Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.30, NO.2, Apr. 2020 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2020.30.2.169

Adaptive Boosting을 사용한 패커 식별 방법 연구*

장 윤 환,[1†] 박 성 준,[1] 박 용 수[2‡]
[1,2]한양대학교 (대학원생, 교수)

Packer Identification Using Adaptive Boosting Algorithm*

Yun-Hwan Jang,[1†] Seong-Jun Park,[1] Yongsu Park[2‡]
[1,2]Hanyang University(Graduate student, Professor)

요 약

악성코드 분석은 컴퓨터 보안의 중요한 관심사 중 하나로 분석 기법의 진보는 컴퓨터 보안의 중요 사항이 되었다. 기존에는 악성코드를 탐지할 때 Signature-based 방식을 사용하였으나 패킹된 악성코드의 비율이 높아지면서 기존 Signature-based 방식으로는 탐지에 어려움이 많아 졌다. 이에, 본 논문에서는 머신러닝을 사용하여 패킹된 프로그램의 패커를 식별하는 방법을 제안한다. 제안한 방법은 패킹된 프로그램을 파싱하여 패커를 특정 지을 수 있는 특정 PE 정보를 추출하고 머신러닝 모델 중 Adaptive Boosting 알고리즘을 사용하여 패커를 식별한다. 제안한 방법의 정확도를 확인하기 위해 12가지 종류의 패커로 패킹된 프로그램 391개를 수집하여 실험하였으며, 약 99.2%의 정확도로 패커를 식별하는 것을 알 수 있었다. 또한, Signature-based PE 식별 도구인 PEiD와 기존 머신러닝을 사용한 방법으로 식별한 결과를 제시하였으며, 본 논문에서 제안한 방법이 기존의 방법보다 패커를 식별하는데 정확도와 속도면에서 더 뛰어난 성능을 발휘하는 것을 알 수 있다.

ABSTRACT

Malware analysis is one of the important concerns of computer security, and advances in analysis techniques have become important for computer security. In the past, the signature-based method was used to detect malware. However, as the percentage of packed malware increased, it became more difficult to detect using the conventional method. In this paper, we propose a method for identifying packers of packed programs using machine learning. The proposed method parses the packed program to extract specific PE information that can identify the packer and identifies the packer using the Adaptive Boosting algorithm among the machine learning models. To verify the accuracy of the proposed method, we collected and tested 391 programs packed with 12 types of packers and found that the packers were identified with an accuracy of about 99.2%. In addition, we presented the results of identification using PEiD, a signature-based PE identification tool, and existing machine learning method. The proposed method shows better performance in terms of accuracy and speed in identifying packers than existing methods.

Keywords: Computer security, Information security, Malware detection, Machine learning.

I. 서 론

패커(Packer)는 원본 프로그램을 다양한 방법의 압축 기술과 난독화, 안티-리버싱(Anti-Reversing) 기법을 적용하여 동일한 동작을 수행하는 새로운 파일로 변환하는 소프트웨어 프로그램이다. 패커는 일

Received(11. 28. 2019), Modified(02. 14. 2020), Accepted(03. 26. 2020)
* 본 연구는 한국연구재단 연구과제(2017R1D1A1B030295 50) 지원으로 수행하였습니다.
† 주저자, dbsghksdlwkd@hanyang.ac.kr
‡ 교신저자, yongsu@hanyang.ac.kr(Corresponding author)

반적으로 상용 프로그램의 중요한 정보를 보호하기 위해 사용되지만 악성코드 개발자는 분석과 탐지를 회피하기 위해 사용한다[1]. 많은 종류의 패커가 온라인에 존재하며 동일한 원본 프로그램으로 패커의 설정만 변경하여 많은 변형된 파일을 생성 할 수 있다. 바이러스 백신 및 보안 소프트웨어 평가 기관인 AV-TEST 사에 따르면 전체 악성코드의 92% 이상이 패킹(압축 및 난독화)이 적용되어 있다[2]. 패킹된 프로그램을 분석할 때 패킹된 패커의 종류를 안다면 안티-리버싱 기법을 우회하거나 OEP[3-5]를 식별하여 악성코드를 분석하는데 도움이 된다.

기존의 Machine Learning-based 식별 방법에서는 PE 프로그램의 전체 Byte 값을 이용한다. 이는 패킹된 원본 프로그램과 패커의 종류나 옵션에 따라 압축 데이터가 변경될 수 있다는 문제점이 존재한다. 따라서 본 논문에서는 패킹된 프로그램의 PE header 정보와 머신러닝을 사용하여 패커를 식별하는 방법을 제안한다.

