ISSN 1229-800X [Print] / ISSN 2586-7792 [Online]
The Transactions of the Korean Institute of Electrical Engineers, vol. 69P, no. 3, pp. 187~192, 2020
https://doi.org/10.5370/KIEEP.2020.69.3.187

A Study on Malware Detection System Using Static Analysis and Stacking

정적 분석과 스태킹 기법을 활용한 악성 코드 탐지 시스템에 관한 연구

Jin-Young Cho · Eun-Gi Ko · Hye-Bin Yoo · Mi-Ri Cho · Chang-Jin Seo

조진영* · 고은기* · 유혜빈* · 조미리* · 서창진†

Abstract

The Fourth Industrial Revolution, the introduction of ICT technology, expanded throughout society, increases daily convenience and industrial productivity. But along with the advancing technology, new malicious program threats have also continued to surge. This study proposes a malware detection method using static analysis and stacking techniques to detect new malware as fast and accurately. And we used PE header features, which are extracted through static analysis to process malware without executing it actually. The pe_packer feature was the most efficient in the experiment due to processing the extracted data in various ways and applying it to the machine learning model. So we selected as the feature data to be used to the stacking model. The detection model is implemented based on the stacking technique rather than a single model to detect with high accuracy. The proposed system can classify malware or normal files with fast and accurately. And It has a 94.7 percent detection rate and is better than the existing single model-based detection system.

Key Words

Detection, Malware, Benign, Static Analysis, PE header, Machine Learning, Stacking model

1. 서 론

정보 산업 기술의 발전으로 인해 일상의 편리성과 산업 분야의 생산성을 얻었지만 이와 동시에 보안을 위협하는 사례 또한 계속해서 증가하고 있다. 빅데이터, IoT 분야 등의 IT 기술이 빠르게 발전하면서 인터넷 사용이 당연시되는 사회가 되었고 사이버 공격과 보안은 더 PC 내에서 만의 일이 아니라 네트워크와 인프라 전체로 확대됨으로써 우리의 생활에 직접적인 영향을 끼치고 있다. 이러한 문제의 중심에는 악성 코드가 존재한다. 악성 코드는 사용자의 기기에 접근하여 개인정보 유출, 원격 제어, 네트워크 트래픽 발생, 시스템 성능 저하, 금전적 손해 등 여러 문제를 발생시킨다. AV-TEST의 통계에 따르면 2018년 신종 악성 코드 위협은 137,473,281개로 하루에 376,639개, 초당 4.4개다. 최근 3년간 전체 악성 코드의 수는 2017년 719.15M(Million), 2018년 856.62M, 2019년 1001.52M로 계속해서 증가하고 있음을 확인할 수 있다[1]. 이러한 악성 코드를 탐지하기 위해 가장 일반적으로 사용되는 방식은 시그니처 기반(Signature Based) 방식이나 휴리스틱 기반(Heuriestic Based) 방식이다. 시그니처 기반 방식은 분석 인력이 악성 코드 행위 규칙이나 고유 바이너리 형태를 시그니처로 제작해 비교 분석하여 악성 코드를 탐지하는 방식이고, 휴리스틱 기반 방식은 어느 정도 유사도가 있는지 비교하는 방식이다. 휴리스틱 기반 방식은 시그니처 기반 방식의 단점을 보완하기 위해 사용하고 있지만 오탐율(False Positive Rate)의 가능성이 커진다. 또한, 이전에 수집된 악성 코드로부터 추출한 데이터를 기반으로 탐지하는 방식이기 때문에 변종 된 악성 코드나 제로데이(Zero-day) 공격에 즉각적으로 대응하기 어렵다. 이러한 한계를 극복하기 위해 인공지능을 이용한 악성 코드 탐지 방식이 연구되고 있다. 인공지능 분야는 알파고를 시작으로 자율주행, 챗봇 등 최근 각종 분야에서 주목받고 있는 기술이다. 삼성SDS에서 시그니처 기반 안티바이러스와 인공지능 기반 Anti-Virus의 탐지율 테스트를 진행한 결과 변종 악성 코드 부분에서 시그니처 기반 Anti-Virus는 93.8%의 오탐율을 기록했지만, 인공지능 기반 Anti-Virus는 0%의 오탐율을 기록하였다. 더불어 인공지능 기반 Anti-Virus는 문서 파일 이외에 실행 파일, 랜섬웨어 등의 파일 타입에서도 1% 안팎의 오탐율을 기록하였다[2]. 이렇듯 기존의 악성 코드 탐지 방식의 한계점을 보완하기 위해 인공지능을 기반으로 한 악성 코드 탐지 기술에 관한 연구가 활발하게 진행 중이다.

