딥러닝을 활용한 악성코드 분류 문제 고찰과 회색조 이미지 변환을 통한 암호화 파일 탐지

A Study on the Malware Classification Using Deep Learning and the Detection of Encrypted Files through the Transformation of Grayscale Images

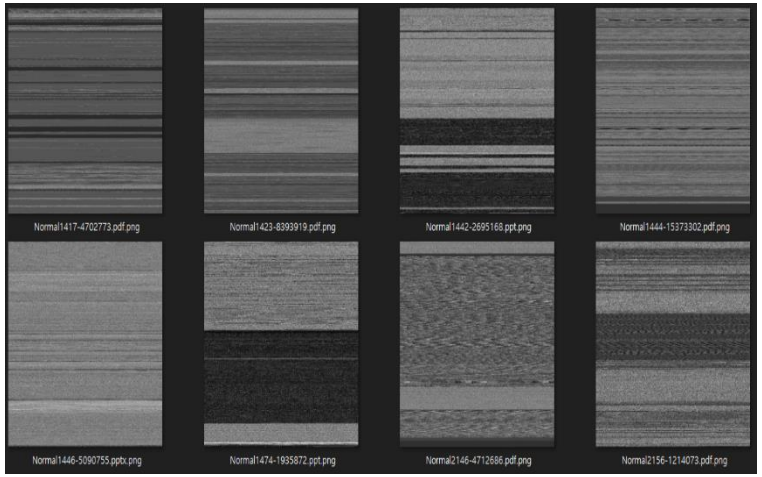
발표자 박건호

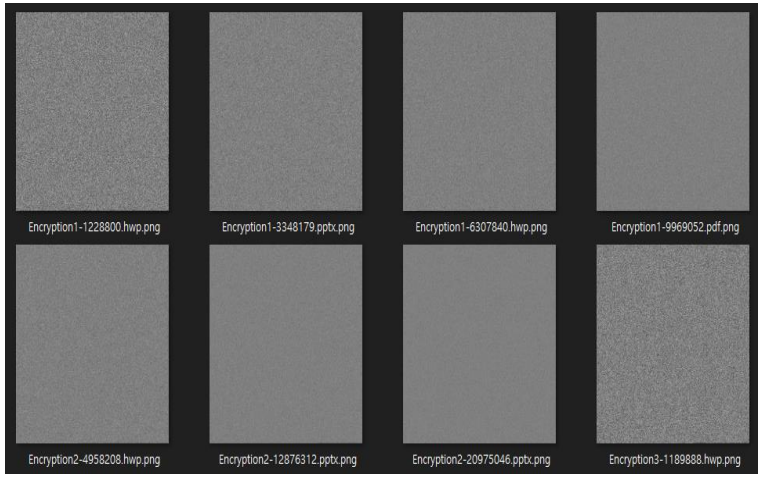
개요

1. 배경
2. 선행연구
3. 악성코드 분류를 위한 딥러닝
4. 학습에 필요한 데이터 셋 구축
5. 암호화 파일 탐지 모델 설계
6. 결과
7. 결론

배경

- 파일의 암호화 여부를 확인하기 위한 전략으로 회색조(Grayscale) 이미지 변환을 통한 악성코드 분류 문제를 적용하여 암호화 파일을 구별함
- 각 파일을 마우스 오른쪽 단추로 클릭하여 속성, 일반, 고급을 선택하여 암호화 여부 확인
- Windows 상에선 Cipher.exe 와 같은 Command Line Utility 를 사용하지 않는 한 암호화된 파일과 그렇지 않은 파일을 한 눈에 파악하기란 쉽지 않음





파일을 회색조 이미지로 변환한 결과(좌), 암호화 파일을 변환한 결과(우)

선행연구

Entropy 기반 분석

- Entropy 기반 분석은 정상과 패킹된 Malware, 암호화된 Malware에 대한 변수의 무질서 값을 측정하는 방법
- 알려진 Section에 대해 Entropy 값을 분석한 결과표

Section Analysis		Range(Count)		
Entropy Range		0 ~ <4	4 ~ <6	6 ~ 8
.bss	Normal	5	0	0
	Malware	11	0	0
.data	Normal	176	39	4
	Malware	105	20	31
.edata	Normal	3	1	0
	Malware	2	0	0
.idata	Normal	5	27	0
	Malware	2	18	0
.rdata	Normal	17	51	0
	Malware	30	81	27
.rsrc	Normal	100	83	54
	Malware	31	91	55
.text	Normal	1	51	187
	Malware	4	21	145
.tls	Normal	5	0	0
	Malware	9	0	0
.reloc	Normal	50	34	13
	Malware	41	42	17

sample type	Entropy weighted average
Normal	2.0973 ~ 3.8877
Packing Malware	4.3147 ~ 5.8062
Encryption Malware	6.0850 ~ 6.6613