제안한 방법은 패킹된 프로그램의 PE 정보 중 패커의 정보와 관련된 특정한 필드의 값을 추출하고, 추출한 데이터를 전처리 과정을 적용한 후, Adaptive Boosting 알고리즘으로 학습하여 패커를 식별한다. 본 논문에서 제안한 방법의 정확성을 확인하기 위해서 Tuts4You[6]와 VIRUSTOTAL[7]에서 12종의 패커로 패킹된 391개의 Windows 32bit 프로그램을 수집하여 패커 식별 실험을 수행하였다. 또한, 기존 Signature-based 식별 방법인 PEiD[8]와 Jung ByeongHo et al.[9]의 방법을 사용하여 식별 실험을 수행한 결과를 제시하였다. 제안한 방법의 정확도가 PEiD보다 39.7%, Jung ByeongHo et al.의 방법보다 11.1% 높은 99.2%로 패커를 식별하는 것을 확인하였다.

II. 관련 연구

패커를 식별하는 방법에는 Signature-based 식별 방법과 Machine Learning-based 식별 방법이 있다. Signature-based 식별 방법은 프로그램의 특정한 정보를 사용하여 서명을 생성하고 미리 저장한 서명들과 비교하여 식별한다. Machine Learning-based 식별 방법은 프로그램을 Byte Sequence나 Byte/Markov Plot 등 일련의 변환 과정을 거쳐 수치로 변환하고 머신러닝 모델을 사용하여 식별을 하는 것이다.

Hai, Nguyen Minh et al.[10]은 패킹된 프로그램에서 난독화 기법의 발생 빈도를 메타데이터화 해서 패커를 식별하는 방법을 제안한다. 에뮬레이터인 BE-PUM(Binary Emulator for Pushdown Model)[11]에서 Concolic Testing을 사용하여 패킹된 프로그램을 디스어셈블한 정보와 제어 흐름 그래프(Control Flow Graph)를 추출한다. 미리 정의한 난독화 기법(14가지 종류)의 기준과 제어 흐름 그래프의 난독화 기법(특정 Instruction)을 비교하여 난독화 기법을 식별한다. 식별한 난독화 기법의 빈도수를 측정한 빈도수에 Chi-square test를 적용하여 포함된 값의 임계점을 기준으로 패커를 식별한다.

Naval, Smita et al.[12]의 방법은 MSA(Multiple Sequence Alignment)로 프로그램에 대한 서명을 생성하고 식별한다. MSA로 서명을 생성하기 위해서는 프로그램을 OllyDbg 디버거[13]를 사용하여 디스어셈블 하고, Two molecular sequences 알고리즘을 이용하여 Score matrix와 Traceback matrix를 계산하기 위한 Local alignment를 한다. Traceback matrix의 최댓값 위치에서 0이 될 때까지 추적하여 Aligned sequences를 생성하고, MSA를 사용하여 패커의 특정 클래스 유사성을 나타내는 서명을 생성한다. 생성된 Score와 서명으로 패커를 식별한다.

Ban Tao et al.[14]의 방법은 Signature-based 접근법과 Machine Learning-based 접근법을 사용하여 패커를 식별한다. 제안한 방법은 학습한 사전 정보와 유클리드 거리의 유사성 매트릭스의 “Most likely” 매칭을 사용하여 패커를 식별한다. “Most likely” 매칭을 위해 PEiD의 서명 기법을 사용하여 프로그램에 서명이 포함되어 있는지 확인하고, k-NN 알고리즘(K=1)으로 거리가 가장 가까운 패커 그룹의 종류로 식별한다. 다만, k-NN 알고리즘은 복잡한 프로그램에 대해서는 성능이 저하 될 수 있기 때문에 유사도 matrix를 Levenshtein Distance 커널의 SVM(Support Vector Machine) 알고리즘을 사용하여 서명 매칭에서 발생하는 오류 허용 방법을 제안하였다.

KANCHERLA Kesav et al.[15]은 프로그램을 Byte plot과 Markov plot으로 변환하고 머신러닝의 SVM 알고리즘을 사용하여 패커를 식별하는 방법을 제안한다. 프로그램을 Byte 배열로 변환하고, 프로그램의 크기를 참고하여 행의 크기를 정한 2

차원 Byte Plot으로 변환 한다. Byte plot을 Markov plot으로 변환 할 때에는 상태 전환을 나타내는 Markov chain을 사용하며, 256*256의 크기로 변환한다. Markov plot의 가중치와 방향 벡터에 대한 데이터를 SVM 알고리즘으로 학습하여 패커를 식별한다.

