc','.bss','.crt','.tls'])
            for section in pe.sections:
                tmp.append(section.Name.split('\x00')[0])
            non_sus_sections = len(set(tmp).intersection(benign_sections))
            result=[len(tmp) - non_sus_sections, non_sus_sections]
```

```
        return result

    def check_packer(self,filepath):

        result=[]
        matches = self.rules.match(filepath)

        try:
            if matches == [] or matches == {}:
                result.append([0,"NoPacker"])
            else:
                result.append([1,matches['main'][0]['rule']])
        except:
            result.append([1,matches[0]])

        return result

    def get_text_data_entropy(self,pe):
        result=[0.0,0.0]
        for section in pe.sections:
            s_name = section.Name.split('\x00')[0]
            if s_name == ".text":
                result[0]= section.get_entropy()
            elif s_name == ".data":
                result[1]= section.get_entropy()
            else:
                pass
        return result

    def get_file_bytes_size(self,filepath):
        f = open(filepath, "rb")
        byteArr = map(ord, f.read())
        f.close()
        fileSize = len(byteArr)
        return byteArr,fileSize

    def cal_byteFrequency(self,byteArr,fileSize):
        freqList = []
        for b in range(256):
            ctr = 0
            for byte in byteArr:
                if byte == b:
```

```
                    ctr += 1
            freqList.append(float(ctr) / fileSize)
        return freqList

    def get_file_entropy(self,filepath):
        byteArr, fileSize = self.get_file_bytes_size(filepath)
        freqList = self.cal_byteFrequency(byteArr,fileSize)
        # Shannon entropy
        ent = 0.0
        for freq in freqList:
            if freq > 0:
                ent +=  - freq * math.log(freq, 2)

            #ent = -ent
        return [fileSize,ent]

    def get_fileinfo(self,pe):
        result=[]
        try:
            FileVersion     = pe.FileInfo[0].StringTable[0].entries['FileVersion']
            ProductVersion                                                    =
pe.FileInfo[0].StringTable[0].entries['ProductVersion']
            ProductName =     pe.FileInfo[0].StringTable[0].entries['ProductName']
            CompanyName = pe.FileInfo[0].StringTable[0].entries['CompanyName']
        #getting Lower and
            FileVersionLS     = pe.VS_FIXEDFILEINFO.FileVersionLS
            FileVersionMS     = pe.VS_FIXEDFILEINFO.FileVersionMS
            ProductVersionLS = pe.VS_FIXEDFILEINFO.ProductVersionLS
            ProductVersionMS = pe.VS_FIXEDFILEINFO.ProductVersionMS
        except Exception, e:
            result=["error"]
        #print "{} while opening {}".format(e,filepath)
        else:
        #shifting byte
            FileVersion = (FileVersionMS >> 16, FileVersionMS & 0xFFFF,
FileVersionLS >> 16, FileVersionLS & 0xFFFF)
            ProductVersion = (ProductVersionMS >> 16, ProductVersionMS &
0xFFFF, ProductVersionLS >> 16, ProductVersionLS & 0xFFFF)
            result = [FileVersion,ProductVersion,ProductName,CompanyName]
        return int ( result[0] != 'error')

    def write_csv_header(self):
```

```
        filepath = self.output
        HASH = ['filename', 'MD5']
        header = HASH + self.IMAGE_DOS_HEADER + self.FILE_HEADER +
self.OPTIONAL_HEADER + self.Derived_header
        header.append("class")
        csv_file= open(filepath,"wa")
        writer = csv.writer(csv_file, delimiter=',')
        writer.writerow(header)
        csv_file.close()

    def extract_all(self,filepath):
        data =[]
        #load given file
        try:
            pe = pefile.PE(filepath)
        except Exception, e:
            print "{} while opening {}".format(e,filepath)
        else:
            data += self.extract_dos_header(pe)
            data += self.extract_file_header(pe)
            data += self.extract_optional_header(pe)
            # derived features
            #number of suspicisou sections and non-suspicsious section
            num_ss_nss = self.get_count_suspicious_sections(pe)
            data += num_ss_nss
            # check for packer and packer type
            packer = self.check_packer(filepath)

            # Appending the packer info to the rest of features
            data += packer[0]
            entropy_sections = self.get_text_data_entropy(pe)
            data += entropy_sections
            f_size_entropy = self.get_file_entropy(filepath)
            data += f_size_entropy
            fileinfo = self.get_fileinfo(pe)
            data.append(fileinfo)
            data.append(self.type)

        return data

    def write_csv_data(self,data):
        filepath = self.output
```

```
        csv_file= open(filepath,"a")
        writer = csv.writer(csv_file, delimiter=',')
        writer.writerow(data)
        csv_file.close()

    def getMD5(self, filepath):
        with open(filepath, 'rb') as fh:
            m = hashlib.md5()
            while True:
                data = fh.read(8192)
                if not data:
                    break
                m.update(data)
            return m.hexdigest()

    def create_dataset(self):
        self.write_csv_header()
        count = 0

        #run through all file of source and extract features
        for file in os.listdir(self.source):
                filepath = self.source + file
                data = self.extract_all(filepath)
                hash_ = self.getMD5(filepath)
                print "hash: ", hash_
                data.insert(0, hash_)
                data.insert(0, file)

                self.write_csv_data(data)
                count += 1
                print "Successfully Data extracted and written for {}.".format(file)
                print "Processed " + str(count) + " files"

def main():

    source_path= raw_input("Enter the path of samples (ending with /) >>  ")
    output_file= raw_input("Give file name of output file. (.csv) >>")
    label = raw_input("Enter type of sample( malware(1)|benign(0))>>")

    features = pe_features(source_path,output_file,label)
    features.create_dataset()
```

