AI-based Malware Analysis

FINAL PROJECT

Report

Subject	COMPUTER SCIENCE Capstone PBL
Professor	임을규 교수님(Eul Gyu Lim)
Submission Date	2022.06.14.
University	Hanyang University
School	College of Engineering
Department	Department of Computer Science
Student ID	**Name**
2019009261	최가온(Gaon Choi)
2019027192	김현수(Hyeonsu Kim)

Ⅰ. Dataset Generation

1. convert_to_image.py

(출처)
https://github.com/hyunas1996/Malware_Classification/blob/master/Source_Code/Malware_Classification_with_VGG/convert_to_image.py

.vir 파일을 바이트 단위로 불러와 grey-scale 이미지로 변환하는 코드이다. 해당 부분과 아래의 resize 이미지로 변환하는 코드를 통해 데이터셋을 생성하였고, 이전에 악성코드를 시각화하여 표현한 후 이를 기반으로 분류작업을 하는 데에서 영감을 받아 해당 자료를 사용하였다.

2. resize_image.py

(출처)

https://github.com/hyunas1996/Malware_Classification/blob/master/Source_Code/Malware_Classification_with_VGG/resize_image.py

위에서 변환된 이미지는 width가 32로 고정이 되어있고, height가 파일의 크기에 비례하여 무한히 자라나는 구조이다. 이러한 가변 길이는 CNN 등에서 적합하지 않다. 따라서, 현재의 이미지를 resize하여 32 * 32 크기로 모두 통일하였다.

(이 부분 이후에는 모두 직접 소스코드를 작성함.)

3. virustotal_scrapper.py

mdataset의 hash 4만개에 대한 VirusTotal의 검색 결과를 얻어오는 코드이다. VirusTotal의 경우 Client Side Rendering 방식으로 사이트 구조가 복잡한 편이기 때문에, 크롬 브라우저의 http request-response를 기록해 주는 selenium-wire 라이브러리를 사용하였다. selenium으로 VirusTotal에 접속하여 hash를 검색하고, 사이트가 렌더링 될 때 서버와 통신하는 json 파일들을 가로채 fetch_result 폴더에 저장한다. 사이트에서 Recapcha를 요구하거나 Too Many Request 에러가 발생하는 경우를 처리하는 로직도 포함되어 있다.

전체 json 파일

https://drive.google.com/file/d/1aZlnd0CRarg79SAFL2jmsmgYuHd9hziZ/view?usp=sharing

4. check_progress.py

Recapcha나 사이트 자체의 오류로 인해 json 파일이 불완전한 경우가 존재한다. check_progress.py는 이런 경우를 감지해 todo 목록에 넣어 virustotal_scrapper.py가 해당 hash에 대한 데이터를 다시 다운로드 할 수 있도록 한다.

5. build_dataset.py

앞서 얻은 json은 복잡한 형식이므로 사용하기 어렵다. build_dataset.py는 파일의 type_description, packer, entropy와 같이 흥미로운 몇 가지 feature를 추출하여 csv 파일로 저장한다.

생성된 파일

https://drive.google.com/file/d/1mVv0SUHHyL1KTD7uteeD5tnO1MD-MIBp/view?usp=sharing

Ⅱ. Malware Binary Classification

1. json_search.py

바이러스 정보들이 담겨있는 train.csv, test1.csv, test2.csv 파일을 불러와서 이 파일들에서 각각의 해시값과 그 해시값에 대응되는 악성여부 값(0 또는 1)을 받는다. Csv 파일의 형식상 8번째 column에 해시값이, 9번째 column에 0 또는 1의 값이 저장되어 있었다.

이 값들중 해시값만을 추출하여 .txt 파일에 write하여 저장하는 코드이다.

2. json_malware.py

Resize된 32 * 32 크기의 이미지를 malware와 non-malware 두 개의 폴더에 나눠서 담아야했다. 이 과정 이후 DataLoader를 이용하여 데이터를 처리할 생각이었다. 이 코드는 csv 파일에서 각 해시값을 검색하여 감염 여부(0 또는 1)를 읽어와 그 값을 보고 malware 폴더 또는 non-malware 폴더로 이미지 파일을 나누어 담는 코드이다.

3. malware_detection.ipynb

ResNet50을 기반으로 32 * 32 크기의 이미지가 주어지면 여기에서 malware인지 benign인지를 구분해주는 binary classification 모델을 만들었다. 이 코드는 해당 모델의 구조, 데이터 불러오기, Train, Eval 등 전반적인 부분을 모두 담고 있다.