악성 코드의 특징 데이터를 추출하는 방식은 크게 세 가지

† Corresponding Author : Dept. of Information Security Engineering Sangmyung University, Korea.
E-mail: cjseo@smu.ac.kr
* Dept. of Information Security Engineering Sangmyung University, Korea.
https://orcid.org/0000-0002-3761-1564 https://orcid.org/0000-0002-1697-944X
https://orcid.org/0000-0002-6402-7988 https://orcid.org/0000-0002-4027-8889
Received : July 6, 2020 **Revised** : August 14, 2020 **Accepted** : August 29, 2019

로 나뉜다. 샌드박스와 같은 자동화 플랫폼을 이용한 자동화 분석, 악성 코드를 실행하면서 실제 실행되는 코드와 행위를 위주로 분석하는 동적 분석, 악성 코드를 실행하지 않고 악성 코드 바이너리 파일 자체를 분석하여 정보를 얻는 정적 분석으로 구분할 수 있다. 정적 분석 방식은 다른 방식에 비해 비교적 간단하고 빠르게 프로그램의 특성에 대한 전반적인 정보를 얻을 수 있다. 특히 직접 프로그램을 실행하지 않아 감염의 위험이 적다.

따라서 본 연구에서는 악성/정상 파일을 정적 분석 방식으로 데이터 전처리하여 얻은 PE(Portable Executable) header 특징값을 스태킹 기법에 적용하여 학습 및 분석하고 악성 코드 파일을 탐지하는 방식을 제안하고자 한다.

2. Related Work

Ji-hee Ha[3]의 연구에서는 PE header 내의 Imported DLL (Dynamic Link Library)과 API 특징을 정적 분석하여 악성 코드를 탐지하는 데 사용한다. Feature의 성능을 확인하기 위해서는 DNN(Deep Neural Network) 모델을 사용한다. DLL/API 정보를 추출하여 악성 코드와 정상 파일 간의 출현 비율에 따른 경향성을 파악해 악성 코드 탐지에 신속하고 가벼운 Feature를 선정한다. 머신 러닝 결과를 비교 분석해 정적 분석만으로도 API는 91% 이상, DLL에서는 86% 이상의 정확도가 나타남을 보여준다. 해당 논문의 연구에서는 PE header 정보 전체를 사용하는 것이 아니라 DLL과 API 특징만을 사용한다. 또한, 악성 코드 탐지의 정확도를 위한 것이 아닌 DLL/API 정보에서 악성 코드를 탐지하는데 유용한 특징을 찾고 이를 검증하는 데에 머신 러닝을 사용한다.

Mansour Ahmadi[4]의 연구에서는 Microsoft에서 제공한 Malware Challenge 데이터를 XGBoost 모델을 이용해 학습시킨 후 악성 코드의 특징에 따라 패밀리로 분류한다. 데이터 정적 분석을 통해 PE header에서 추출한 바이트 기반 특징 5개와 디스어셈블리 기반 특징 8개를 특징 데이터로 사용한다. 실험은 교차검증(cross-validation)으로 진행되었다. 대부분의 훈련 특징에서 95% 이상의 정확도를 보였으며, 모든 특징을 훈련 데이터로 구성 시 99%의 정확도로 가장 성능이 우수했다. 하지만 해당 논문의 연구에서는 PE header에서 추출할 수 있는 13개의 특징만을 데이터로 사용하며, 악성 코드 파일을 탐지하는 것이 아닌 악성 코드에 따라 구별되는 특성을 그룹화한 것을 기반으로 패밀리로 분류하는 방식을 제안한다.