Jung ByeongHo et al.[9]에서는 프로그램을 Byte sequence를 이용한 엔트로피(entropy) 기반 스코어링 방법으로 암호화된 섹션을 탐지하고, 머신러닝 기반의 학습 알고리즘을 사용하여 패커를 식별하는 방법을 제안한다. 이 방식은 크게 암호화된 섹션 탐지와 패커 식별로 구분 할 수 있으며, 암호화된 섹션 탐지는 패커 식별보다 우선 수행되어야 한다. 암호화된 섹션을 탐지는 프로그램의 헤더(header) 정보를 이용하여 섹션(section)을 식별한다. 식별한 섹션별로 시스템에서 장애 또는 불확실성의 정도를 나타내는 엔트로피 값을 이용한 N-Byte sliding windows를 기반으로 한 Byte sequence를 구하여 엔트로피 스코어를 계산한다. 계산된 엔트로피 스코어가 가장 높은 섹션을 암호화된 섹션으로 간주한다. 패커 식별에는 암호화된 섹션 탐지에서 구한 엔트로피 값의 시퀀스에 대해 표준 편차, 평균 점수 및 최소 및 최댓값을 계산한다. 또한, Byte frequency를 계산하여 Feature vectors를 생성하고, Feature vectors와 Random forest 및 Gradient boosting 알고리즘을 사용하여 패커를 식별한다.

III. 제안하는 패커 식별 방법

패킹된 프로그램은 패커 제작자가 작성한 코드(unpacking code)를 실행한 후, 원본 프로그램의 코드를 실행한다. 따라서 패킹된 프로그램은 패커 제작자의 코드와 원본 프로그램의 코드 및 정보가 모두 포함되어 있으며, 패커마다 고유의 동작을 하는 unpacking code가 실행될 수 있도록 PE header의 정보가 변경되어 있다.

기존 Machine Learning-based 방법들은 프로그램을 파싱(parsing)하여 학습하는 것이 아닌 프로그램의 전체 Byte 값을 사용한다[9,14,15]. 이는 패킹된 원본 프로그램과 패커의 종류나 옵션에 따라 압축 데이터가 변경될 수 있다는 문제점이 존재한다. [9]의 기법을 사용하여 패킹된 프로그램의 엔트로피 값에 대한 표준 편차, 평균 점수 및 최소 및 최댓값을 계산하면 동일 종류의 패커에서도 압축률이나 파일의 크기, 파일 종류에 따라 매우 상이한 결과를 보인다.

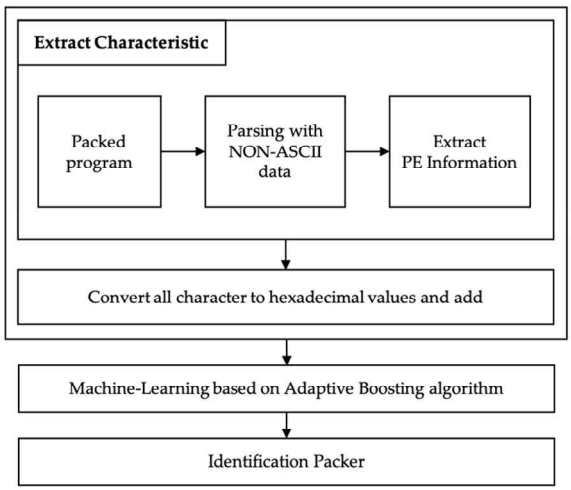

Fig. 1. Overview of the proposed method

제안한 기법의 패커를 식별하는 전체 순서는 Fig. 1.과 같다. 패킹된 프로그램을 파싱하여 unpacking code 및 섹션과 관련된 특징에 대한 정보를 추출하고, 추출한 정보에 필드를 확인하여 NON-ASCII 또는 ASCII 데이터를 16진수 Byte 값으로 변환하는 전처리 과정을 수행한다. 이후, 추출된 데이터를 머신 러닝 알고리즘인 Random forest와 Adaptive Boosting 모델을 사용하여 패커를 식별한다.

3.1 PE 파싱 (parsing)

PE는 프로그램이 메모리에 로드 될 때 참조하는 정보를 포함하고 있다. PE에 포함된 데이터 중에서 NT header의 'SizeOfCode'와 'BaseOfCode', 'BaseOfData', 'NumberOfSections' 필드 그리고 각 섹션의 Section header의 'Name', 'SizeOfRawData'는 패커와 밀접한 관련이 있는 PE 필드이다.