4. train.py

build_dataset.py로 생성된 csv 파일을 대상으로, lightgbm 라이브러리를 이용해 decision-tree를 생성하는 코드이다. 생성된 decision-tree을 이용해 test set을 predict하고, sklearn 라이브러리로 confusion matrix, accuracy score, precision score, recall score 등을 알아낸다.

Test set 1

Confusion Matrix

[[4531 354]

 [51 4834]]

accuracy: 0.9585, precision: 0.9318, recall: 0.9896, f1: 0.9598, roc_auc: 1.0000

Test set 2

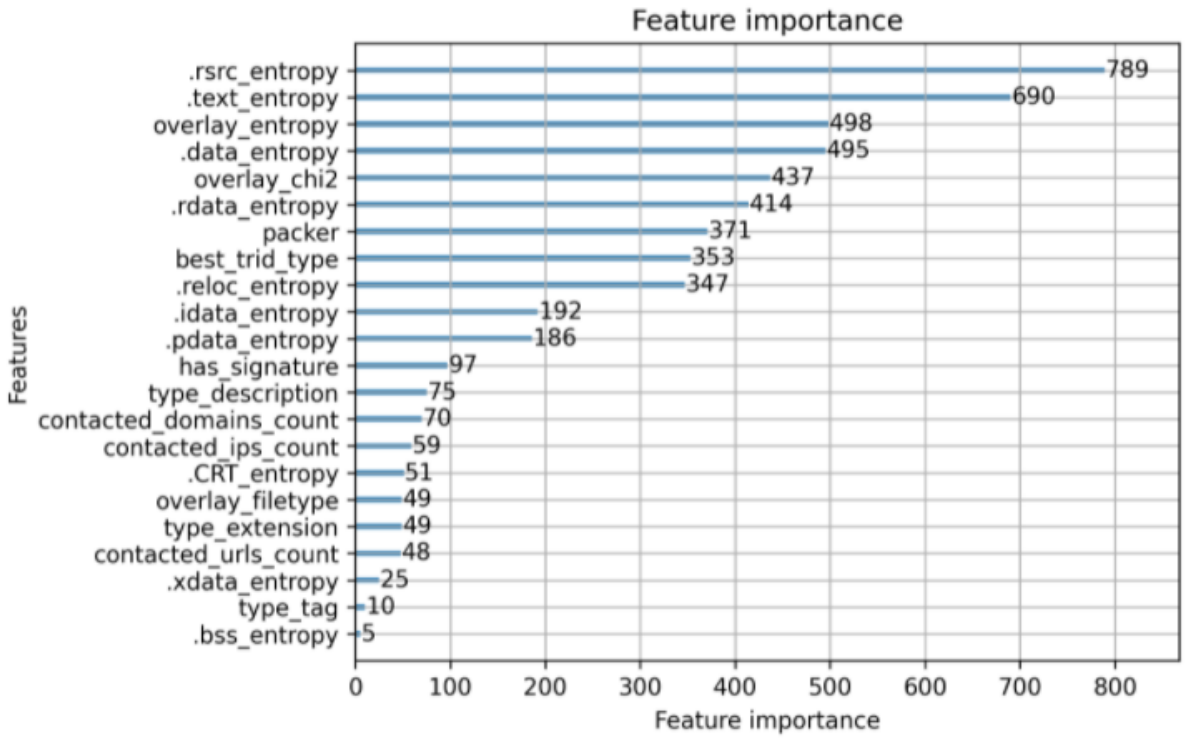
Confusion Matrix

[[4226 547]

 [87 4686]]

accuracy: 0.9336, precision: 0.8955, recall: 0.9818, f1: 0.9366, roc_auc: 1.0000

이후 decision-tree의 feature importance를 그래프로 출력한다. Feature importance는 decision-tree 생성 시 어떤 feature로 노드가 많이 생성되었는지 보여주는 지표이다.



Ⅲ. Malware Family Classification

1. family_classification.py

VirusTotal에서 분석한 결과들이 fetch_result 폴더 안에 각각의 해시값별로 각각의 파일들로 저장되어 있다. 각각의 악성코드가 어떤 family에 속하는지에 대한 정보는 base.json 파일의 data -> attributes -> popular_threat_classification -> popular_threat_category에 있다. VirusTotal에서 약 40여개의 백신 프로그램에 의해 얻은 검사결과를 "해당 family: 개수"의 구조로 해당 부분에 저장되어 있다. 이 정보들을 기반으로 다수결의 원칙을 가장 기본적인 기준으로 삼았으며, 다만 trojan에 대해서는 유일할 경우에만 trojan으로 분류하고, 이외의 family가 있다면 두번째 항목(개수별 오름차순 정렬 기준)을 선택하였다.