```
if __name__ == '__main__':
    main()
```

pe(python2) -> pe_cython.py

```
BYTE = ctypes.c_ubyte
WORD = ctypes.c_ushort
DWORD = ctypes.c_uint
FLOAT = ctypes.c_float
LPBYTE = ctypes.POINTER(ctypes.c_ubyte)
LPTSTR = ctypes.POINTER(ctypes.c_char)
HANDLE = ctypes.c_void_p
PVOID = ctypes.c_void_p
LPVOID = ctypes.c_void_p
UINT_PTR = ctypes.c_uint
SIZE_T = ctypes.c_uint


class DOS_HEADER(ctypes.Structure):
    _pack_ = 1
    _fields_ = [
        ('e_magic', WORD),
        ('e_cblp', WORD),
        ('e_cp', WORD),
        ('e_crlc', WORD),
        ('e_cparhdr', WORD),
        ('e_minalloc', WORD),
        ('e_maxalloc', WORD),
        ('e_ss', WORD),
        ('e_sp', WORD),
        ('e_csum', WORD),
        ('e_ip', WORD),
        ('e_cs', WORD),
        ('e_lfarlc', WORD),
        ('e_ovno', WORD),
        ('e_res', BYTE * 8),  # 8Byte
        ('e_oemid', WORD),
        ('e_oeminfo', WORD),
        ('e_res2', BYTE * 20),  # 20Byte
        ('e_lfanew', DWORD),
    ]
```

```
class FILE_HEADER(ctypes.Structure):
    _pack_ = 1
    _fields_ = [
        ('Machine', WORD),
        ('NumberOfSections', WORD),
        ('CreationYear', DWORD),
        ('PointerToSymbolTable', DWORD),
        ('NumberOfSymbols', DWORD),
        ('SizeOfOptionalHeader', WORD),
        ('Characteristics', WORD),
    ]


class OPTIONAL_HEADER(ctypes.Structure):
    _pack_ = 1
    _fields_ = [
        ('Magic', WORD),
        ('MajorLinkerVersion', BYTE),
        ('MinorLinkerVersion', BYTE),
        ('SizeOfCode', DWORD),
        ('SizeOfInitializedData', DWORD),
        ('SizeOfUninitializedData', DWORD),
        ('AddressOfEntryPoint', DWORD),
        ('BaseOfCode', DWORD),
        ('BaseOfData', DWORD),
        ('ImageBase', DWORD),
        ('SectionAlignment', DWORD),
        ('FileAlignment', DWORD),
        ('MajorOperatingSystemVersion', WORD),
        ('MinorOperatingSystemVersion', WORD),
        ('MajorImageVersion', WORD),
        ('MinorImageVersion', WORD),
        ('MajorSubsystemVersion', WORD),
        ('MinorSubsystemVersion', WORD),
        ('Reserved1', DWORD),
        ('SizeOfImage', DWORD),
        ('SizeOfHeaders', DWORD),
        ('CheckSum', DWORD),
        ('Subsystem', WORD),
        ('DllCharacteristics', WORD),
        ('SizeOfStackReserve', DWORD),
        ('SizeOfStackCommit', DWORD),
```

```
        ('SizeOfHeapReserve', DWORD),
        ('SizeOfHeapCommit', DWORD),
        ('LoaderFlags', DWORD),
        ('NumberOfRvaAndSizes', DWORD),
    ]


class DATA_DIRECTORY(ctypes.Structure):
    _pack_ = 1
    _fields_ = [
        ('VirtualAddress', DWORD),
        ('Size', DWORD),
    ]


class SECTION_HEADER(ctypes.Structure):
    _pack_ = 1
    _fields_ = [
        ('Name', BYTE * 8),
        ('Misc_VirtualSize', DWORD),
        ('VirtualAddress', DWORD),
        ('SizeOfRawData', DWORD),
        ('PointerToRawData', DWORD),
        ('PointerToRelocations', DWORD),
        ('PointerToLinenumbers', DWORD),
        ('NumberOfRelocations', WORD),
        ('NumberOfLinenumbers', WORD),
        ('Characteristics', DWORD),
    ]


def enum(*sequential, **named):
    enums = dict(zip(sequential, range(len(sequential))), **named)
    reverse = dict((value, key) for key, value in enums.iteritems())
    enums['reverse_mapping'] = reverse
    return type('Enum', (), enums)

image_directory_entry = enum('EXPORT', 'IMPORT', 'RESOURCE', 'EXCEPTION',
'SECURITY',
                             'BASERELOC', 'DEBUG',
                             'COPYRIGHT',  # Architecture on non-x86 platforms
                             'GLOBALPTR',         'TLS',         'LOAD_CONFIG',
```