NT header에 있는 'SizeOfCode'는 코드가 포함되어 있는 섹션의 전체 크기, 즉, unpacking code 영역의 크기를 의미한다. 'BaseOfCode'는 Code 섹션이 시작하는 첫 번째 Byte 주소로 Unpacking Code의 맨 처음 주소이다. 'BaseOfData'는 Data 섹션의 시작 주소로 Unpacking code가 사용하는 데이터의 시작 주소

이다. Section header에 있는 'Name'은 해당 섹션의 이름이며 'SizeOfRawData'는 해당 섹션의 크기를 의미한다.

3.2 전 처리 (pre-processing)

일부 필드의 경우, ASCII 값 또는 NON-ASCII 값이 포함되어 있다. Section header의 'Name' 필드는 어떠한 명시적인 규칙이 존재하지 않는다. 대표적으로 UPX[16]의 경우 해당 필드가 'UPX1' 또는 'UPX2'로 ASCII 문자열로 이루어져 있지만, VMProtect[17]의 경우 ASCII 문자열인 'vmp0'과 'vmp1' 그리고 NON-ASCII '2F 34 00 00', '2F 31 39 00' 등이 동시에 존재한다. 따라서, NON-ASCII 또는 ASCII로 이루어진 필드는 16진수 Byte 값으로 변환하고, 2 Byte씩 분할하여 모두 합한 값을 16진수 값으로 사용한다.

3.3 Machine-Learning

머신러닝을 기반으로 한 학습 모델은 파싱 후 전처리 과정을 수행한 데이터를 이용하여 생성된다. 머신러닝 모델은 Random forest 알고리즘과 Adaptive Boosting 알고리즘을 사용하였다. 실험 결과는 Random forest 알고리즘과 Adaptive Boosting 알고리즘의 성능이 큰 차이는 존재하지 않았다. 다만, Adaptive Boosting 알고리즘이 약 1% 높은 성능을 보였다.

IV. 실 험

4.1 실험 환경

본 논문에서 제안한 방법의 정확성을 확인하기 위해 Tuts4You[6]와 VIRUSTOTAL[7]에서 12종의 패커로 패킹된 Windows 32bit PE 프로그램 391개를 수집하여 실험을 진행하였다. 실험을 한 환경의 CPU는 Intel Xeon E5-i7이며 메모리는 DDR3 64GB, 그래픽 카드는 NVIDIA GeForce 1050을 사용하였다. 운영체제는 Ubuntu Server 18.04이며, Cuda 10.1, TensorFlow 1.15, Python 2.7을 사용하였다.

수집된 391개의 패킹된 PE 프로그램은 실험을 위해 학습 Dataset과 테스트 Dataset로 각 273개, 118개(학습 Dataset : 70%, 테스트 Dataset : 30%)로 분할하였다. 분할된 학습 Dataset과 테스트 Dataset에 대한 패커별 비율은 아래 Table 1.과 같다.

Table 1. Ratio of Packer to train and test dataset.

Packer	Rate of train dataset	Rate of test dataset
ACProtect	7.69%	7.63%
ASProtect	10.26%	10.17%
Armadillo	26.01%	26.27%
Enigma	5.86%	5.93%
Obsidium	4.76%	5.08%
PECompact	5.86%	5.93%
Petite	5.86%	5.93%
Safengine	2.93%	2.54%
Themida	10.99%	11.02%
UPX	4.76%	5.08%
VMProtect	10.99%	10.17%
Yoda's Protector	4.03%	4.24%
Total	100%	100%

4.2 실험 구현

제안한 방법은 PE를 파싱하는 파서 프로그램과 머신러닝으로 학습하는 두 개의 프로그램으로 구현되었으며, 파서 프로그램의 경우 Python을 사용하여 구현하였으며, Random forest 알고리즘과 Adaptive Boosting 알고리즘의 경우 Scikit-learn 라이브러리[18]의 Sklearn 패키지를 사용하여 구현하였다.

4.3 실험 결과

본 논문에서 제안한 기법과 PEiD, Jung ByeongHo et al.[9]의 기법으로 패커 식별을 수행한 식별률은 Table 2.와 같다. 전체 정확도와 패커별 정확도는 f1-score 값을 사용하였다.

PEiD의 경우 평균 59.5%의 정확도를 보이며, ASProtect, Armadillo, PECompact 등 잘 알려져 있거나 널리 사용되는 패커의 경우 높은 식별률을 보인다. 그러나 Safengine, VMProtect 등 복

Table 2. Experimental results on the accuracy of the PEiD, [9]'s method and the proposed method.