- 정상 파일과 Packing된 악성코드, Encryption된 악성코드의 Entropy 값을 통해 차이를 확인할 수 있음

출처 : Malware discrimination technology based on Malware obusacation (2017.12)

선행연구

Symbol 기반 분석

- 파일의 Symbol 은 특수 문자로 되어 있는 String
- Symbol 에 특수문자 =, *, !, #, {, }, [등을 추출

Symbol	Range	Malware Frequency	Normal Frequency
!	1 ~ 30	11.62%	33.10%
	31 ~ 100	13.12%	18.47%
	101 ~ 800	51.39%	17.91%
	801 ~ 1000	23.87%	21.10%
=	1 ~ 30	6.43%	27.73%
	31 ~ 100	12.05%	22.33%
	101 ~ 800	55.08%	26.25%
	801 ~ 1000	26.45%	23.68%
[	1 ~ 30	4.94%	19.20%
	31 ~ 100	11.05%	20.61%
	101 ~ 800	54.42%	30.01%
	801 ~ 1000	29.59%	25.78%

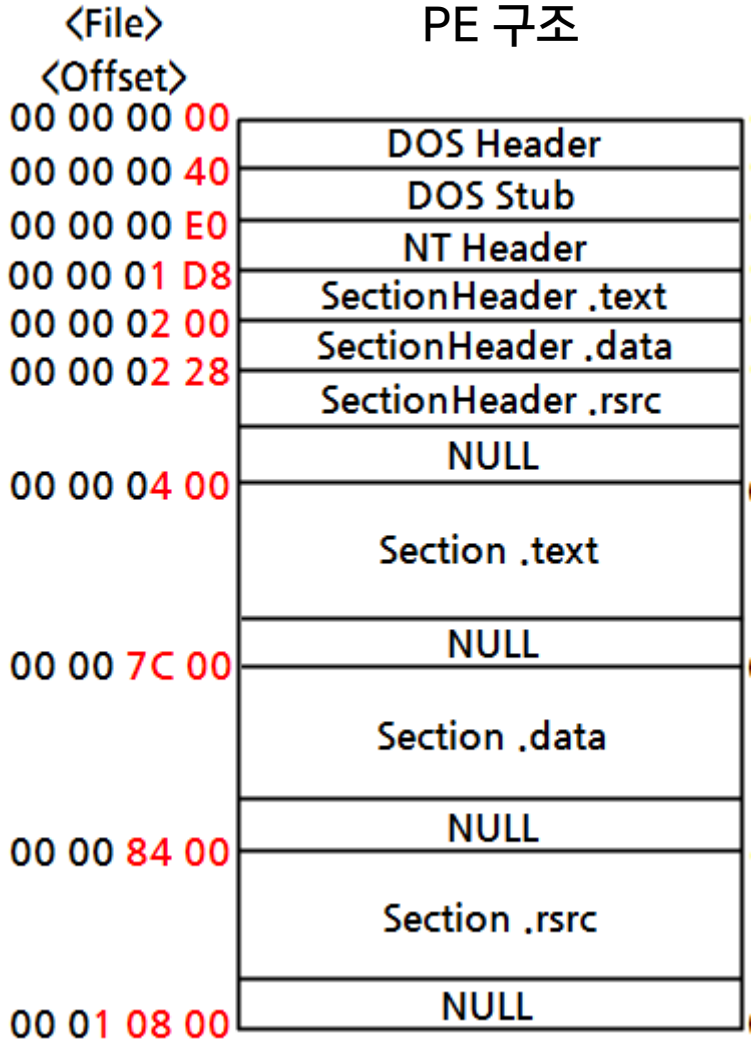
출처 : Malware discrimination technology based on Malware obusacation (2017.12)