```
'BOUND_IMPORT',
                                'IAT',      'DELAY_IMPORT',      'COM_DESCRIPTOR',
'RESERVED')


p_str = re.compile(r'[^\x00]*')  # NULL 문자 직전까지 복사


class PE:
    def __init__(self, mm, verbose, filename):
        self.filename = filename
        self.filesize = os.path.getsize(filename)
        self.verbose = verbose
        self.mm = mm
        self.sections = []  # 모든 섹션 정보 담을 리스트
        self.data_directories = []  # 모든 데이타 디렉토리 정보를 담을 리스트
        self.pe_file_align = 0
```

in k2.py

```
 # 진단/치료 가능한 악성코드 수 출력
    num_sig = format(kav.get_signum(), ',')
    msg = 'Signature number: %s\n\n' % num_sig
    cprint(msg, FOREGROUND_GREY)

    log_print('# Signature number: %s\n' % num_sig)
    log_print('#\n\n')  # 로그 파일 헤더 끝...

    if options.opt_vlist is True:  # 악성코드 리스트 출력
        kav.listvirus(listvirus_callback)
    else:
        if args:
            kav.set_result()   # 악성코드 검사 결과를 초기화

            # 검사 시작 시간 체크
            start_time = datetime.datetime.now()

            # 검사용 Path (다중 경로 지원)
            for scan_path in args:  # 옵션을 제외한 첫번째가 검사 대상
                scan_path = os.path.abspath(scan_path)

                if os.path.exists(scan_path):  # 폴더 혹은 파일가 존재하는가?
```

```
                    kav.scan(scan_path, scan_callback, disinfect_callback,
update_callback, quarantine_callback)
                else:
                    # 출력되지 못한 결과물을 출력한다.
                    print_display_scan_result(None, None, None)

                    print_error('Invalid path: \'%s\'' % scan_path)

            # 검사 종료 시간 체크
            end_time = datetime.datetime.now()

            global g_delta_time
            g_delta_time = end_time - start_time

            # 출력되지 못한 결과물을 출력한다.
            print_display_scan_result(None, None, None)

            # 악성코드 검사 결과 출력
            ret = kav.get_result()
            print_result(ret)

    #kav.uninit()

if __name__ == '__main__':
    main()
```

gui.py

```
#-*-coding:utf-8-*-

import os
import threading
import time
import subprocess
import sys
reload(sys)
sys.setdefaultencoding('utf-8')

import tkinter.ttk as ttk
import tkinter.font as tkFont

from tkinter import filedialog
from tkinter import *
```

```
global output
output = []

# ============================ Window ================================

def program_info():
    infoWindow = Tk()
    infoWindow.title("Program Information")
    infoWindow.geometry("640x480+100+100")
    infoWindow.resizable(0, 0)

    test_font                                                                    =
tkFont.Font(family="/usr/share/fonts/Wemakeprice/Wemakeprice-Regular",
size=12)

    # Project
    # Team name
    # About Project
    a = StringVar()
    project_name = StringVar()
    team_name = StringVar()
    #program_def = StringVar()
    #manual = StringVar()

    a.set("2021 정보보호학과 졸업작품")
    project_name.set("인공지능을 활용한 악성파일 탐지 시스템 ")
    team_name.set("팀명 : 네카라쿠배")
    #program_def.set("")
    program_def = '''윈도우의 PE 구조의 파일을 PE헤더, 4-gram(N-gram), 바이너리
이미지의
방법으로 특징을 추출하였고 분류 알고리즘을 이용하여 모델링을 진행,
가장 높은 정확도가 나오는 방법을 택해서 본 프로젝트를 진행하였다.
PE헤더와 check_packer 기능과 함께 추출한 특징을 RandomForest 알고리즘을
이용한 방법을 활용하여 Kicom 오픈소스 백신을 이용, GUI 프로그램을 만들어
프로젝트를 완료하였다.
'''
    #manual.set("")
    manual = '''1. 스캔하고자 하는 폴더를 선택합니다.
2. 선택된 경로를 확인하고 스캔 버튼을 눌러서 결과를 확인합니다.
3. 악성 확률을 확인하고 파일을 지울지 선택합니다. '''
```

```
    w1 = Label(infoWindow, textvariable=a).pack(side='top')
    w2 = Label(infoWindow, textvariable=project_name).pack(side='top')
    w3 = Label(infoWindow, textvariable=team_name).pack(side='top')
    w4 = Label(infoWindow, justify=LEFT, text=program_def, font=test_font
).pack(side='top')
    w5 = Label(infoWindow, justify=LEFT, text=manual, font=test_font
).pack(side='top')

    closeBtn = Button(infoWindow, text='Close', command=infoWindow.destroy)
    closeBtn.pack(side='bottom', pady=10)

    infoWindow.mainloop()

# ============================ Window3 ================================
def team_info():

    import webbrowser

    new = 1
    url = "http://www.nekabe.me"

    webbrowser.open(url, new=new)