Packer	PEiD [8]	[9]'s method	Proposed method using AB	Proposed method using RF
ACProtect	33%	90%	100%	94%
ASProtect	97%	80%	100%	98%
Armadillo	99%	93%	100%	96%
Enigma	30%	83%	100%	100%
Obsidium	36%	92%	91%	100%
PECompact	100%	71%	100%	93%
Petite	56%	93%	100%	100%
Safengine	0%	100%	100%	100%
Themida	69%	82%	100%	100%
UPX	94%	100%	89%	100%
VMProtect	0%	100%	97%	100%
Yoda's Protector	100%	33%	91%	100%
Accuracy	59.5%	88.1%	99.2%	98.3%

잡한 동작을 수행하는 패커에 대해서는 식별을 하지 못하거나 매우 낮은 정확성을 보인다. [9]의 기법은 PECompact와 Yoda's Protector에서 다른 패커보다 낮은 식별률을 보이지만 PEiD보다 높은 정확도인 88.1%의 정확도를 보인다.

제안한 기법은 Random forest(RF) 알고리즘을 사용한 경우 평균 98.3%의 정확도를 보였으며, Adaptive Boosting(AB) 알고리즘을 사용한 경우 평균 99.2%의 정확도를 보였다. 제안한 기법은 머신러닝에 사용된 알고리즘에 큰 영향을 받지 않으며 대부분의 패커에 대하여 높은 식별률을 보이는 것을 알 수 있다.

Table 3.은 머신러닝을 사용하는 기법에 대한 실행 속도를 측정한 것이다. 머신러닝을 사용하는 기법은 전처리 과정과 학습(Learning) 과정으로 분할하여 측정하였으며, 전처리 과정과 학습 과정에 소요된 시간을 합하여 전체 소요시간을 측정하였다. 제안한 기법의 경우 학습 과정은 Random forest와 Adaptive Boosting 알고리즘 모두 미세한 속도 차이(0.01초 이하)를 보였기에 Adaptive Boosting 알고리즘을 기준을 하였다.

Table 3. Execution time for [9]'s method and the proposed method.

	[9]'s method	Proposed method
Pre-processing	46m 23.207s	0.127s
Learning	0.099s	0.040s
Total	46m 23.307s	0.167s

[9]의 기법은 전처리 과정에서 46분 23.207초, 학습 과정에서는 0.099초로 총 46분 23.307초가 소요되었다. 반면 제안한 기법은 전처리 과정에서 0.127초, 학습 과정에서는 0.040초로 총 0.167초가 소요되었다.

Table 4.는 1개의 패킹된 프로그램에 대해 전처리 과정을 수행할 때 소요된 시간이다. 제안한 기법의 경우 최대 0.001초이지만, [9]의 기법은 패킹된 프로그램의 크기가 커짐에 따라 소요 시간이 증가하는 것을 알 수 있다.

Fig. 2.는 PEiD를 사용하여 패커를 식별한 결과를 비율로 변경하여 시각화 한 것이다. Fig. 2.의 X축은 PEiD를 사용하여 식별한 패커이며, Y축은 입력한 패커이다. X축의 'etc'는 12종류의 패커가 아닌 다른 종류의 프로그램으로 식별된 경우이다.

Fig. 3.는 [9]의 기법을 사용하여 패커를 식별

Table 4. Execution time for pre-processing 1 item in [9]'s method and the proposed method.

	[9]'s method	Proposed method
〈 100 KB	〈 0.890s	〈 0.001s
〈 1 MB	〈 9,922s	
〈 10 MB	〈 192.690s	

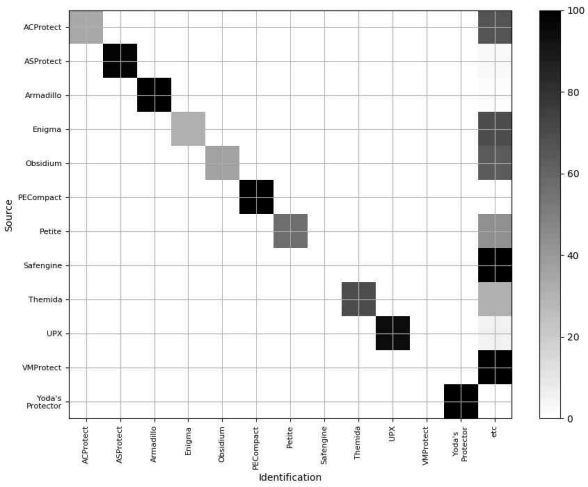

Fig. 2. Packer identification visualization in PEiD.

하였을 때, Estimators 값을 1에서 100까지 변경하며 정확도를 측정한 결과이다. Estimators 값이 1부터 100까지의 변함에 따라 약 75% ~ 88%의 정확도를 갖는다.