선행연구

Symbol 기반 분석

- 파일의 Symbol 은 특수 문자로 되어 있는 String
- Symbol 에 특수문자 =, *, !, #, {, }, [등을 추출

Symbol	Range	Malware Frequency	Normal Frequency
!	1 ~ 30	11.62%	33.10%
	31 ~ 100	13.12%	18.47%
	101 ~ 800	51.39%	17.91%
	801 ~ 1000	23.87%	21.10%
=	1 ~ 30	6.43%	27.73%
	31 ~ 100	12.05%	22.33%
	101 ~ 800	55.08%	26.25%
	801 ~ 1000	26.45%	23.68%
[	1 ~ 30	4.94%	19.20%
	31 ~ 100	11.05%	20.61%
	101 ~ 800	54.42%	30.01%
	801 ~ 1000	29.59%	25.78%



출처 : Malware discrimination technology based on Malware obusacation (2017.12)

악성코드 분류를 위한 딥러닝

- 기계학습을 이용한 악성코드 분류 기법이 기존의 패턴인식 알고리즘을 이용한 접근 방식 보다 더 높은 성능을 보여줌

Microsoft (Kaggle) 악성코드 샘플을 분석, 9가지의 Family로 분류 문제

- 높은 정확도(99.15%)를 나타내는 기존의 연구에서는 악성 프로그램은 미리 처리되어 그레이 스케일 영상으로 시각화
- 이후, 다중 계층으로 구성된 아키텍처를 사용하여 해당 이미지, 프로그램의 분류 프로세스를 수행
- 주어진 데이터셋을 통해서만 연구가 진행되었기 때문에, 공격자가 탐지를 피하고자 변형한 악성 프로그램 파일을 탐지하는 것에 대해서는 한계가 있음

Family Name	# Train Samples	Type
Ramnit	1541	Worm
Lollipop	2478	Adware
Kelihos_ver3	2942	Backdoor
Vundo	475	Trojan
Simda	42	Backdoor
Tracur	751	TrojanDownloader
Kelihos_ver1	398	Backdoor
Obfuscator.ACY	1228	Any kind of obfuscated malware
Gatak	1013	Backdoor

표 1 악성코드 군집 데이터 목록

R. Ronen, M. Radu, C. Feuerstein, E. Yom-Tov and M. Ahmadi, "Microsoft Malware Classification Challenge", Feb, 2018

학습에 필요한 데이터 셋 구축

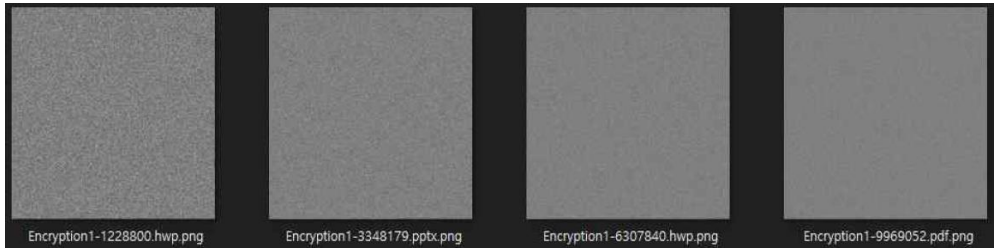
- 암호화 여부를 판단하기 위한 대상 파일을 Grayscale 이미지 변환 및 저장

```
def imgConv_Btn_function(self):
    print("[*] Image Conversion")
    fileCnt = 0
    saveFileCnt = 0
    for(path, dir, files) in os.walk(self.filePath.text()):
        for filename in files:
            ext = os.path.splitext(filename)[-1]
            fileCnt += 1
            print("[%d] %s\%s" % (fileCnt, path, filename))
            fileSize = os.path.getsize(path+'/'+filename)

            # 1MB(1024KB) ~ 200MB
            if((fileSize >= 1048576) & (fileSize <= 1048576*200)):
                saveFileCnt += 1
                binaryData = getBinaryData(path+'/'+filename)
                print("  size: ", fileSize, "/ sqrt: ",
                      (int)(math.sqrt(fileSize)))
                grayScaleImg = Image.new(
                    'L', ((int)(math.sqrt(fileSize))+1, (int)(math.sqrt(fileSize))+1))
                grayScaleImg.putdata(binaryData)
                # grayScaleImgName = filename + ".png"
                grayScaleImgName = "Normal" + \
                    str(saveFileCnt) + "-" + str(fileSize) + ext + ".png"
                grayScaleImg.save(grayScaleImgName)
    print("[*] Image Conversion END")
```

Grayscale 이미지 변환 PyQt 코드 일부,
https://github.com/devgunho/Malware_Image_Classification/tree/master/Grayscale_Image_Converter_%5BPyQt%5D