# ============================ Function ================================
def delMal():
    comm = ['python', 'k2.py']
    comm.append('-l')
    comm.append(path)
    print comm
    p = subprocess.call(comm)
    '''
    while True:
        res = p.stdout.readline()
        if res == '' and p.poll() is not None: break
        if res:
            outputBox.insert('end', res.strip() + '\n')
            outputBox.see(END)
    '''
    popUp.destroy()

def midDel():
    t2 = threading.Thread(target=delMal)
```

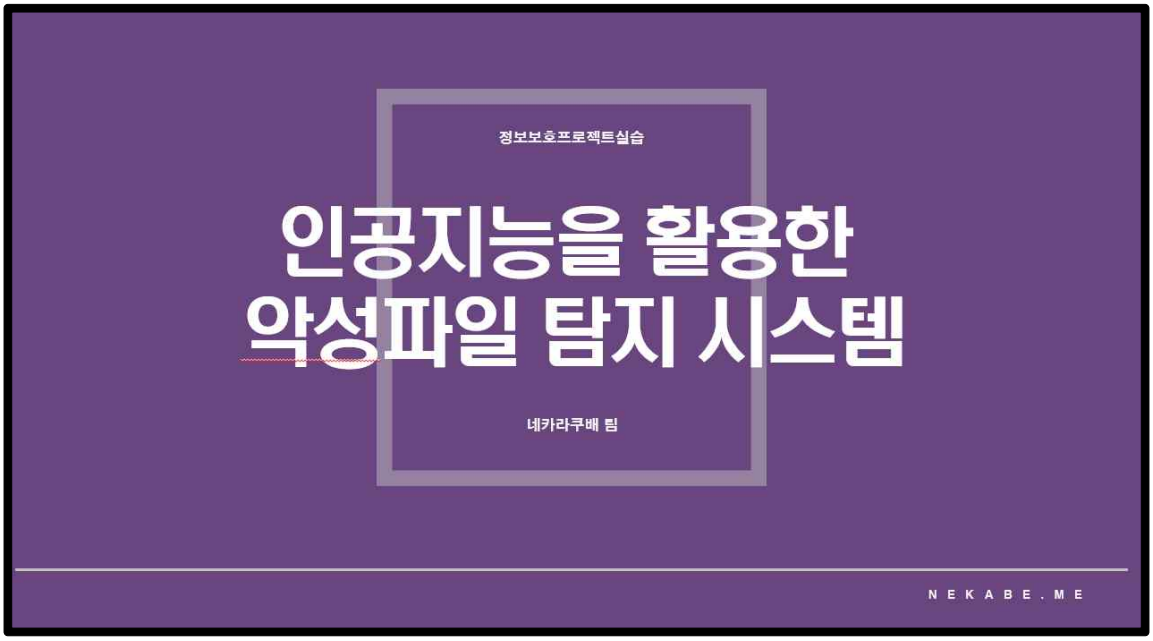
```
    t2.start()

def run():
    comm = ['python', 'k2.py']
    comm.append('-f')
    comm.append(path)
    print comm
    p = subprocess.Popen(comm, stdout = subprocess.PIPE, stderr =
subprocess.STDOUT, shell = False)
    while True:
        res = p.stdout.readline()
        if res == '' and p.poll() is not None: break
        if res:
            outputBox.insert('end', res.strip() + '\n')
            outputBox.see(END)
    rc = p.poll()

    global popUp
    popUp = Tk()
    popUp.title('Result Scan')
    popUp.geometry("200x120+200+200")
    popUp.resizable(0,0)

    popUp_label = Label(popUp, text='Delete?')
    popUp_label.pack(pady=20)

    cancleBtn = Button(popUp, text='Cancle', command=popUp.destroy)
    cancleBtn.pack(side='bottom', pady=5)

    delBtn = Button(popUp, text='Delete', command=midDel)
    delBtn.pack(side='bottom')
    popUp.mainloop()

def midRun():
    t1 = threading.Thread(target=run)
    t1.start()

