**컴퓨터 보안 Assignment2**

2016043790 정희정

목차

1. 참고 이론
2. 개발 환경 및 사용 라이브러리
3. 실행 결과
4. 구현 알고리즘 설명
5. 참고이론
   1. N-gram 기법

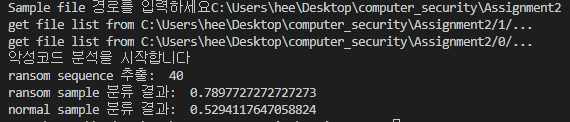
n-gram 언어 모델은 ‘일부’ 단어들이 얼마나 자주 등장하느냐에 따라 문서들을 분류하는 기법입니다. 이때 몇 개의 단어들이 연속으로 등장할지 결정하는 변수가 n 이며, 이 프로젝트에서는 4-gram 기법을 사용하였습니다.

* 1. TF-IDF 기법

TF-IDF (Term Frequency-Inverse Document Frequency) 는 단어의 빈도와 역 문서 빈도를 사용하여 코퍼스 내의 각 단어들의 중요성을 가중치로 줄 수 있는 기준을 정의합니다. 이 프로젝트에서는 TF 만 사용하였습니다.

* 1. 이론 출처
     1. 위키독스: <https://wikidocs.net/book/2155>
     2. 논문: 랜섬웨어 Native API 정보 분석을 통한 탐지 방법 연구

1. 개발 환경 및 사용 라이브러리
   1. 개발 언어: Python3.8
   2. 라이브러리: Pandas, Numpy, os
   3. 개발 및 실행 환경: Windows10
2. 실행 결과

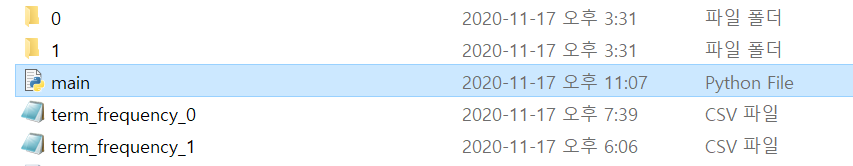


* 악성코드 샘플 파일로부터 40개의 악성코드 특징 서열을 추출합니다.
* 악성코드 특징 서열을 포함하고 있는 파일은 악성코드로 분류됩니다.
* 악성코드 샘플 파일 중 약 79%의 파일이 악성코드 파일로 분류되었습니다.
* 정상 샘플 파일 중 약 52%의 파일이 악성코드 파일로 분류되었습니다.

* 1. 실행 방법



악성코드 API 샘플 파일과 일반 샘플 API 파일이 포함 된 경로를 입력합니다. 해당 경로에는 반드시 다음과 같은 폴더와 파일이 존재하여야 정상 실행 됩니다.



* ‘0’ 폴더에는 187 개의 일반 샘플 파일, ‘1’ 폴더에는 176개의 악성코드 샘플 파일이 포함되어 있습니다.
* Term\_frequency\_0.csv: column은 정상 샘플 파일들로부터 추출한 4gram api sequence, row는 각 샘플 파일별로 4gram api sequence 의 등장 빈도 카운트가 row 로 구성되어 있습니다.
* Term\_frequency\_1.csv: 악성코드 샘플 파일로부터 추출한 csv 파일 입니다.

\*\* 4gram api sequence: 예를 들어, file1이 [A, B, C, D, E, F, G, H] 로 구성되어 있다면, 4gram api sequence 는 [ABCD, BCDE, CDEF, DEFG, EFGH] 입니다.

1. 구현 알고리즘 설명
2. **Datas 클래스**

* 전처리 과정에 필요한 데이터들을 담고 있는 객체를 생성합니다.

class Datas:

    \_path\_dir = ''

    \_isRansome = False

    \_file\_list = ''

    \_4grams = []

    \_docs = []

    \_terms\_list = set()

    def \_\_init\_\_(self,name):

        self.name = name

* \_path\_dir(string): 필요한 데이터를 추출하기 위한 샘플 파일이 담긴 경로
* \_isRansome(bool): 객체 종류가 랜섬파일인지, 일반 파일인지 구분합니다.
* \_file\_list(list): 샘플 파일 이름들을 담고있습니다.
* \_4grams(list): 4gram api sequence 를 담고 있습니다.
* \_terms\_list(set): 샘플 데이터에 존재하는 4gram sequence 의 모든 종류를 담고 있습니다.

1. **Main**
2. r\_datas = preprocess(path\_dir+'1/', True)
3. n\_datas = preprocess(path\_dir+'0/', False)
4. analysis(r\_datas, n\_datas)

* 알고리즘은 크게 데이터 전처리 과정과 샘플파일 분류 과정으로 구분됩니다.
* 악성코드 파일과 일반 파일로부터 필요한 데이터들을 추출하여 각각 r\_datas, n\_datas 에 저장합니다.
* 이 객체들로 샘플이 악성코드인지, 일반 파일인지 분석합니다.

