정보보호와 시스템보안

- 프로젝트2 보고서 -

소프트웨어학부 김영훈(20171597)
소프트웨어학부 한준호(20153239)
정보보안암호수학과 김해찬(20183441)

1) 특징 추출

먼저 PEMINER, EMBER, PESTUDIO 각각의 JSON 파일에 존재하는 파일들의 특징들 중에서 악성코드 분류 학습에 사용할 특징을 추출한다. 여기서 추출한 특징들을 가공하여 특징 벡터를 만들어 전처리를 진행하고 이어서 모델을 학습한다.

1-1) PEMINER

000b2d190e99af161538280dec3dc714...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
000b958b05b0dfe2af6d9a560b6a5c33f...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
000c4ae5e00a1d4de991a9decf9ecbac5...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
000f2108db11449003f5c159b7fd4698...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00a293e61a8ed068aae3f4bbd58a7d3c...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00abc03ce485ecb2947711570d1e81c4...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00b51e08bab0f41f0dce51c76b276cbe...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00b424fce60e8c3abb5bc05e2e1c4d86f...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00b10454e058f4e2a9585574f4464ca3...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00bb9236c7170da3d3f044553cf4f48ec...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00bc60b911ddd8b811db2963994bb98...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00be3f5e3620ac485bd653489b431c5d...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00bf28d42e14245b5a0abfadbef08af28...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00c359cfdffbac50151d0cfeceb325434a...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00c890a8b4f5331a4719ae2a9cefad27c...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00c8940b0974330add118781c1df17ec...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00cb9b017083403f3628369480317c9...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00cb91e9f809ed24d6deaa58c4eabface...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00cee9974c96a860550b381d1c16851...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00d027b36af19b25ae1062202617e1c8...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00d83b667526aec1941ea170c0930cb3...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00dbe3e89bdc92146ee0e4a0f5f8d3b7c...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00dd09b0fba666f9105dd9ae78852277...	2020-10-29 오후 10:29	JSON 원본 파일	8KB
00df3d16d6574c9909feefd8efa0be618...	2020-10-29 오후 10:29	JSON 원본 파일	8KB

PEMINER의 경우 데이터 크기가 작아 전체 데이터를 모두 학습에 사용한다.

1-2) EMBER

사용한 특징 값)

● historgram 값과 byte entropy 값

```
def get_histogram_info(self):
    histogram = np.array(self.report["histogram"])
    total = histogram.sum()
    vector = histogram / total
    return vector.tolist()

def get_byte_entropy_info(self):
    histogram = np.array(self.report["byteentropy"])
    total = histogram.sum()
    vector = histogram / total
    return vector.tolist()
```

● string 값

```
def get_string_info(self):
    strings = self.report["strings"]

    hist_divisor = float(strings['printables']) if strings['printables'] > 0 else 1.0
    vector = [
        strings['numstrings'],
        strings['avlength'],
        strings['printables'],
        strings['entropy'],
        strings['paths'],
        strings['urls'],
        strings['registry'],
        strings['MZ']
    ]

    vector += (np.asarray(strings['printabledist']) / hist_divisor).tolist()
    return vector
```

● general file 값

```
def get_general_file_info(self):
    general = self.report["general"]
    vector = [
        general['size'], general['vsize'], general['has_debug'], general['exports'], general['imports'],
        general['has_relocations'], general['has_resources'], general['has_signature'], general['has_tls'],
        general['symbols']
    ]
    return vector
```

● data directory 값

```
def get_data_directory_info(self):
    data_directory = self.report['datadirectories']
    if len(data_directory) != 15:
        vector = [0] * 15
    else:
        vector = []
        for data in data_directory:
            vector.append(data['size'])
    return vector
```

1-3) PESTUDIO

사용한 특징 값)

● overview의 entropy 값

● file-header 값

```
vector = [
    self.tf_to_int(file_header['relocation-stripped']),
    self.tf_to_int(file_header['executable']),
    self.tf_to_int(file_header['large-address-aware']),
    self.tf_to_int(file_header['processor-32bit']),
    self.tf_to_int(file_header['uniprocessor']),
    self.tf_to_int(file_header['system-image']),
    self.tf_to_int(file_header['dynamic-link-library']),
    self.tf_to_int(file_header['debug-stripped']),
    self.tf_to_int(file_header['media-run-from-swap']),
    self.tf_to_int(file_header['network-run-from-swap']),
    int(file_header['sections']),
    int(file_header['number-of-symbols']),
    int(file_header['size-of-optional-header'])
]
```

● optional-header 값

```
vector = [
    int(optional_header['size-of-code']),
    int(optional_header['size-of-initialized-data']),
    int(optional_header['size-of-uninitialized-data']),
    self.hex_to_int(optional_header['entry-point']),
    self.hex_to_int(optional_header['base-of-code']),
    self.hex_to_int(optional_header['base-of-data']),
    self.hex_to_int(optional_header['image-base']),
    self.hex_to_int(optional_header['section-alignment']),
    self.hex_to_int(optional_header['file-alignment']),
    int(optional_header['size-of-image']),
    int(optional_header['size-of-headers']),
    self.hex_to_int(optional_header['file-checksum']),
    self.tf_to_int(optional_header['code-integrity']),
    self.tf_to_int(optional_header['structured-exception-handling']),
    self.tf_to_int(optional_header['data-execution-prevention'])
]
```