# ============================ Function ================================
def openDir():
    global path
    dir_path = filedialog.askdirectory(parent=root,
initialdir="/home/stud/Deskop/", title="Select Dir")
```

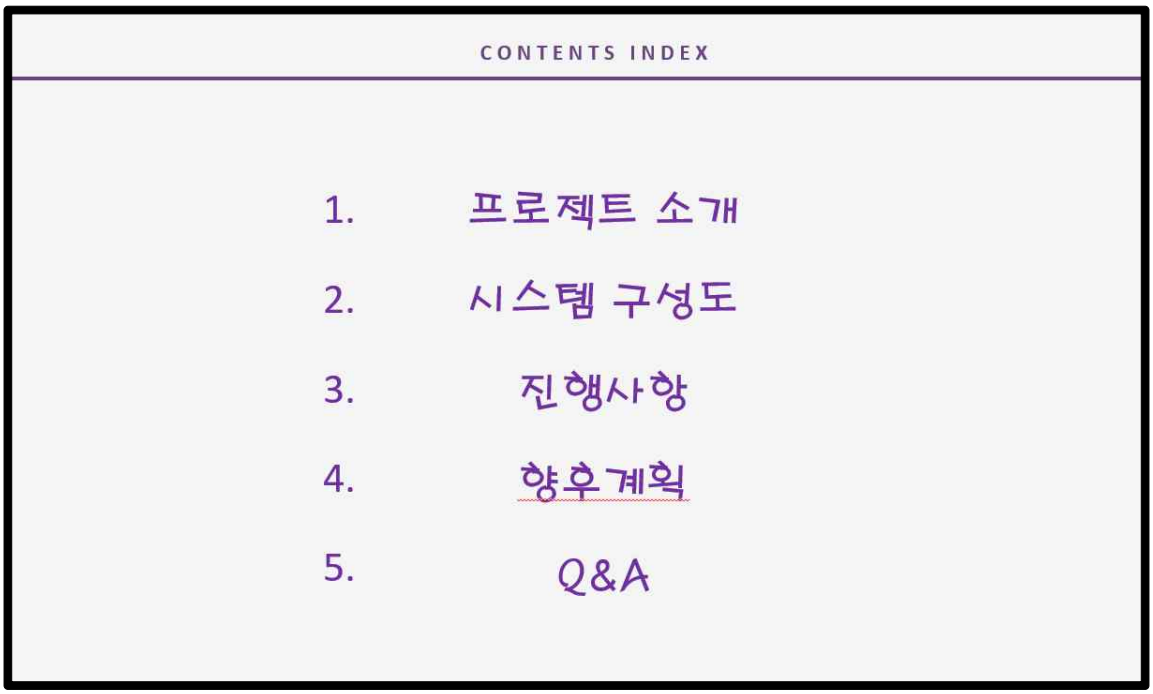
```
    path = dir_path
    lblText.set('Selected\n' + path)

def force_quit():
    pass

# ============================== Main ==================================
root = Tk()
root.geometry('640x480+200+200')
root.resizable(0, 0)
root.title("네카라쿠배 v1.0")

# ============================== Frame1 ================================
top_frame = Frame(root, width=640, height=100)
top_frame.pack(side='top', fill='x')

title_font = tkFont.Font(size=16)

# Frame_left, Frame_right
frame_left  = Frame(top_frame, width=240, height=100)
frame_right = Frame(top_frame, width=400, height=100)
frame_left.pack(side='left', fill='x')
frame_right.pack(side='right', fill='x')

# logo_img
#jbu_logo = Image.open("./jbu.png")
image = PhotoImage(file='jbu.png')
logo_label = Label(frame_left, image=image)
logo_label.place(x=130, y=0)

# Title_label
title_text = StringVar()
#title_text.set("2021 정보보호학과")
#title_text.set("2021 정보보호학과 졸업작품")
#title_text.set("정보보호학과 네카라쿠배")
title_text.set("MalWare Scanner")

title_label = Label(frame_right, textvariable=title_text, font=("Verdana", 24,))
title_label.place(x=0, y=40)

#                   ==============================                    Frame1-2
================================
```

```
top2_frame = Frame(root, width=600, height=40)
top2_frame.pack(side='top', fill='x', ipadx=35)

# result_label
result_text = StringVar()
#result_text.set("[  결 과 창  ]")
result_text.set("[ Result ]")

#result_label = Label(top2_frame, textvariable=result_text, font=('Times', 8))
result_label = Label(top2_frame, textvariable=result_text)
result_label.pack(side='left', padx=35)

test_label = Label(top2_frame, text='')
test_label.pack(side='right', padx=15)

# team_info_button
#tInfoBtn = Button(top2_frame, width=14, text="만든 사람들", font=("Times", 8), command=team_info)
tInfoBtn = Button(top2_frame, width=12, text="CREDIT", command=team_info)
tInfoBtn.pack(side='right', padx=5, pady=2)

# program_info_button
#pInfoBtn = Button(top2_frame, width=14, text="프로그램 정보", font=("Times", 8), command=None)
pInfoBtn = Button(top2_frame, width=12, text="PROGRAM INFO", command=program_info)
pInfoBtn.pack(side='right')

# ============================== Frame2 ================================
middle_frame = Frame(root, width=640, height=280)
middle_frame.pack(side='top')

# Result
outputBox = Text(middle_frame)
outputBox.pack(expand=True)

# ============================== Frame3 ================================
bottom_frame = Frame(root, width=640, height=80)
bottom_frame.pack(side='bottom', fill='x')

lblText = StringVar()
lblText.set("Not Search <Dir>")
```

```
#label font

# Path Label
path_lbl = Label(bottom_frame, textvariable = lblText) #font=font
path_lbl.pack(pady=5)

# Quit Button -> 3
quitBtn = Button(bottom_frame, width=12, text="Quit", command=root.destroy)
#quitBtn.pack(side="right", padx=5, pady=10)
quitBtn.pack(side="right", padx=5, pady=7)

# Scan Button -> 2
scanBtn = Button(bottom_frame, width=12, text="Scan", command=midRun)
scanBtn.pack(side="right", padx=5)