1. **Preprocess(path\_dir, isRansome)**
2. def preprocess(path\_dir, isRansome):
3. datas = Datas('ransome')
4. if(isRansome == False):
5. datas = Datas('normal')
7. datas.\_path\_dir = path\_dir
8. datas.\_isRansome = isRansome
9. datas = get\_file\_list(path\_dir, datas)
10. # file\_list 로부터 파일들을 읽어와 datas.\_docs 에 저장한다.
11. datas = docs\_from\_filelist(datas)
12. # 변환된 파일을 4gram 화하여 datas.\_4grams 에 저장한다.
13. datas = get\_4gram\_terms(datas)
14. # <get\_malware\_featue\_terms(datas)>
15. # term\_frequency.csv file 을 만든다.
16. # term\_frequency.csv 파일 구조는 다음과 같다.
17. # columns: unique code 로 변환된 api call 을 4-gram 화 하여 나타낸 'term'의 모든 종류.
18. # rows: 파일 별로 나타낸 각 'term' 별 term frequency (count 형태)
19. # 큰 용량의 csv file 을 만드는 것이 오래걸리므로 미리 만들어서 첨부
20. # get\_malware\_feature\_terms(datas)
21. return datas

* 샘플 파일로부터 데이터를 추출하여 전처리하는 과정입니다.

1. Datas 객체를 처음 생성함
2. **get\_file\_list(path\_dir)**: 입력 받은 경로로부터 ‘.txt’ 파일만 추출하여 datas.\_file\_list 에 저장함
3. def get\_file\_list(path\_dir, datas):
4. print('get file list from '+path\_dir + '...')
5. file\_list = []
6. for i in os.listdir(path\_dir):
7. if i.endswith('.txt'):
8. file\_list.append(i)
10. datas.\_file\_list = file\_list
11. return datas
12. **docs\_from\_filelist(datas)**: file\_list 로부터 파일을 읽어서 datas.\_docs에 저장함
13. def docs\_from\_filelist(datas):
14. docs = []
15. file\_list = datas.\_file\_list
16. path\_dir = datas.\_path\_dir
17. for f in file\_list:
18. file = open(path\_dir+f, 'r')
19. doc = file.read()
20. file.close()
21. docs.append(doc)
22. datas.\_docs = docs
23. return datas

(4) **get\_4gram\_terms(datas)**: file 내용을 4gram화 하여 저장하고, 4gram sequence 의 종류를 저장한다. 이 terms\_list 는 이후 생성할 csv file 의 칼럼이 된다.

def get\_4gram\_terms(datas):

    datas = docs\_from\_filelist(datas)

    \_4grams = []   # 4gram 화 된 doc 이 들어있는 list

    terms\_list = set()   # 4gram 종류들이 담긴 set

    docs = datas.\_docs

    for doc in docs:

        list\_api = doc.splitlines()

        seq = []

        for i in range(len(list\_api)-3):

            seq.append(list\_api[i]+list\_api[i+1]+list\_api[i+2]+list\_api[i+3])

        \_4grams.append(seq)

        terms\_list = terms\_list.union(set(seq))

    datas.\_4grams = \_4grams

    datas.\_terms\_list = terms\_list

    return datas

1. 파일을 한 줄씩 읽는다.
2. 각 파일을 4줄씩 합쳐서 \_4grams 에 저장한다.
3. 어떤 종류의 4gram sequence 가 존재하는지 알기 위해 중복된 sequence 를 제거하여 terms\_list 에 저장한다.

(5)**get\_malware\_feature\_terms(datas)**: 추출한 데이터로부터 term\_frequency.csv 파일을 만든다.

def get\_malware\_feature\_terms(datas):

    \_4grams = datas.\_4grams

    terms\_list = datas.\_terms\_list

    file\_list = datas.\_file\_list

    df = pd.DataFrame(columns=terms\_list)

    print('df 만듬')

    df.insert(0, 'file\_name', file\_list)

    i = 0

    # 파일이 커서 이부분이 오래걸림

    for term in terms\_list:

        i+=1

        tf\_list = []

        #docs 들안에 term이 몇개 들어가있는지 반환.

        for d in \_4grams:

            tf\_list.append(d.count(term))

        df[term] = tf\_list

        if(i%100 == 0):

            print(i)

    df.to\_csv('term\_frequency.csv', mode='w')

* Term\_frequency.csv 파일 구조는 다음과 같다.
* Columns: 샘플파일 내 존재하는 4gram화된 api sequence(term 이라고 명명함) 의 모든 종류
* Rows: 파일 별로 나타낸 각 ‘term’ 별 term frequency 를 count 형태로 나타냄
* 악성코드 샘플파일 176개, 일반 샘플 파일 187개에 대해서 dataframe에 tf 값을 추가하는 작업은 시간 복잡도가 매우 큼.
* 따라서 미리 함수를 실행하여 과제 파일에 첨부함. 주석을 해제하고 실행하면 같은 csv file 이 저장됨.