Fig. 4.는 [9]의 기법을 사용하여 테스트 데이터를 식별한 결과를 비율로 변경하여 시각화 한 것이다. X축과 Y축의 정보는 Fig. 2.와 동일하지만 [9]의 기법과 제안한 기법은 학습한 종류의 패커 내에서 식별되기 때문에 etc가 존재하지 않는다.

Fig. 5.는 제안한 기법을 사용하여 패커를 식별 하였을 때, Estimators 값을 1에서 100까지 변경하며 정확도를 측정한 결과이다. Adaptive Boosting

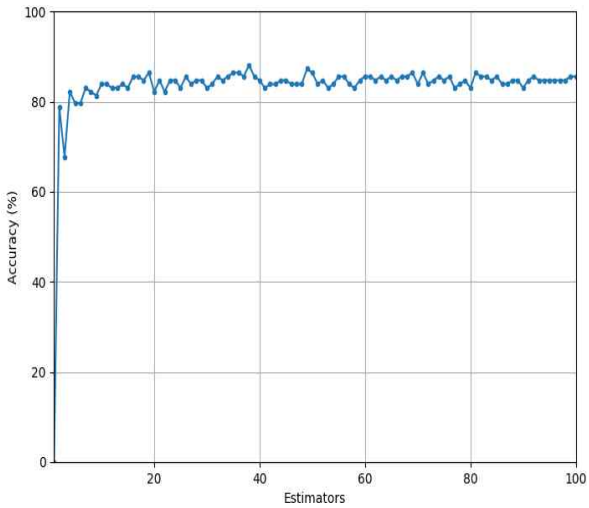

Fig. 3. Accuracy according to 'Estimators' value in [9]'s method.

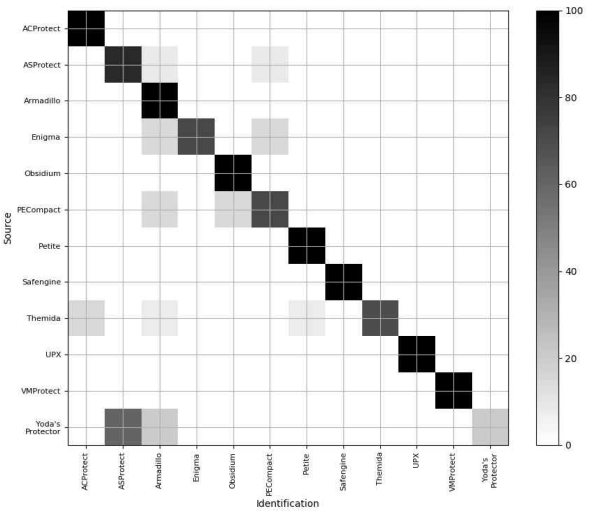

Fig. 4. Packer identification visualization in [9]'s method.

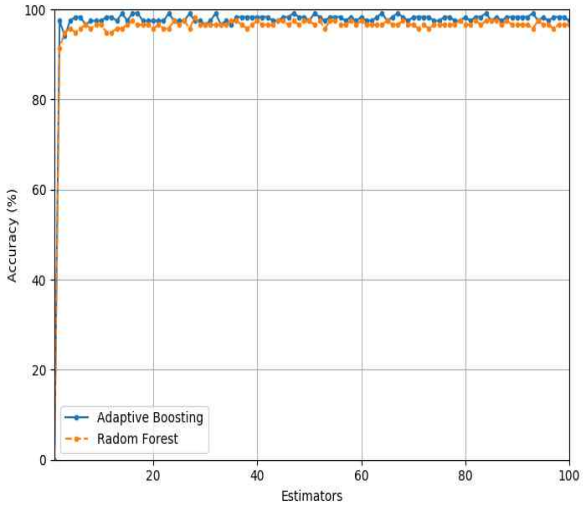

Fig. 5. Accuracy according to 'Estimators' value in proposed method.

알고리즘은 Estimators 값이 2일 때 94%를 제외하고, 1에서 100까지의 Estimators 값에 대해 모두 97% 이상의 정확도를 갖는다. Random forest 알고리즘의 경우에는 Estimators 값이 1일 때 91%, 2에서 100에 대해 모두 95% 이상의 정확도를 갖는다.

Fig. 6.는 제안한 기법에서 Adaptive boosting 알고리즘을 사용하여 패커를 식별한 결과를 비율로 변경하여 시각화 한 것이다. X축과 Y축의 정보는 Fig. 3.과 동일하다.