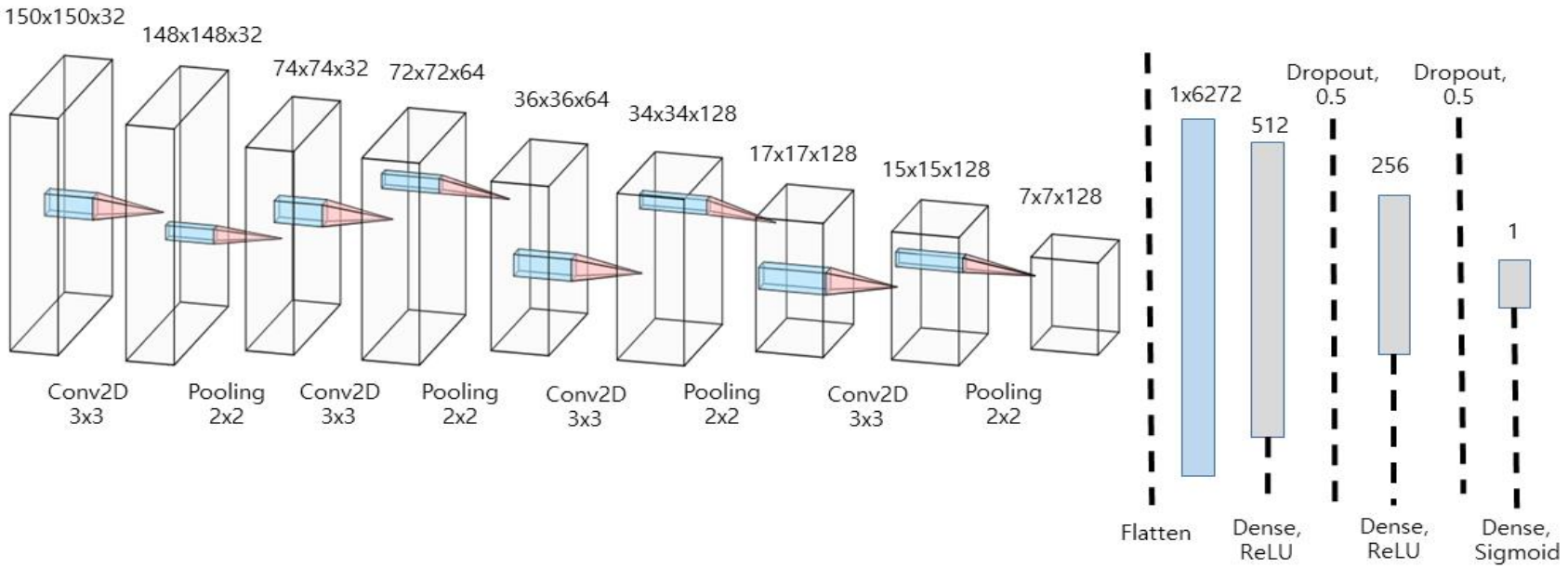
- 총 4,559쌍(9,118개)의 정상, 암호화 이미지 파일을 대상으로 설정
- 변환 대상으로 설정한 파일은 크기가 너무 작으면 특징 추출이 어렵고, 파일의 크기가 너무 크면 이미지로 변환하는데 시간이 너무 오래 걸리는 문제로 1MB부터 200MB 사이로 설정



- 암호화된 파일의 경우 이미지로 변환하였을 때 회색조 색 분포가 원본에 비해 고르게 분포되는 것을 확인

암호화 파일 탐지 모델 설계

- 암호화 파일 이진 분류를 위해 사용한 모델은 CNN(합성곱신경망 : Convolution Neural Network)
- 데이터의 특징을 추출하여 특징들의 패턴을 파악하는 구조



- 총 4개의 Convolution 층, 3개의 Pooling 층 사용

결과

- 최적의 Parameter 값을 찾기 위해 다음과 같이 매개 변수 값을 조정

	batch size	steps per epoch	epochs	validation steps	validation accuracy
1st.	20	100	30	50	86.70%
2nd.	20	80	70	50	91.50%
3th.	20	100	70	20	92.50%
4th.	**20**	**261**	**30**	**10**	**98.00%**

  - steps_per_epoch : 한 epoch에 사용한 스텝 수를 지정
  - epochs : 전체 훈련 데이터 셋에 대해 학습 반복 횟수를 지정
  - validation_steps : 한 epoch 종료 시 마다 검증할 때 사용되는 검증 스텝 수를 지정
  - validattion accuracy : validation_steps 에서 지정한 검증 스텝 수에 따른 정확도 출력