# Select Folder Button -> 1
openBtn = Button(bottom_frame, width=12, text="Select Folder",
command=openDir)
openBtn.pack(side="right", padx=5)

copyright = StringVar()
#copyright.set("Copyright 2021 네카라쿠배")
copyright.set("Copyright 2021 Nekalikube")
copyrightLabel = Label(bottom_frame, textvariable=copyright).pack(side='left',
padx=5)
# ============================== END CODE ==============================
root.mainloop()
```

5.2 발표자료


정보보호프로젝트실습
인공지능을 활용한
악성파일 탐지 시스템
네카라쿠배 팀
NEKABE.ME


CONTENTS INDEX
1. 프로젝트 소개
2. 시스템 구성도
3. 진행사항
4. 향후계획
5. Q&A

CHAPTER.1

프로젝트 소개

프로젝트 주제 선정

‘AI 기반 멀티레이어 악성코드 탐지·차단’ 차세대 안티바이러스 부상

발행일 : 2021.03.09

[올쇼TV] 토목 BIM 계획/설계 효율성을 향상할 수 있

[솔루션가이드] AI 기반 막강 보안 안티바이

날로 지능화되는 사이버 공격은 올해 기업의 비대면 화, AI를 활용한 해킹 지능화, 산업설비에 대한 위협 증대, 클라우드 대상 공격 증가, 의료 분야 집중 공격

기업들은 이제 기존의 방식이 아닌 보다 적극적인 보 강의 등 비대면 시대로 다양하고 지능적인 해킹 및 악 안티 바이러스 솔루션도 절실 해졌다.

IT >

KISA, “AI와 빅데이터 기술 활용해 사이버 위협 방어”

이원태 KISA 원장 기자간담회

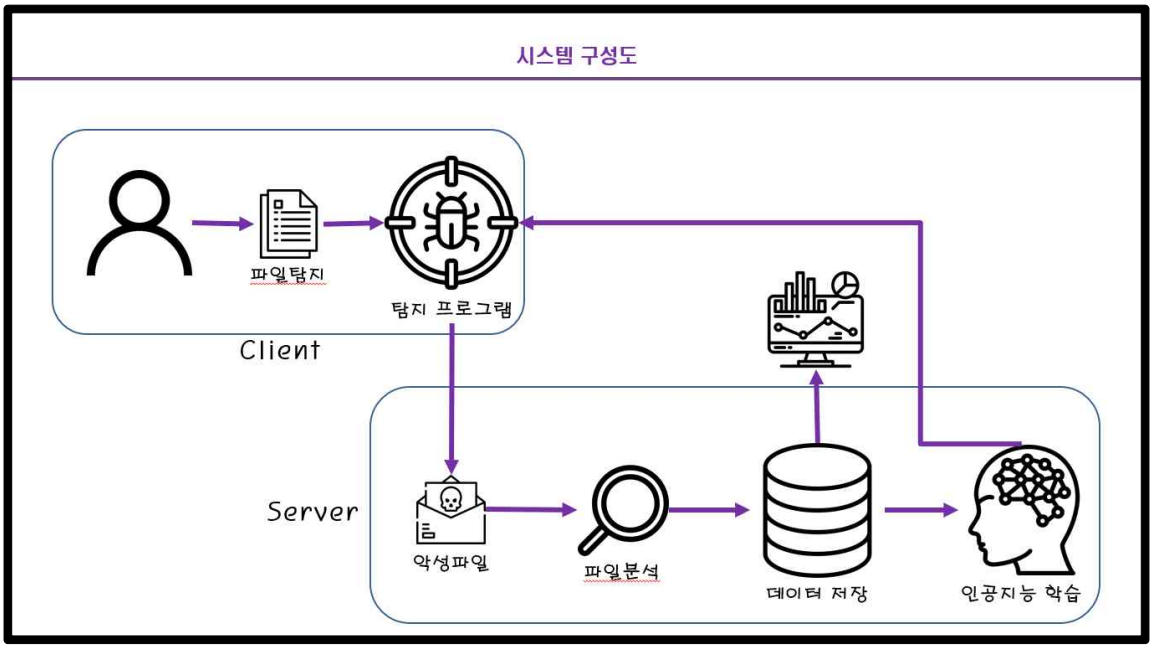
박지영 기자

입력 2021.05.23 13:44

이원태 한국인터넷진흥원(KISA) 원장이 취임 후 처음으로 가진 기자간담회에서 사이버 위협 방어를 위해 인공지능(AI)과 빅데이터 기술을 활용해 예방 시스템을 구축하겠다는 계획을 밝혔다.

CHAPTER.2

시스템 구성도



진행사항

자료/데이터 수집

진행사항

특징 추출 / 분석

구분	설명	장점	단점
PE 헤더	PE 헤더의 필드 값을 추출 및 가공	- 추출이 비교적 쉽고 간단함. - 프로그램 특성에 대한 많은 정보 획득 가능	- 프로그램의 내부 로직은 반영하지 못함. - 많은 부분이 쉽게 변조 가능
PE 이미지	전체 프로그램을 이미지로 변환	- 사람이 인식할 수 없는 많은 특징을 자동으로 추출 가능	- 결과 해석이 불가능 - 많은 데이터가 필요
코드 패턴	코드 섹션에 위치한 어셈블리 코드의 패턴 추출	- 실제 프로그램의 기능을 특징에 반영 - 강력한 특징을 생성해 낼 수 있음	- 안티 리버싱이 적용된 경우 코드 패턴이 달라질 수 있음. - 정확한 추출이 어려움.

특징추출 – PE 헤더

특징 추출 / 분석

DOS 헤더 추출