위의 기준으로 얻어낸 최종적인 family 값을 기준으로 이전의 binary classification에서 썼던 방식과 동일하게 family 별로 폴더를 만들어 옮기는 작업을 진행하였는데 이 코드에서 해당 부분을 담당하고 있다.

2. file_numbers.py

위의 기준으로 family 별로 데이터셋을 따로 분류하여 모았을 때 각각의 family별로 분포가 고르지 않았다. 우선, 각각의 family 별로 몇 개의 데이터가 있는지 분석하기 위해 각 폴더를 들어가 이미지의 개수를 세는 코드를 만들었다.

3. convert_and_resize.py

vir 파일을 이미지로 변환하고, 32x32 크기로 resize하는 코드이다. 기존 convert_to_image.py나 resize_image.py는 4만개 이미지를 생성하는 데 30분 내외로 많은 작업 시간을 소요하였다. 이를 줄이기 위해 두 과정을 하나로 통합하고 멀티프로세싱을 하도록 개선하여, 작업 시간을 3분 내외로 줄일 수 있었다.

4. malware_family_classification.ipynb

ResNet50을 기반으로 악성코드 이미지를 입력으로 하여 그것의 malware family를 분류하는 multi-class classification 모델이다. 맨 마지막 층에 소프트맥스(softmax) 층을 적용하여 해당 부분을 구현하였다. 분류 대상이 되는 malware family는 아래와 같다.

i. Adware
ii. Banker
iii. Downloader
iv. Dropper
v. Fakeav
vi. Hacktool

vii.	Miner	x.	Spyware	xiii.	Worm
viii.	Pua	xi.	Trojan		
ix.	Ransomware	xii.	Virus		

5. malware_family_classification.py

위의 부분에서 실제 train을 진행할 때 사용한 코드이다.

IV. Final Presentation

[1페이지]

이번 프로젝트의 주제는 악성코드 분석이다. 팀은 김현수, 최가온 등 2인으로 한 학기동안 프로젝트를 진행하였다.

[2페이지]

최종 발표 이전까지 해온 활동들에 대해 간략하게 설명한다. 이중에서 가장 중요한 부분은 바이러스에 감염된 파일인 vir 파일을 이미지로 변환했다는 점이다. 이러한 방식으로 변환한 데이터셋을 이용하여 총 두 가지의 문제를 해결하였다. 첫 번째 문제는 악성코드 이미지를 인풋으로 받아 해당 데이터가 malware인지 non-malware(benign)인지 이진 분류적으로 판단하는 것이다. 두 번째 문제는 첫번째 문제에서 malware로 정탐한 데이터에 대해, 해당 데이터가 어떤 악성코드 계통(malware family)에 속하는지를 다중 클래스 분류하는 문제이다.

[3페이지]

이번 프로젝트에서는 악성코드 파일을 이미지로 변환한 후, 이미지를 기반으로 악성 여부를 판단하고, 악성코드 계통을 분류하는 문제를 인공지능 모델의 설계와 학습을 통해 해결하였다.

[4페이지]

먼저 지난 발표 시기까지의 프로젝트 진행 과정에 대해 간략하게 설명하도록 하겠다.

[5페이지]

지난 발표 시간까지의 진전 과정 중 가장 핵심적인 부분은 vir 파일을 이미지로 변환하는 과정이었다. 너비가 32픽셀인 이미지를 vir파일로부터 생성하였다. 다만, 이러한 형식은 높이가 가변적이라는 점에서 CNN만으로 다루기는 어렵다는 특성이 있다. 따라서, 해당 문제를 보완하기 위해 높이도 32픽셀이 되도록 resize를 진행하였다.

[6페이지]

위의 데이터셋 정제 과정이 모두 완료된 후, 가장 먼저 시도해본 모델은 VGGNet이다. 이때 데이터셋은 malware, benign 각각 500개씩을 사용하였다. 이 상황으로 실험을 진행하였을 때, 약 70% 전후의 정확성을 보이는 것을 확인할 수 있었다.

[7페이지]

다음으로는 악성코드 파일을 이미지화하는 과정과 이렇게 새롭게 생성된 데이터셋의 특성에 대해 분석한 것들을 설명하도록 하겠다.