1. **def analysis(r\_datas, n\_datas):**

* 이렇게 형성 된 r\_datas 객체(from ransom sample file) 와 n\_datas 객체(from normal sample file) 에 담긴 정보를 이용하여 분석을 진행한다.

1. **Csv file 로부터 dataframe 생성하기**

df\_r = pd.read\_csv(r\_datas.\_path\_dir[:-2]+'term\_frequency\_1.csv')

df\_n = pd.read\_csv(n\_datas.\_path\_dir[:-2]+'term\_frequency\_0.csv')

r\_4grams = r\_datas.\_4grams  #4gram으로 묶인 파일 내용

     n\_4grams = n\_datas.\_4grams

     ran\_seq = []    #ransome 특징 서열

     nor\_seq = []

(2) **ransom sample file term frequency 로부터 악성코드 특징서열 추출하기**.

    for i in range(len(r\_4grams)):

        doc = r\_4grams[i]

        for w in doc:

            TFs = df\_r[w]

            if(TFs[i]/len(doc) > 0.06):

                ran\_seq.append(w)

                break

    ran\_seq = set(ran\_seq)

* ‘i’는 각 4gram화된 파일들의 index 순서 값이며, 동시에 data frame 의 row file의 index 이기도 하다.
* ‘doc’에 포함된 각 ‘4gram term’ 에 대하여, 만약 해당 term 의 tf 값이 전체 term 등장 횟수에 대해서 **6% 초과** 하는 경우, 해당 term 은 악성코드 특징 서열이라고 간주한다.

**(3) normal sample file term frequency로부터 일반 파일 특징 서열 추출하여 악성코드 특징 서열 리스트에서 제거하기**

    for j in range(len(n\_4grams)-1):

        doc = n\_4grams[j]

        for w in doc:

            TFs = df\_n[w]

            if(TFs[j]/len(doc) > 0.05):

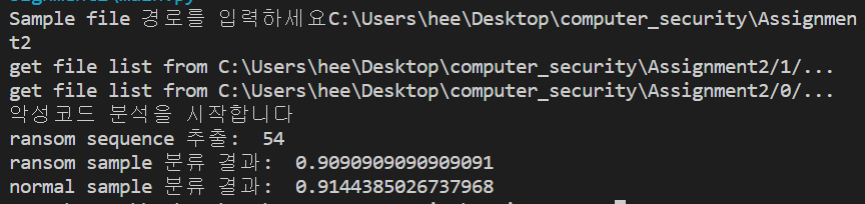
                nor\_seq.append(w)

                break

    nor\_seq = set(nor\_seq)

    ran\_seq = set.difference(ran\_seq, nor\_seq)

* 해당 작업은 랜섬웨어 특징 서열 파일에 포함 된 일반 파일 특징 서열을 제거하여 일반 파일 샘플을 랜섬 웨어로 잘못 분류하는 확률을 줄이기 위해서이다.
* 해당 작업을 하지 않았을 때는 랜섬웨어 샘플 파일 중 약 91%를 랜섬웨어로 분류하고, 일반 샘플 파일 중 약 91%를 램섬웨어 파일로 분류하여 실질적으로 일반 파일과 랜섬 웨어를 유의미하게 구분하지 못하였다. (아래 결과 화면 참고)



1. (2)와 (3) 으로부터 추출한 악성코드 특징 서열 리스트를 이용하여 샘플 분류하기

    count = 0

    for i in range(len(r\_4grams)):

        r\_doc = r\_4grams[i]

        for rs in ran\_seq:

            if (rs in r\_doc and df\_r[rs][i] > 1):

                count+=1

                break

    n\_count = 0

    for j in range(len(n\_4grams)-1):

        n\_doc = n\_4grams[j]

        for rs in ran\_seq:

            if (rs in n\_doc and df\_n[rs][j] > 1):

                n\_count+=1

                break

    print('ransom sample 분류 결과: ', count/len(r\_4grams))

    print('normal sample 분류 결과: ', n\_count/len(n\_4grams))

* 4gram 화 된 각 샘플 파일 내용에 악성 코드 특징 서열이 포함되어 있고, (and) 그 빈도수가 한번 이상인 것에 대해서 랜섬웨어 파일로 분류하도록 하였다.
* 해당 작업을 일반 파일 샘플에 대해서도 똑같이 진행한다.
* 분류결과는 전체 파일 개수 중에서 몇 개가 랜섬웨어로 분류되었는지 확률로 계산하여 표시한다.