```
def extract_dos_header(self,pe):
    IMAGE_DOS_HEADER_data = [ 0 for i in range(6)]
    try:
        IMAGE_DOS_HEADER_data = [
                    pe.DOS_HEADER.e_cblp,#
    except Exception,
        print e
    return IMAGE_DOS
```

```
def extract_file_header(self,pe):
    FILE_HEADER_data = [ 0 for i in range(
    FILE_HEADER_char =  []
    try:
        FILE_HEADER_data = [
                    pe.FILE_HEADER.NumberOfSec
```

Optional 헤더 추출

```
def extract_optional_header(self,pe):
    OPTIONAL_HEADER_data = [ 0 for i in range(21)]
    DLL_char =[]
    OPTIONAL_HEADER_data2 = [ 0 for i in range(6)]

    try:
        OPTIONAL_HEADER_data = [
            pe.OPTIONAL_HEADER.MajorLinkerVersion,#
            pe.OPTIONAL_HEADER.MinorLinkerVersion,#
            pe.OPTIONAL_HEADER.SizeOfCode,#
                                                    ADER.SizeOfHea
    return OPTIONAL_HEADER_data + DLL_char + OPTIONAL_HEADER_data2
```

filename	MD5	e_cblp	e_cp	e_cparhdr	e_maxalloc	e_sp	e_lfanew	NumberOfS	CreationYe	FH_char0	FH_char1	FH_char2	FH_char3	FH_char4	FH_char5	FH_char6
d7f8d89e1	d7f8d89e1	144	3	4	65535	184	128	3	1	0	1	0	0	0	0	0
d952e206l	d952e206l	0	1	2	17744	332	12	3	1	0	1	1	1	0	0	0
bce82de8	bce82de8	144	3	4	65535	184	200	4	1	1	1	0	0	0	0	0
13C9E872	13c9e872f	144	3	4	65535	184	264	5	1	0	1	0	0	0	0	0
BE0E0BBA	be0e0bba	144	3	4	65535	184	216	5	0	0	1	0	0	0	0	0
9ebaf9f71	9ebaf9f71	80	2	4	65535	184	256	8	1	1	1	1	1	0	0	1
ebd2813e	ebd2813e	80	2	4	65535	184	256	8	1	0	1	1	1	0	0	1
1d6d1570	1d6d1570	80	2	4	65535	184	256	3	1	0	1	1	1	0	0	1
D7CD00A	d7cd00a1	80	2	4	65535	184	256	9	0	0	1	1	1	0	0	1
d39fdfe34	d39fdfe34	144	3	4	65535	184	128	5	1	1	1	1	0	0	0	0
5502856b	5502856b	144	3	4	65535	184	128	3	0	0	1	0	0	0	0	0
fe5ab4c19	fe5ab4c19	144	3	4	65535	184	128	4	1	0	1	0	0	0	0	0
89f3a3fb9	89f3a3fb9	144	3	4	65535	184	232	5	1	1	1	1	1	0	0	0
e37d3914	e37d3914	80	2	4	65535	184	256	3	1	1	1	1	1	0	0	1
8163C7A4	8163c7a4	80	2	4	65535	184	256	8	1	0	1	1	1	0	0	1
6eb468e4	6eb468e4	80	2	4	65535	184	512	8	0	0	1	1	1	0	0	0
f2cd4cd14	f2cd4cd14	144	3	4	65535	184	216	4	0	0	1	0	0	0	0	0
41f24c96d	41f24c96d	144	3	4	65535	184	264	5	0	0	1	0	0	0	0	0

* 60개의 Raw 특징

모델링

모델링

분류 알고리즘

```
class Classifiers():

    def __init__(self, X, Y):

        self.x_train, self.x_test, self.y_train, self.y_test = #
        train_test_split(X, Y, test_size=0.2, random_state=0)


    def do_svm(self):
        clf = SVC()
        clf.fit(self.x_train, self.y_train)
        y_pred = clf.predict(self.x_test)

        return accuracy_score(self.y_test, y_pred)

    def do_randomforest(self, mode):

        clf = RandomForestClassifier()
        clf.fit(self.x_train, self.y_train)

        if mode == 1:
            return clf.feature_importances_
        y_pred = clf.predict(self.x_test)

            return accuracy_score(self.y_test, y_pred)

    def do_naivebayes(self):
        clf = clf = GaussianNB()
        clf.fit(self.x_train, self.y_train)
        y_pred = clf.predict(self.x_test)

        return accuracy_score(self.y_test, y_pred)
```

```
def do_all(self):
    rns = []

    rns.append(self.do_svm())
    rns.append(self.do_randomforest(0))
    rns.append(self.do_naivebayes())
    rns.append(self.do_dnn())

    return rns
```

```
pe_all_tmp = pe_all
pe_all = pe_all.drop(['filename',

Y = pe_all['class']
X = pe_all.drop('class', 1)
Y_bak = Y

md_pe = model.Classifiers(X, Y)
df.loc['pe'] = md_pe.do_all()
```

* DNN 은 너무 길어 생략.

One-hot encoding