Hyunjong Lee[5]의 연구에서는 악성 코드 패밀리 분류를 위한 훈련 데이터의 API/DLL 특징을 구성해 앙상블 모델을 기반으로 한 다중 분류 성능을 분석한다. PE 구조의 IAT(Import Address Table)을 분석하여 API/DLL 정보를 추출하고 어셈블리 코드를 분석하여 전처리한다. 악성 코드 학습을 위한 모델로 트리 기반 알고리즘인 XGBoost과 Random Forest를 사용한다. 정상 코드와 악성 코드를 이진 분류하는 실험과 악성 코드 패밀리를 분류하는 다중 분류 실험을 진행했다. 실험은 교차검증으로 진행되었다. 성능 비교에서 악성 코드 탐지율은 Random Forest가 93%, 악성 코드 패밀리 분류 정확도는 XGBoost가 92%, 정상 코드를 포함하는 테스트 오탐율은 3.5%이다. 해당 논문의 연구에서는 API 특징을 사용하기 위해 API 목록을 사전에 정의해야 하며 윈도우 운영체제 업데이트나 함수 이름 변경으로 인해 목록과 이름이 상이해 질 경우 특징값이 클래스와 상관없이 희소해진다. 따라서 API 특징을 기반으로 한 탐지 모델은 장기간 사용하기에 부적절하다.

이러한 연구 동향들은 정적 추출 가능한 다양한 정보를 기반으로 특징을 추출하고 머신 러닝을 이용하여 악성 코드 탐지에 활용하고 있음을 보여준다. 하지만 PE 파일에서 얻을 수 있는 정보 중 일부만을 사용하거나 머신 러닝 단일 모델만을 사용하여 분류 모델을 구현하였다. 본 연구에서는 대량으로 발생하는 악성 코드를 적은 자원을 소모하면서 신속하게 처리하고자 정적 분석 방식을 사용하여 얻을 수 있는 PE header와 opcode 특징 데이터를 추출하여 특징 데이터로 사용하고자 한다. 또한, 다른 앙상블 기법과는 달리 단일 모델을 혼합하여 사용하는 스태킹 기법을 사용하여 악성 코드 파일 탐지의 정확도를 높이고 오탐율을 낮추고자 한다.

3. Proposed Model

본 연구에서 제안하는 악성 코드 탐지 시스템은 데이터 전처리(data preprocessing), 특징 추출(feature extraction), 모델 학습(model learning), 모델 평가(model evaluation)의 단계를 거친다.

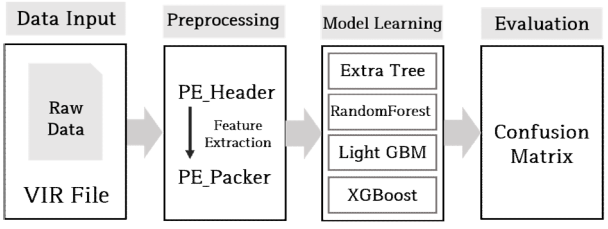

그림 1 시스템 구성도
Fig. 1 System Overview

3.1 Data Preprocessing and Feature Extraction

PE 파일은 윈도우 운영체제에서 사용되는 실행파일, DLL, 폰트 파일, 객체 코드 등을 의미한다. PE 파일에서 필요한 정보는 대부분 PE 구조의 header 부분에 존재한다. PE header 특징을 추출하기 위해 ClaMP(Classification of Malware with PE header) 오픈소스 프로젝트에서 제공하는 스크립트를 사용한다[6]. 이 스크립트를 이용하여 PE header 중 DOS header에서 6개, FILE header에서 17개, OPTIONAL header에서 37개 총

60개의 Raw 특징과 7개의 Derived 특징을 추출한다. Derive 특징은 PE header의 값을 한 번 더 가공하여 의미 있는 정보를 추출한 특징이다. 본 연구에서는 최적의 성능을 보이는 특징을 찾고자 추출한 PE header를 세 가지 방식으로 가공하여 테스트한다. 우선 추출한 PE header 중 packer_type column이 범주 형 데이터를 가지고 있으므로 이 column을 제거하여 pe_header라는 특징을 생성하였으며, 같은 column을 one-hot encoding 하여 가공한 pe_packer 특징을 생성한다. <그림 2>는 packer_type column의 데이터 중 대표적인 4가지를 one-hot encoding 시킨 모습이다. 마지막으로 피어슨의 상관 계수를 사용하여 pe_top 특징을 생성한다. 전체 pe_header 특징 중 일부를 상관 계수 시각화하여 <그림 3>과 같이 나타내었다. class와 0.0~0.3까지의 상관 계수를 가지는 45개의 column만을 따로 추출하여 pe_top 특징으로 사용한다.