[8페이지]

제시된 그림과 같이 malware family에 따라 질감이 확연히 차이가 남을 시각적으로 알 수 있다. 이러한 점을 CNN 모델이 감지하여 family를 분류할 수 있을 것이라는 가정을 세웠다. 데이터셋은 총 3개의 그룹으로 형성되었다. Train set은 malware, benign 기준으로 각각 7000개, 3000개였으며 test set1과 test set2는 각각 5000개, 5000개로 균형을 맞추었다.

[9페이지]

우리 팀에서 해결한 첫번째 문제인 "악성여부 이진분류 모델 설계 및 학습"에 대해 설명하도록 하겠다.

[10페이지]

Malware Binary Classification은 32 * 32 크기의 그레이스케일 이미지를 입력으로 받아, 해당 데이터가 malware인지 non-malware인지 이진법 방식으로 분류하는 문제이다.

[11페이지]

이전에 제시한 3개의 데이터셋 중 train set으로 test set1을, test set으로 test set2를 사용하였다. 기본적으로 train set과 validation set의 데이터셋 상의 분포는 같아야 이상적이기 때문에 양쪽 모두 malware:non-malware = 1:1인 데이터셋을 사용하였다.

[12페이지]

상단에 제시된 그림은 ResNet50의 상세한 구조를 시각적으로 나타낸 그림이다. 우리의 목적은 Binary Classification이므로 fc 부분을 아래 코드와 같이 output number를 1로 설정하고, 맨 마지막에 활성화 함수로 시그모이드를 사용하였다.

[13페이지]

결론적으로 첫번째 문제는 나름 우수한 성능을 내는 것으로 확인했다. Loss 함수는 BCELoss를 사용하였고, 옵티마이저로는 SGD가 사용되었다. 400 에폭 정도를 소요하여 실험을 진행하였는데 정확도를 기준으로 train accuracy는 99.5%, test accuracy는 93.0%로 측정되었다.

[14페이지]

왼쪽의 그림은 여러 데이터셋 중 임의로 24개를 뽑아낸 것이고(대략적으로 1개의 batch라고 생각해도 무방하다.), 오른쪽의 수치는 모델이 각 이미지 데이터에 대해 출력한 값과 ground truth를 병행하여 표기한 것이다. 대부분의 데이터에 대해 제대로 된 값을 출력한다고 볼 수 있다.

[15페이지]

그러나 항상 그런 것은 아니다. 모델이 출력한 값과 ground truth를 플롯으로 그려 표현해보면 빨강 동그라미로 표시된 부분처럼 실제와 맞지 않은 값을 출력하기도 한다는 것을 알 수 있다.

[16페이지]

다음은 두 번째 문제에 해당하는 "악성코드 계통 분류 모델 설계 및 학습"을 주제로 한 소주제 프로젝트에 대해 설명하도록 하겠다.

[17페이지]

이 문제도 첫번째에서 설명한 Malware Binary Classification과 같이 32 * 32 크기의 그레이스케일 이미지를 입력으로 받는 것은 동일하다. 그러나, 이번에는 이 데이터에 대해 해당 악성파일이 감염된 악성코드의 malware family를 분류하는 데에 그 목적이 있다. 예를 들면 adware, downloader 등이 그 예시이다.

[18페이지]

이번에는 3개의 데이터셋 train set, test set1, test set2을 우선 하나로 합쳤다. 최대한 많은 데이터들을 사용하고자 함이 그 주요 목적이었는데, 해당 데이터셋은 기본적으로 malware, non-malware 기준으로는 그 개수 분포가 균등했다. 그러나, 이를 malware family 기준으로 나누었을 때 어떠한 분포가 될지 모르기에 우선 데이터를 최대한 확보하는 것을 생각했다.

또한, 이번 문제는 기본 전제가 "이 입력은 malware이다."라고 할 수 있다. 말 그대로 해당 파일이 감염된 악성코드가 어떤 계통에 속하는지를 분류하는 것이기 때문에 입력 이미지는 malware여야 할 것이다. 따라서, 데이터셋에서도 malware로 라벨링이 된 데이터들만 사용하였다. 그 결과, train set, test set1, test set2 각각 7000, 5000, 5000개를 얻을 수 있었다.

[19페이지]

이들을 합치면 총 17000개의 데이터를 얻을 수 있다.