- 테스트 검증에는 ImageDataGenerater 를 사용하지 않은 실데이터를 기반으로 테스트

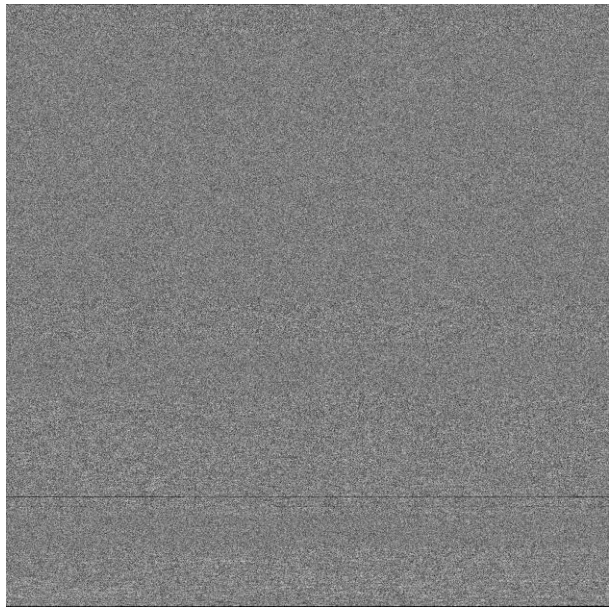
```
1 실제 : normal / 예측 : normal | Normal647-4915254.bmp.png
2 실제 : normal / 예측 : normal | Normal648-3191360.exe.png
3 실제 : normal / 예측 : normal | Normal648-4915254.bmp.png
4 실제 : normal / 예측 : normal | Normal649-3191360.exe.png
5 실제 : normal / 예측 : normal | Normal649-4915254.bmp.png
6 실제 : normal / 예측 : normal | Normal65-1126806.aif.png
7 실제 : normal / 예측 : normal | Normal65-2415170.dic.png
```

결과

- 정상 파일(Normal)로 Labeling 한 867개의 파일 중 1개의 exe 파일에 대해 encryption이라는 예측 결과를 확인

```
858 실제 : normal / 예측 : normal | Normal995-25876480.msp.png
859 실제 : normal / 예측 : normal | Normal995-2644816.dll.png
860 실제 : normal / 예측 : normal | Normal996-14422016.msi.png
861 실제 : normal / 예측 : normal | Normal996-8894792.dll.png
862 실제 : normal / 예측 : normal | Normal997-1475160.dll.png
863 실제 : normal / 예측 : normal | Normal997-1777056.exe.png
864 실제 : normal / 예측 : normal | Normal998-2525600.dll.png
865 실제 : normal / 예측 : normal | Normal998-4274016.EXE.png
866 실제 : normal / 예측 : normal | Normal999-2608944.dll.png
867 실제 : normal / 예측 : normal | Normal999-4925344.exe.png
```

```
650 실제 : normal / 예측 : encryption | Normal908-1368064.exe.png
```



- 대상 exe 파일을 분석해본 결과, Packing(실행 압축, 파일 용량이나 파일 보호를 위해 암호화)되어 있는 것을 확인

결론

- 기존에 이미지 변환을 통한 악성코드 분류 문제를 어떻게 다루었는지 확인하고, 한계점을 파악
- 간단한 딥러닝 모델로도 높은 정확도의 암호화 파일 탐지 모델 설계 가능
- 기존 연구와 달리 특정 영역이나, 일정 수치(Entropy)에 의존하여 암호화 여부를 판별하지 않고, Binary 전체를 Grayscale로 변환하는 것에 의의

향후 연구

- 딥러닝 기술을 적용한 악성코드 탐지는 여전히 많은 실험과 검증 뿐만 아니라 많은 양의 축적된 데이터를 필요로 함
- 또한, 100개의 악성코드 중 단 1개의 악성코드를 탐지하지 못한다면 대응에 실패하게 되므로, 여러 단계에 걸쳐 악성 행위를 판별할 수 있는 전략을 설계할 필요성이 있음
- 암호화 파일 이진 분류 모델을 응용한 다양한 분류 모델을 설계할 수 있을 것으로 기대

딥러닝을 활용한 악성코드 분류 문제 고찰과

회색조 이미지 변환을 통한 암호화 파일 탐지

감사합니다.