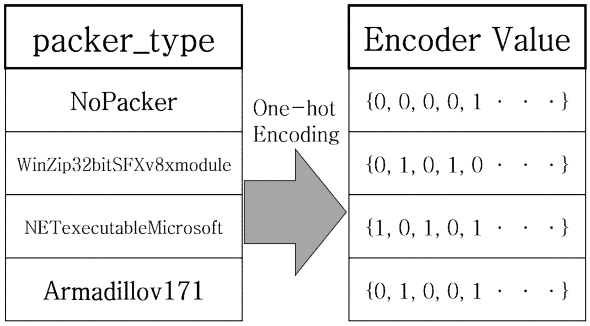

그림 2 pe_packer특징 가공 시 이용한 one-hot encoding 방식의 예시
Fig. 2 Example of One-Hot Encoding Method used in Processing pe_packer Features

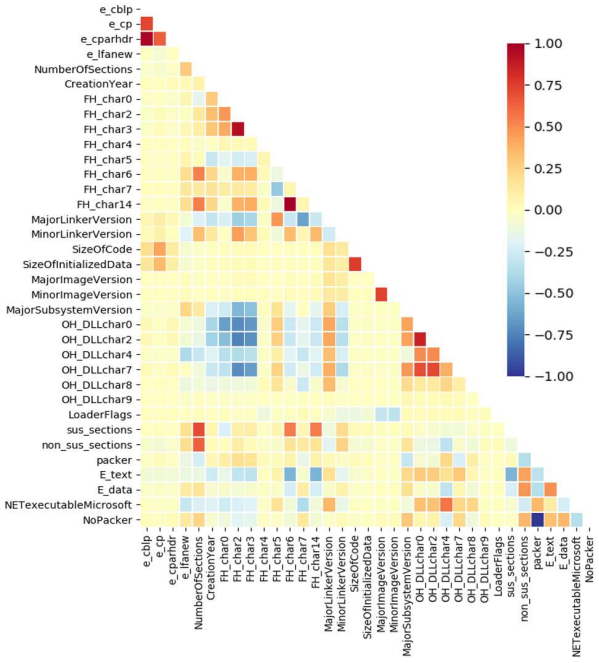

그림 3 pe_top특징을 위한 상관 계수 시각화
Fig. 3 Correlation Matrix for pe_top Features

PE header 부분의 특징뿐만 아니라 데이터 영역에 있는 실제 코드 부분에서도 특징을 추출한다. 코드 부분의 위치를 찾아 바이트로 되어있는 opcode 데이터를 어셈블리 코드로 변환한다. 변환한 코드를 N-gram 분석으로 특징 추출한다. N-gram은 문자열에서 N개의 연속된 요소를 추출하는 자연어 처리 방식이다. 연속된 N개의 opcode를 묶어 하나의 패턴으로 인식하고 같은 패턴의 수를 카운트한다. N의 수가 커질수록 패턴의 크기가 커지고 개수를 세는 확률이 줄어들어 희소 문제가 생길 수 있으므로 본 연구에서는 N의 값이 3과 4인 경우만 테스트한다. 각 데이터 파일의 opcode 3-gram, 4-gram 패턴을 추출하여 하나의 파일에서 패턴 개수가 상위 100개인 패턴만을 따로 추출하여 특징값으로 사용한다.

추출된 특징 중 최적의 조합을 평가하기 위해 여러 분류 알고리즘에 넣어 테스트한다. 테스트에 사용한 분류 알고리즘은 Logistic Regression, SVM, Random Forest, XGBoost이다. 특징 추출을 위한 데이터로는 2018년 정보 보호 R&D Data Challenge AI 기반 악성 코드 탐지 트랙에서 제공된 학습 데이터 15,000개를 사용한다. 15,000개의 파일 중 10,500개는 악성 파일, 4,500개는 정상 파일이며, 추출한 전체 데이터 15,000개 중 80%를 학습에 사용하고 20%를 테스트에 사용한다.

