<정보보호와시스템보안 프로젝트2 보고서>

팀원 : 장진우/20171694/소프트웨어학부
박영신/20202082/정보보안암호수학과
정혜승/20202105/정보보안암호수학과
작성자 : 장진우

<파일 주의사항>

- 이번 프로젝트는 로컬에서 실행하는 것을 전제로 하였습니다.
  - 다른 곳에서 실행시키시려면 datapath의 경로를 변경해주세요. 해당 부분의 경로가 각종 데이터들이 담긴 최상위 폴더입니다.

<작업 내용>

- EmberParser와 PestudioParser에 추가적인 벡터값을 넣었습니다.
- 벡터값을 만들고 처리하는 process함수를 개량했습니다.

<파일 설명>

- EmberParser와 PestudioParser를 중점적으로 변경했습니다.
  - 먼저 EmberParser에 넣은 추가적인 부분만 보여드리겠습니다.

```
def get_section_info(self):
    # 나머지는 비슷하거나 영향이 미미하므로 "characteristics"만 가져옴
    section = self.report["section"]["sections"]
    name_size = 20
    props_size = 45
    name_vec = [0 for i in range(name_size)]
    props_vec = [0 for i in range(props_size)]

    size_sum, ent_sum, vs_sum = 0, 0, 0
    if len(section) == 0:
        print('EMP')
    for i in section:
        if not i["name"] == "":
            self.hasher.update(i["name"].encode("UTF-8"))  # String
            name_vec[int(self.hasher.hexdigest(), 16) % name_size] += 1
            for h in i["props"]:
                self.hasher.update(h.encode("UTF-8"))  # String
                props_vec[int(self.hasher.hexdigest(), 16) % props_size] += 1
```

- section에 있는 부분 중 sections를 추가로 벡터화시켰습니다. 해당 부분에는 보통 5가지의 값들이 존재합니다(존재하지 않는 경우와, 아예 빈 값으로 두는 경우도 있었습니다). 그 중 name과 props[]는 텍스트이고, 이들은 피쳐 해싱을 통해 벡터화시켰습니다. name과 props[]에 각각 20, 45칸의 고정적인 배열이 쓰였습니다.

```
        size_sum += i["size"]
        ent_sum += i["entropy"]
        vs_sum += i["vsize"]
    size_sum = size_sum / len(section) if size_sum > 0 else 0
    ent_sum = ent_sum / len(section) if ent_sum > 0 else 0
    vs_sum = vs_sum / len(section) if vs_sum > 0 else 0
    vector = [size_sum, ent_sum, vs_sum]
    vector.extend(name_vec)
    vector.extend(props_vec)
```

- 위쪽의 세 값들은 For문에 포함된 값들입니다. 저 값들은 for문을 한 번 돌 때마다 새로운 값을 더하고, 그 값을 section 변수의 길이로 나눠 일종의 평균값처럼 만들었습니다. 나눌 수 없는 경우(이 값도 0이 되는 경우가 있습니다) 그냥 0을 대입했습니다.
- 이렇게 만든 값들은 전부 vector 변수에 모아서 리턴합니다.

```
def get_import_info(self):
    imports = self.report["imports"]
    imp_size = 20
    imp_vec = [0 for i in range(imp_size)]
    for i in imports:
        for j in i:
            self.hasher.update(j.encode("UTF-8"))  # String
            imp_vec[int(self.hasher.hexdigest(), 16) % imp_size] += 1

    return imp_vec
```

- Import에 있는 모든 파일명 텍스트(라이브러리는 PESTUDIO에서 구할 거라 제외했습니다)들을 모으는 부분입니다. 마찬가지로 파일 이름에 피쳐 해싱을 사용했습니다. 고정적으로 20칸을 사용합니다.
- 해당 함수 밑에 get_export_info가 있지만, 사용하지는 않았습니다. 없는 함수로 생각하셔도 될 것 같습니다.

```
def get_overview_info(self):
    overview = self.report["image"]["overview"]
    self.hasher.update(overview["size"].encode("UTF-8"))
    vector = [int(self.hasher.hexdigest(), 16) % 10, float(overview["entropy"])]
    return vector
```

- PESTUDIO의 parser 부분입니다. overview의 값들을 벡터화시키는 부분입니다. 저희는 size와 entropy를 사용했습니다.

```
def get_library_info(self):
    lib_vec = [0 for j in range(20)]
    try:
        library = self.report["image"]["libraries"]["library"]
    except KeyError:
        return lib_vec
    if type(library) == dict:
        self.hasher.update(library["@name"].encode("UTF-8"))  # String
        lib_vec[int(self.hasher.hexdigest(), 16) % 20] += 1
    else:
        for i in library:
            self.hasher.update(str(i["@name"]).encode("UTF-8"))  # String
            lib_vec[int(self.hasher.hexdigest(), 16) % 20] += 1
```

- library의 값들을 가져옵니다. 이 값은 EMBER의 get_import_info에서 imports dictionary의 킷값입니다. 가져오기 애매한 관계로 이곳에서 같이 가져왔습니다. 해당 값은 @name을 통해 가져왔고, 피쳐 해싱을 통해 가져왔습니다. 이 피쳐해싱에는 고정적으로 20칸의 배열이 쓰였습니다.