```
from sklearn.preprocessing impor
from sklearn.preprocessing impor

def hot_encoding(df):

    enc = OneHotEncoder(handle_unknown='ignore', sparse=False)
    lab = LabelEncoder()

    dat = df['packer_type']
    lab.fit(dat)
    lab_dat = lab.transform(dat)

    df = df.drop('packer_type', 1)
    lab_dat = lab_dat.reshape(len(lab_dat), 1)
    enc_dat = enc.fit_transform(lab_dat)
    enc_dat = pd.DataFrame(enc_dat, columns=lab.classes_)

    df = df.reset_index(drop=True)
    enc_dat = enc_dat.reset_index(drop=True)

    df = pd.concat([df, enc_dat], axis=1)

    return df, lab.classes_
```

모델링

모델링

PE 헤더 -> Random Forest 모델을 이용, 중요도 값을 조회, 이를 이용한 상위 20가지 특징

PE 헤더 상위 20

```
imp = md_pe.do_randomforest(1)

imp = dict(zip(col, imp))
sorted_imp = sorted(imp.items(), key=operator.itemgetter(1), reverse=True)
imp_20 = sorted_imp[0:20]
pe_all = pe_all_tmp
pe_all = pe_all.drop(['filename', 'MD5', 'packer_type'], 1)

Y = pe_all['class']
pe_top20 = pe_all.drop('class', 1)
X = pe_top20[imp_20.keys()]

md_pe_top20 = model.Classifiers(X, Y)
df.loc['pe_top20'] = md_pe_top20.do_all()
```

	svm	randomforest	naivebayes	dnn	avg
pe	0.942222	0.987302	0.151111	0.061587	0.535556
pe_packer	0.942222	0.988571	0.151111	0.061587	0.535873
pe_top20	0.942222	0.986667	0.196825	0.061587	0.546825

교차검증 - 10 fold

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import model

pe_nor = pd.read_csv('nor_mod.csv')
pe_mal = pd.read_csv('mal_mod.csv')
pe_all = pd.concat([pe_nor, pe_mal])

pe_all.shape

pe_all = pe_all.drop(['filename', 'MD5', 'pa

Y = pe_all['class']
X = pe_all.drop('class', 1)

md = model.Classifiers(X, Y)
acc = md.do_randomforest(0)
acc
```

Grid Search

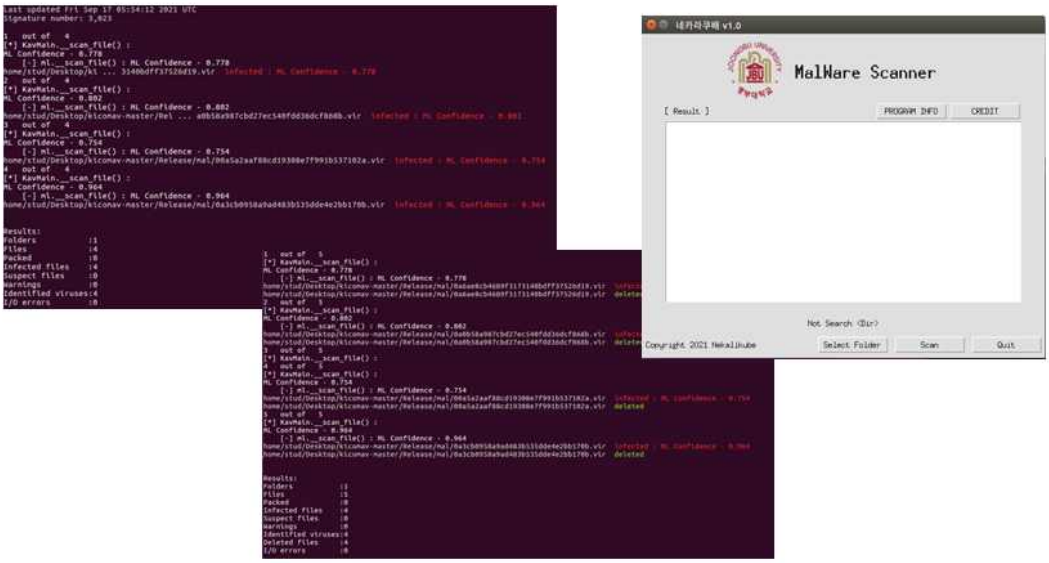
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

parameters = {'n_estimators':[100, 200, 500, 1000], 'max_features':['auto', None],
              'max_depth':[6, 10, 12]}
from sklearn.model_selection import cross_val_score

abc = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=12, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)

abc.fit(X,Y)

score = cross_val_score(abc, X, Y, cv=10)
print score
print np.mean(score)
```

10-fold 교차검증

```
[0.98604061 0.98984772 0.99111675 0.99111675 0.99491741 0.99491741
 0.99364676 0.98854962 0.98727735 0.99618321]
0.9913613581131774
```

10-fold 교차검증 -> 최적의 매개변수 조합 찾기

클라이언트



CHAPTER.4

향 후 계 획

향후계획

- 계속해서 모델을 구축하여 평가 및 최적화를 진행
- 구현된 모델에 많은 데이터를 학습 및 테스트 진행
- Anti-malware 개발
- 데이터 처리 및 AM을 위한 백엔드 서버 구축
- 서버 및 로그 관리를 위한 관리자 웹서버 구축