표 1 추출한 특징을 분류 알고리즘을 이용해 학습한 결과
Table 1 The Result of Learning the Extracted Features using Classification Algorithms

Model / data	LR	SVM	RF	XGB	AVG
pe_header	0.708	0.709	0.939	0.933	0.822
pe_packer	0.708	0.709	0.943	0.934	0.824
pe_top	0.707	0.709	0.927	0.919	0.816
4gram	0.743	0.735	0.804	0.811	0.773
3gram	0.773	0.730	0.823	0.819	0.786
pe_4gram	0.690	0.696	0.909	0.931	0.807
pe_3gram	0.691	0.696	0.908	0.923	0.804

pe_3gram과 pe_4gram 특징은 pe_packer 특징과 N-gram 특징을 열 병합하여 하나의 데이터로 합친 특징이다. <표 1>의 학습 결과를 보면 특징의 전체적인 평균은 PE header를 가공한 특징들이 N-gram 단일 특징보다 높다는 것을 볼 수 있다. 또한, packer_type column을 one-hot encoding 한 packer 정보를 추가하여 생성한 pe_packer 특징이 pe_header 특징보다 정확도가 높다. 최고 정확도 기준으로 볼 때 pe_packer 특징을 XGboost 모델에 적용했을 때 가장 높은 정확도를 기록한다. 파일에 따라 추출할 수 없는 경우가 존재하는 N-gram 특징과 달리 PE header는 프로그램 전반에 대한 정보를 제공하는 특징으로 모든 윈도우 프로그램이 가지고 있다. 따라서 모든 파일에서 추출할 수 있으며 다른 특징에 비해 더 높은 정확도를 보인 pe_packer 특징을 모델링에 사용할 최종 특징으로 선택한다.

3.2 Model Learning

악성 코드 학습을 위해 Meta modeling이라고 불리는 스태킹(stacked generalization)기법을 사용한다[7]. 스태킹은 다른 머신 러닝 앙상블 기법과 달리 최고의 성능을 내기 위해 서로 다른 단일 모델을 혼합하여 사용하는 방식이다. 훈련 데이터 세트를 이용하여 Sub-model의 예측 결과를 생성하고 이 결과를 다시 Meta leaner의 훈련 데이터로 사용한다. 즉 앞선 sub-model의 예측값을 입력값으로 하여 meta leaner가 최종 예측값을 내는 방식의 알고리즘이다. 같은 데이터로 반복해서 훈련하게 되면 과적합(overfitting) 문제가 발생할 수 있으므로 CV 기반의 스태킹 방식을 사용한다. CV 기반의 스태킹 방식은 각 Sub-model의 훈련 데이터 세트를 K개로 나눈 뒤 테스트를 K 번 수행하는 K-fold 교차검증 방식을 이용한다. 각 Sub-model을 정의한 후, fold 별로 나누어진 훈련 데이터 세트(train)로 Sub-model 모델을 훈련하고 검증 데이터 세트(test)에 대한 예측을 수행하여 예측 결과를 내는 K-fold averaging prediction이 수행된다. 모델마다 K 번 예측 후 이 예측치의 평균을 결과 예측값(Mean of Temporary predictions)으로 지정한다. 생성된 결과 예측값은 Meta learner의 훈련 데이터로 사용하여 모델 훈련을 진행한다. 이후 최종적으로 x_test으로 예측을 수행하여 y_test와 비교해 최종 모델을 평가한다.

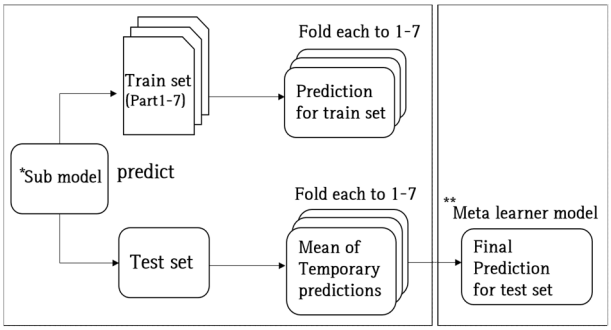

그림 4 스태킹 모델 구성도
Fig. 4 Stacking Model Overview

표 2 스태킹 Sub-model 예측 결과
Table 2 Predicted Result of Stacking Sub-model

	ET	RF	LGBM	XGBM
fold 0	0.9387	0.9387	0.9428	0.9375
fold 1	0.9329	0.9393	0.9329	0.9340
fold 2	0.9305	0.9399	0.9381	0.9352
fold 3	0.9405	0.9527	0.9556	0.9475
fold 4	0.9416	0.9521	0.9457	0.9481
fold 5	0.9311	0.9335	0.9370	0.9352
fold 6	0.9381	0.9492	0.9463	0.9399
MEAN	0.9362 +0.004	0.9436 +0.007	0.9426 +0.007	0.9396 +0.005
FULL	0.9362	0.9436	0.9426	0.9396