[20페이지]

위에서 합친 17000여개의 데이터에 대해 VirusTotal를 이용하여 자동으로 정보를 크롤링하여 JSON 파일을 다운받을 수 있는 코드를 작성하였고, 해당 부분을 수행하였다. 약 40여개의 백신이 검사한 결과를 받아볼 수 있다. Base.json 파일 안에 category에 대응되는 부분이 있다. 그 부분을 기준으로 다수결의 원칙을 제1순위로 삼았다. 그러나, trojan에 대해서는 예외를 두었다. Trojan은 그것의 특성 상 정상 프로그램인 것처럼 PC 내에 상주하다가 특정 시점이 되면 공격을 개시하는 형태인데, 이때 어떤 malware family로 분화하는 것이기 때문이다. 따라서, 40여개의 백신 프로그램이 Trojan으로만 분류(판정)했다면 이때에만 trojan인 것으로 기준을 정하였다. 만약 다른 family로 분류했다면 2등에 해당하는 malware family를 기준으로 라벨링을 진행하였다.

[21페이지]

15000개의 데이터를 대상으로 VirusTotal에 검사를 진행하고 malware family를 위에서 제시한 기준으로 분류했을 때 데이터는 총 13개의 레이블이 존재했다. 다만, fakeav와 spyware에 대해서는 그 개수가 너무 작았다.

[22페이지]

현재 가지고 있는 데이터만으로는 해당 family는 정확하게 예측할 수 없다고 판단하여 해당 데이터들을 모두 지우고 13개에서 2개를 제외한 11개의 레이블을 대상으로 실험을 진행하였다.

[23페이지]

그러나, 왼쪽의 분포도를 보면 알겠지만 여전히 각 클래스 별로 그 개수 분포가 고르지 않다는 것을 알 수 있다. 이 부분을 해결하기 위해 RandomOverSampler를 이용하여 over-sampling을 수행하였다. 이를 통해 데이터의 개수를 클래스 별로 균등하게 맞출 수 있었다.

[24페이지]

이번 문제에서도 ResNet50을 그대로 사용하였다. 다만, malware family가 11개로 여러 개인 상황을 고려하여 output number를 11로 설정하였으며, 맨 마지막에 활성화 함수로 Softmax를 사용하였다.

[25페이지]

두 번째 문제는 약 500 epoch를 거쳐 test accuracy를 기준으로 약 84%의 정확도를 보였다.

[26페이지]

(생략)

[27페이지]

Virustotal scrapper의 전반적인 구조이다. Scrapper로 계속해서 json 파일을 생성하고, 실패한 hash를 따로 구분해 놓는다. 이후 주기적으로 integrity check를 실행하여 잘못 다운로드 된 hash와 실패한 hash를 다시 todo hashes에 넣었다.

[28페이지]

수집 결과는 총 4만개 폴더에 파일은 414784개로 총 용량은 2.84GB이다.

[29-32페이지]

29-32 페이지는 base.json의 흥미로운 몇 가지 feature를 소개하는 부분이다. Type description, TrID은 hash에 해당하는 파일이 어떤 종류인지 나타내는 feature이다. 만약 파일이 malware라면

popular_threat_classification이라는 key가 json에 추가되는데, malware가 주류 백신 프로그램들에 어떻게 감지되는지 감지명과 카테고리를 갖고 있다. 또 PEiD라는 툴로 파일이 어떻게 패킹되어있는지 탐지한 결과나, 각 Sections들의 시작 위치, 사이즈, 엔트로피 등도 나와있다.

[33페이지]

json 파일에서 흥미로워 보이는 feature들을 따로 빼내 csv를 생성하였다.

[34페이지]

Test set 1의 경우 정확도 약 96%, Test set 2의 경우 정확도 약 94%를 보였다.

[35페이지]

생성된 decision tree의 Feature importance이다. Feature importance는 decision tree가 생성될 때 어떤 feature가 주로 고려되었는지를 나타낸다.

단 주의해야 할 점은, decision tree 특성상 값의 개수가 많거나 범위가 넓을 경우 실제 중요도보다 feature importance가 overestimate 되고, 값의 범위가 작다면 unserestimate 되는 경향이 있다.

실제로 decision tree를 확인 해 본 결과 behaviour_count나 dropped_files_count 등이 생각보다 더 중요하게 다루어졌음을 알 수 있었다.

[36페이지]

이상으로 프로젝트 진행 상황과 결과에 대해 발표하였다. 들어주셔서 감사합니다. (끝)