Fig. 7.과 Fig. 8은 [9]의 기법과 제안한 기법에서 Adaptive boosting 알고리즘으로 패커를 식별

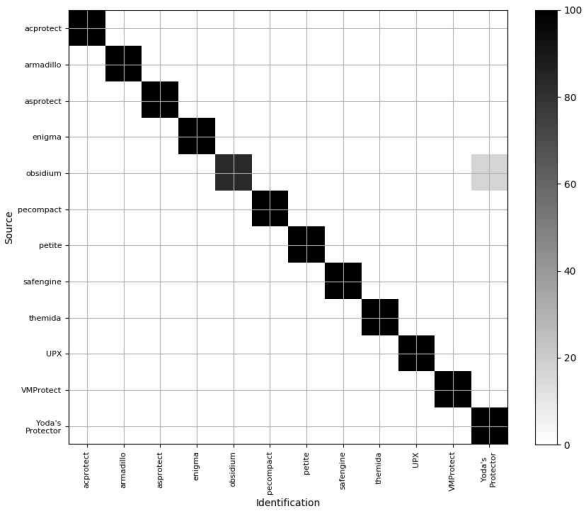

Fig. 6. Packer identification visualization in proposed method using Adaptive boosting.

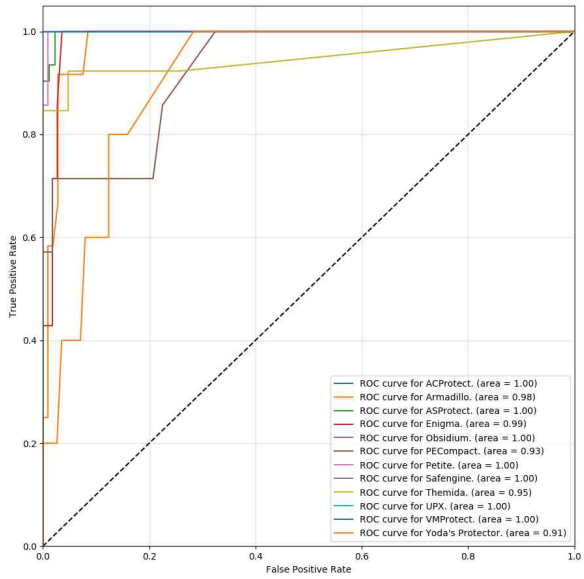

Fig. 7. ROC curve in [9]'s method.

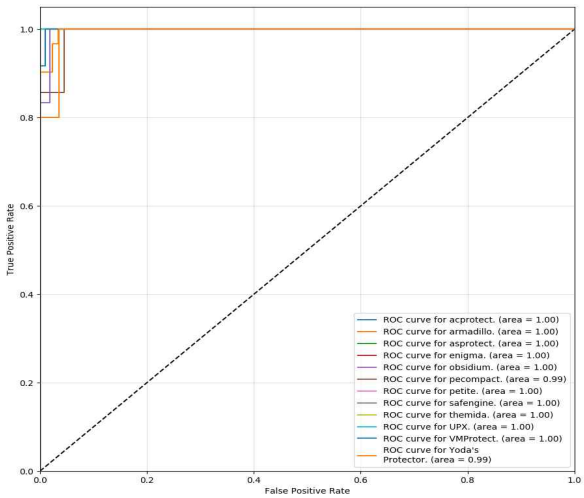

Fig. 8. ROC curve in proposed method using Adaptive boosting.

하였을 때의 ROC curve이다. 제안한 기법이 [9]의 기법보다 ROC 그래프의 커브가 좌측 상단으로 밀집되어 있는 것을 알 수 있다.

V. 결 론

본 논문에서는 프로그램에서 PE 정보를 이용하여 패커를 식별하는 새로운 방법을 제안하였다. 제안한 방법에서는 머신러닝을 사용하였으며, 식별에 대한 정확도를 확인하기 위해서 12가지 종류의 패커로 패킹된 Windows 32bit 프로그램을 사용하였다. 391개의 수집된 프로그램을 사용하여 99.2%의 정확도를 갖는 것을 알 수 있다.

대표적인 Signature-based 식별 방법인 PEiD에서는 59.5%의 정확도로 패커를 식별하였다. 또한, Machine Learning-based 식별 방법인 [9]의 기법으로는 88.1%의 정확도를 보였다.

본 논문에서 제안한 방법은 PEiD보다 패커를 식별하는데 39.7% 더 정확한 것을 알 수 있었으며, 기존 Machine Learning-based 식별 방법인 [9]의 기법보다 11.1% 더 정확한 것을 알 수 있었다. 또한, 제안한 기법을 Random forest 알고리즘과 Adaptive boosting 알고리즘을 사용하여 실험한 결과 각 98.3%와 99.2%의 정확도로 인하여 특정 머신러닝 모델에 의해 정확도가 높아지는 것이 아니라는 것을 알 수 있다. 뿐만 아니라 식별 속도에서도 기존 Machine Learning-based 식별 방법보다 매우 빠르게 동작하는 것을 알 수 있다.