본 연구에서는 Sub-model에 Extra Tree, Random Forest, Light XGBoost, XGBoost를 사용하고 최종 Classifier인 Meta learner에는 XGBoost를 사용한다. 또한, 스태킹 모델을 구현하기 위해 vecstack[8]과 Sklearn 패키지를 사용한다. 전체 데이터 중 80%를 훈련 데이터 세트로 사용하고 20%를 검증 데이터 세트로 사용하였으며 K-fold 교차검증의 K값을 7로 지정하여 학습을 진행한다. 7-fold averaging prediction이 수행되면 훈련 데이터 세트와 검증 데이터 세트에 대해 각각의 예측 결과인 S_train과 S_test가 생성되고 이 값을 최종 Classifier인 XGBoost로 학습시켜 최종 예측 결과(Final Prediction)를 출력한다. 스태킹 모델의 구성도는 <그림4>와 같고, Sub-model에 데이터 세트를 넣어 추출한 각 모델의 예측값은 <표 2>와 같다. 각 모델 예측값의 평균을 최종 Classifier에 입력한 결과 0.952969의 최종 예측 결과가 측정되었다. 이는 앙상블 기법인 스태킹 모델을 사용함으로써 단일 모델만을 사용한 경우보다 정확도가 개선됨을 보인다.

4. Experiments Result

4.1 Model Evaluation

실험 결과를 서술하기 이전에 성능 평가 지표에 관해 설명한다. 본 논문에서는 성능 평가를 위한 척도로써 정밀도 PRE(Precision), 탐지율 TPR(True Positive Rate), 오탐율 FPR(False Positive Rate), 분류 정확도 ACC(Accuracy), F 점수(F-score)를 비교한다. 정밀도는 악성 파일이라고 판단한 파일 중 실제 악성 파일에 해당하는 비율을 말한다(식 1). 탐지율은 실제 악성 파일을 악성 파일로 정확하게 판단한 비율을 말하며 클 값을 가질수록 좋은 시스템이고(식 2), 오탐율은 실제 정상 파일을 악성 파일로 판단한 비율을 말하며 다른 평가 점수와 달리 낮은 값을 가질수록 좋은 시스템이다(식 3). 탐지율과 오탐율은 일반적으로 양의 상관관계를 가진다. 또한, 혼동행렬에서 실제 클래스끼리 계산하기 때문에 클래스 비율에 영향을 받지 않으므로 클래스 비율이 다른 경우 유용한 성능 지표로 사용된다. 분류 정확도는 전체 탐지 결과 중 악성 파일과 정상 파일 모두 정확하게 판단한 비율을 말하며 값이 클수록 좋은 시스템이다(식 4). 마지막으로 F 점수는 모델의 정확도에 대한 척도로 정밀도와 탐지율의 상충 관계를 평가에 반영하여 실제 분류기를 비교하는데 사용되는 지표이다(식 5). 탐지율과 정밀도가 균형을 이룰 때 F 점수의 값이 커진다.

$$PRE: \frac{TP}{TP+FP}\times 100 \tag{1}$$

$$TPR: \frac{TP}{TP+FN}\times 100 \tag{2}$$

$$FPR: \frac{FP}{FP+TN}\times 100 \tag{3}$$

$$ACC: \frac{TP+TN}{TP+TN+FP+FN} \times 100 \quad (4)$$

$$F-score: \frac{PRE \times TPR}{PRE+TPR} \times 2 \quad (5)$$

이를 지표로 사용하기 위해 <표 3>의 혼동 행렬(Confusion matrix)을 사용한다. TP(True Positive)는 실제 Malware(Ture)를 Malware로 정확하게 예측한 결과를 의미하고, FN(False Negative)은 Malware를 Benign(False)으로 예측한 결과를 의미한다. FP(False Positive)는 실제 Benign를 Malware로 예측한 결과를 의미하고, TN(True Negative)은 실제 Benign을 Benign으로 정확하게 예측한 결과를 의미한다.