- 만들어 둔 parser 함수들을 통해 데이터를 모으는 process2 함수입니다.
- 앞부분은 튜토리얼 코드와 거의 같습니다. 달라진 부분만 보여드리겠습니다.

```
            except FileNotFoundError:
                if data == peminer:
                    feature_vector += [0 for i in range(188)]
                elif data == ember:
                    feature_vector += [0 for i in range(438)]
                else:
                    feature_vector += [0 for i in range(22)]
            except TypeError:
                print(path)
            except ValueError:
                print('Error occured : ' + fname)

        V.append(feature_vector)
        w.append(label)
        x += 1
        print("path_1 count : ", x)
    return V, w
```

- 해당 부분은 에러를 처리하는 부분입니다. 중요한 부분은 FileNotFoundError부분인데, 파일이 정상적으로 존재하지 않는 경우 data가 어느 부분을 가리켰는지를 파악해서 그 부분의 parser가 제공해야 했던 만큼의 값을 제공해줍니다. 이렇게 빈 값을 넣어주는 이유는, 이 값을 넣어야 직사각형의 행렬값을 얻을 수 있고, 직사각형의 행렬값이어야 나중에 볼 csr_matrix를 만들 수 있기 때문입니다. 직사각형, n x m형 배열이 아니라면 해당 매트릭스를 만들 수 없습니다.

```
def test_process(peminer, ember, pestudio):
    X = []
    x = 0
    a = os.listdir(f"{peminer}")
    b = os.listdir(f"{ember}")
    c = os.listdir(f"{pestudio}")
    a.extend(b)
    a.extend(c)
    all_files = set(a)
    for fname in list(all_files):
        feature_vector = []
        lname = fname.split('.')[0]
        for data in [peminer, ember, pestudio]:
```

- 해당 함수는 라벨을 붙이면 안 되는 테스트데이터를 위해 만들어진 process 함수입니다. process2와 기능상으로는 거의 같지만, 라벨 관련된 기능들은 거의 다 사라져있습니다.

```
aeee = np.asarray(X)
csr = csr_matrix(aeee)
```

- 긴 시간을 거쳐서 process로 배열을 만들었다면, 그걸 기계에게 학습시킬 수 있게 csr_matrix로 바꾸는 작업을 거쳐야 하는데, 이 부분이 그렇게 만들어주는 부분입니다.

```
models = []
for model in ["xg"]:
    clf = train(csr, y, model)
    models.append(clf)
```

- 실제로 학습을 수행하는 부분입니다. clf 변수에 해당 학습모델 객체가 저장되고, 이는 다시 models 배열에 저장됩니다. 저희 팀은 XGBoost를 사용해보았습니다.

```
save_ensemble_result(testV, models, os.listdir(peminer))
```

```
def save_ensemble_result(X, models, path=[]):

    # Soft Voting
    # https://devkor.tistory.com/entry/Soft-Voting-%EA%B3%BC-Ha
    predicts = []
    count = 0
    for model in models:
        prob = [result for _, result in model.predict_proba(X)]
        predicts.append(prob)

    predict = np.mean(predicts, axis=0)
    predict = [1 if x >= 0.5 else 0 for x in predict]
    save_csv(path, predict)
```

- 학습시킨 모델을 통해 해당 파일의 결과물을 CSV로 남기는 함수입니다. 여기서 test_process로 만든 testV를 사용합니다. predict를 통해 결과값을 만든 다음, 파일 이름과 결과값이 담긴 배열들을 전부 save_csv로 보냅니다.

```
def save_csv(filename, predict):
    f = open('./predict.csv', 'w', encoding='utf-8', newline='')
    wr = csv.writer(f)
    wr.writerow(['file', 'predict'])

    for idx in range(len(filename)):
        wr.writerow([filename[idx][:-5], predict[idx]])

    f.close()
```

- 실제로 CSV 파일을 만드는 함수입니다. 여기서 결과값들을 모아서 CSV파일로 만듭니다.

<결과>

```
정확도 0.9002
```

- evaluate의 값입니다. 대략 90% 정도가 나왔습니다.
- ensemble_result는 30분 이상을 실행시켜도 값이 제대로 나오지 않는 관계로 중단시켰습니다.