References

[1] Improving proactive detection of packed malware, "detection packed malware", https://www.virusbulletin.com/virusbulletin/2006/03/improving-proactive-detection-packed-malware, 10, Jun, 2019

[2] T. Brosch and M. Morgenstern. "Runtime packers: The hidden problem," Black Hat USA, 2006

[3] R. Isawa, D. Inoue and K. Nakao. "An original entry point detection method with candidate-sorting for more effective generic unpacking," IEICE TRANSACTIONS on Information and Systems, vol. E98-D, no. 4, pp. 883-893, Apr. 2015

[4] S. D'ALESSIO and S. MARIANI. "PinDemonium: a DBI- based generic unpacker for Windows executables," Black hat, Apr. 2016

[5] Kim Gyeong-Min, Park Juhyun, Jang Yun-Hwan and Park Yongsu. "Efficient Automatic Original Entry Point Detection," Journal of

Information Science & Engineering, vol. 35, no. 4, pp. 887-902, Jul. 2019
〔6〕 Tuts4You, https://tuts4you.com, 02, May, 2019
〔7〕 VIRUSTOTAL, https://www.virustotal.com, 31, May, 2019
〔8〕 PEiD, "peid", https://www.aldeid.com/wiki/PEiD, 13, Dec, 2019
〔9〕 B. Jung, S.I. Bae, C. Choi and E.G. Im. "Packer identification method based on byte sequences," Concurrency and Computation: Practice and Experience, 32.8, e5082, Oct. 2018
〔10〕 N.M. Hai, M. Ogawa and Q.T. Tho. "Packer identification based on metadata signature," In: Proceedings of the 7th Software Security, Protection, and Reverse Engineering/ Software Security and Protection Workshop, pp. 1-11, Dec. 2017
〔11〕 BE-PUM, "BE-PUM", https://github.com/NMHai/BE-PUM, 13, Dec, 2019
〔12〕 S. Naval, V. Laxmi, M.S. Gaur and P. Vinod. "Spade: Signature based packer detection," In Proceedings of the First International Conference on Security of Internet of Things. ACM, pp. 96-101, Aug. 2012
〔13〕 OllyDbg Debugger, "ollydbg", http://www.ollydbg.de, 13, Dec, 2019
〔14〕 T, Ban, R. Isawa, S. Guo, D. Inoue and K. Nakao. "Application of string kernel based support vector machine for malware packer identification," In The 2013 International Joint Conference on Neural Networks (IJCNN). IEEE, pp. 1-8, Aug. 2013
〔15〕 K. Kancherla, J. Donahue and S. Mukkamala. "Packer identification using Byte plot and Markov plot," Journal of Computer Virology and Hacking Techniques, vol. 12, no. 2, pp. 101-111, Sep. 2016
〔16〕 UPX, "upx", https://upx.github.io, 19, Dec, 2019
〔17〕 VMProtect, "vmprotect", https://vmpsoft.com, 19, Dec, 2019
〔18〕 scikit-learn, "sklearn", https://scikit-learn.org/stable, 20, Dec, 2019

〈저자소개〉

장 윤 환 (Yun-Hwan Jang) 학생회원
2015년 8월: 홍익대학교 정보·컴퓨터공학부 컴퓨터공학 졸업
2020년 2월: 한양대학교 대학원 정보보안학과 석사
〈관심분야〉 컴퓨터 보안, 네트워크 보안, 모바일 보안, 웹 보안

박 성 준 (Seong-Jun Park) 학생회원
2019년 2월: 한양대학교 컴퓨터·소프트웨어학과 졸업
2019년 3월~현재: 한양대학교 대학원 컴퓨터·소프트웨어학과 석사과정
〈관심분야〉 악성코드 분석, 소프트웨어 보안, 프로그램 분석

박 용 수 (Yongsu Park) 종신회원
1996년: KAIST 전산학과 졸업(학사)
1998년: 서울대학교 대학원 컴퓨터공학과 졸업(석사)
2003년: 서울대학교 대학원 전기컴퓨터공학부 졸업(박사)
2003년~2004년: 서울대학교 자동제어특화연구센터 연수연구원
2005년~현재: 한양대학교 소프트웨어전공 교수
〈관심분야〉 컴퓨터 보안, 인터넷 보안