표 3 혼동 행렬

Table 3 Confusion Matrix

Condition(실제)			
Malware	Benign		
TP (True Positive)	FP (False Positive)	Malware	Prediction (예측)
FN (False Negative)	TN (True Negative)	Benign	

4.2 Result

본 장에서는 Extra Tree 및 XGBoost 단일 모델의 실험 결과와 스태킹 모델의 결과를 비교하기 위한 성능 평가 지표 결과를 제시한다. 준비된 378개의 악성 파일과 466개의 정상 파일 데이터로부터 pe_packer 특징을 추출하여 입력 데이터 세트로 사용한다. 스태킹 모델의 Sub-model에는 Extra Tree, Random Forest, Light XGBoost, XGBoost를 사용하며 Meta learner에는 XGBoost를 사용한다. 모든 실험은 Ubuntu 18.04 (64bit) 운영체제에서 수행되었으며 Anaconda3(64bit), Python3.6 을 사용하여 진행한다. 자세한 실험 환경은 <표 4>와 같다.

표 4 실험 환경

Table 4 Experiment Environments

Name	Specification
OS	Ubuntu 18.04 (64bit)
CPU	Intel i5-10210U
RAM	8.0GB
GPU	NVIDIA GeForce MX250

Extra Tree, XGBoost 단일 모델을 사용하여 학습을 진행한 결과와 스태킹 기법을 사용하여 학습을 진행한 결과를 기반으로 모델 성능 평가 지표를 수행한 결과는 <표 5>와 같다. 정상/

표 5 성능 평가 지표 결과

Table 5 The Result of Performance Evaluation Standard

Model	PRE	TPR	FPR	ACC	F-score
ET Model	87.6%	88.3%	10.0%	89.2%	88.0
XGB Model	78.9%	84.1%	18.2%	82.8%	81.4
Stacking	88.8%	94.7%	9.6%	92.3%	91.9

악성 파일에 대해 정밀도, 탐지율, 오탐율, 정확도, F 점수 평가 결과를 나타낸다. <그림 5>는 성능 평가 지표 실험 결과에 대한 ROC(Receiver Operation Characteristic Curve) 곡선을 그림 그림으로 그래프 아래의 면적이 클수록 좋은 결과이다. 단일 모델의 악성 코드 탐지율은 평균적으로 86.2%, 오탐율은 14.1%, 분류 정확도는 86.0%, 정밀도는 83.25%, F 점수는 84.71의 성능을 보인다. 앙상블 스태킹 모델의 탐지율은 94.7%, 오탐율은 9.6%, 분류 정확도는 92.3%, 정밀도는 88.8%, F 점수는 91.9의 성능을 보인다. 성능 평가 지표 결과로 보아 단일 모델을 사용한 경우보다 스태킹 모델을 사용한 경우 악성 코드 탐지에 우수한 성능을 나타냄을 확인할 수 있다. 각 모델별 속도를 측정하기 위해 판별하기 전 현재 시각과 판별 후의 현재 시각을 출력해 전후 간의 시간 차를 이용한다. 측정 결과 ExtraTree가 0.6초, XGBoost가 0.1초, Stacking 모델이 2초로 스태킹 모델이 단일 모델보다 비교적 오랜 시간이 소요된다.

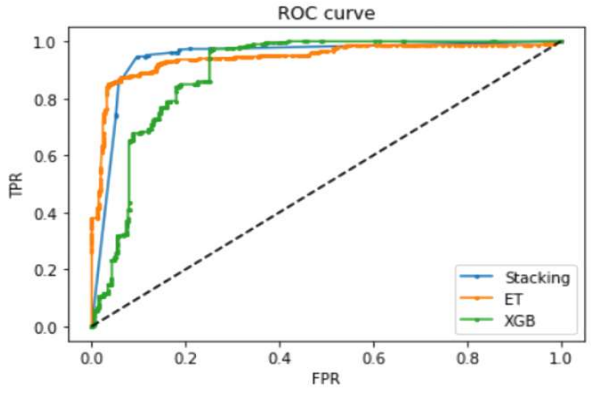

그림 5 ET, XGB 단일 모델과 스태킹 모델 ROC Curve

Fig. 5 ET, XGB Single Model and Stacking Model ROC Curve

5. Conclusion

본 논문에서는 정보 기술이 빠르게 발전함에 따라 많이 증가하고 있는 변종 악성 코드 기반 보안 위협에 빠르게 대응하고자 정적 분석과 스태킹 기법을 사용한 악성 코드 탐지 시스템을 제안한다. 악성 코드를 실행시키지 않고 빠르고 정확하

게 탐지하고자 정적 분석을 통해 악성 코드의 PE header의 특징을 추출하였으며 이를 pe_header, pe_packer, pe_top, n-gram과 같은 방법으로 가공한다. 가공한 특징들을 Logistic Regression, SVM, Random Forest, XG Boost와 같은 머신 러닝 단일 모델에 넣어 정확도를 비교해 가장 높은 정확도를 가지는 pe_packer 특징을 모델에 훈련할 입력 데이터로 선정한다. 정확도가 높은 모델을 구현하고자 단일 모델의 예측 결과를 이용하여 최종 예측값을 내는 스태킹 방식을 사용하여 모델을 구축한다. 실험 결과 스태킹 모델은 95.29%의 모델 정확도와 92.3%의 분류 정확도, 94.7%의 탐지율, 9.6%의 오탐율을 보인다. 머신 러닝 앙상블 기법인 스태킹 모델을 사용함으로써 단일 모델만을 사용하는 경우보다 악성 코드 탐지에 더 좋은 성능을 낸다는 것을 확인할 수 있지만, 모델별 속도는 스태킹 모델이 단일 모델보다 비교적 오랜 시간이 걸린다. 이를 해결하기 위해 Sub-model을 수정하거나 정제된 특징 데이터를 사용하는 등의 방법을 통해 속도 향상을 기대할 수 있다.

향후 변종 악성 코드를 빠르고 정확하게 탐지하는 그것뿐만 아니라, 더 나아가 악성 코드 패밀리 유형과 feature 특성에 따른 적합한 학습 모델 연구 등 다양한 연구 활동이 이뤄질 것이라 기대한다.

References

[1] AV-TEST, “The AV-TEST Security Report 2018/2019,” https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2018-2019.pdf, accessed May. 2020.

[2] CCTVNEWS, “The technical trends machine learning-based anti-virus,” http://www.cctvnews.co.kr/news/articleView.html?idxno=82784, accessed May. 2020.

[3] Ji-hee Ha, Su-jeong Kim, and Tae-jin Lee, “Feature Extraction using DLL/API Statistical Analysis and Malware Detection based on Machine Learning,” The Journal of Korean Institute of Communications and Information Sciences, vol. 43, no. 4, pp. 730-739, Apr. 2018.

[4] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, and G. Giacinto, “Novel feature extraction, selection and fusion for effective malware family classification,” Proceedings of the 6th ACM Conference on Data and Application Security and Privacy, pp. 183-194, Mar. 2016.

[5] Hyunjong Lee, Seongyul Euh, and Doosung Hwang, “API Feature Based Ensemble Model for Malware Family Classification,” Journal of The Korea Institute of Information Security & Cryptology, vol. 29, pp. 531-539, Jun. 2019.

[6] “ClaMP” https://github.com/urwithajit9/ClaMP, accessed Mar. 2020.

[7] D. H. Wolperk, “Stacked generalization,” Neural Networks, vol. 5, pp. 866-871, 1992.

[8] “Vecstack,” https://github.com/vecxoz/vecstack, accessed Mar. 2020.

저자소개

조진영 (Jin-young Cho)

상명대학교 정보보안공학과 재학
관심분야 : 보안시스템, 인공지능
이메일: chojinyoung22@naver.com

고은기 (Eun-gi Ko)

상명대학교 정보보안공학과 재학
관심분야 : 보안소프트웨어, 악성코드
이메일: lovee2756@naver.com

유혜빈 (Hye-bin Yoo)

상명대학교 정보보안공학과 재학
관심분야 : 정보보호, 악성코드, 네트워크 보안
이메일: hyebin218@naver.com

조미리 (Mi-ri Cho)

상명대학교 정보보안공학과 재학
관심분야 : 보안시스템, 악성코드, 웹
이메일: miri77930@naver.com

서창진 (Chang-jin Seo)

부산대학교 멀티미디어 공학박사
경력 : 센서기술연구소 연구원, 상명대학교 정보보안공학과 교수
관심분야 : 인공지능, 보안시스템, 악성코드, 딥러닝
이메일 : cjseo@smu.ac.kr