● section 값

```
vector = [len(sections)]
vector.append(max([int(data['@virtual-size']) for data in sections]))
vector.append(max([int(data['@raw-size']) for data in sections]))
vector.append(len([self.check_x(data['@self-modifying']) for data in sections]))
vector.append(len([self.check_x(data['@blacklisted']) for data in sections]))
vector.append(len([self.check_x(data['@initialized-data']) for data in sections]))
vector.append(len([self.check_x(data['@uninitialized-data']) for data in sections]))
vector.append(len([self.check_x(data['@discardable']) for data in sections]))
vector.append(len([self.check_x(data['@shareable']) for data in sections]))
vector.append(len([self.check_x(data['@executable']) for data in sections]))
vector.append(len([self.check_x(data['@readable']) for data in sections]))
vector.append(len([self.check_x(data['@writable']) for data in sections]))

try:
    entropy = max([float(data['@entropy']) for data in sections])
except:
    entropy = 0
vector.append(entropy)
```

● resources 값

```
if self.report['resources'].get('instance', None) is None:
    return [0, 0, 0.0, 0.0]
else:
    resources = self.report['resources']['instance']
    vector = []
    if type(resources) == list:
        vector.append(len(resources))
        vector.append(max([int(data['@size']) for data in resources if data != None], default=0))
        vector.append(max([float(data['@entropy']) for data in resources if data != None and data['@entropy'] != '-'], default=0.0))
        vector.append(max([float(data['@file-ratio'][:-1]) for data in resources if data != None and data['@file-ratio'] != 'n/a'], default=0.0))
    else:
        vector.append(1)
        vector.append(int(resources['@size']))
        vector.append(float(resources['@entropy'] if resources['@entropy'] != '-' else 0.0))
        vector.append(float(resources['@file-ratio'][:-1] if resources['@file-ratio'] != 'n/a' else 0.0))

    return vector
```

● string 값

```
strings = self.report['strings']
vector = [
    int(strings['@count']),
    int(strings['@bl']),
    int(strings['ascii']['@count']),
]

# 'unicode'에 'null'이 나올 수 있음.
try:
    uni_count = int(strings['unicode']['@count'])
except:
    vector.append(0)
else:
    vector.append(uni_count)
return vector
```

● certificate 값

```
if self.report.get('certificate', None) is None:
    return [0]

if self.report['certificate'] == 'n/a':
    return [0]
else:
    return [1]
```

● overlay 값

```
if self.report.get('overlay', None) is None:
    return [0, 0, 0, 0, 0]

overlay = self.report['overlay']
if overlay == 'n/a':
    return [0, 0, 0, 0, 0]
else:
    vector = [1]
    vector.append(self.hex_to_int(overlay['file-offset']))
    vector.append(int(overlay['size']))
    vector.append(float(overlay['entropy']))
    vector.append(float(overlay['file-ratio'][:-1]))

    return vector
```

● 기타 manifest, debug, version, tls callback 값

```
manifest = self.report.get('manifest', None)
debug = self.report.get('debug', None)
version = self.report.get('version', None)
tls_callback = self.report.get('tls-callbacks', None)

mani_vector = [1 if manifest != 'n/a' and manifest != None else 0]
debug_vector = [1 if debug != 'n/a' and debug != None else 0]
version_vector = [1 if version != 'n/a' and version != None else 0]
tls_vector = [1 if tls_callback != 'n/a' and tls_callback != None else 0]

vector = []
vector += mani_vector
vector += debug_vector
vector += version_vector
vector += tls_vector

return vector
```

이 값은 n/a 인지 여부만 파악해서 0 또는 1로 특징 벡터를 만들어주었다.

```
def process_report(self):
    vector = []
    vector += self.get_overview_info()
    vector += self.get_file_header_info()
    vector += self.get_optional_header_info()
    vector += self.get_sections_info()
    vector += self.get_resources_info()
    vector += self.get_string_info()
    vector += self.get_certificate_info()
    vector += self.get_overlay_info()
    vector += self.etc_info()

    return vector
```

이러한 특징 값을 특징 벡터로 만들어서 vector 값을 리턴해주도록 각 함수를 구성한 다음 최종적으로 process_report 함수에서 각 특징벡터들의 값을 모두 합쳐주는 작업을 한다.
이렇게 합쳐진 특징 벡터 값은 정답 label 데이터와 함께 모델 학습에 필요한 train 데이터로 사용된다.

2) 모델 학습

```python
def peminer_dataset(manual_test=True):
    peminer = "./data/PEMINER/학습데이터/*"
    peminer_valid = "./data/PEMINER/검증데이터/*"
    peminer_test = "./data/PEMINER/테스트데이터/*"

    peminer_file_list = glob.glob(peminer)
    peminer_valid_file_list = glob.glob(peminer_valid)
    peminer_fname_list = [file.split("\\")[1].split(".")[0] for file in peminer_file_list]
    peminer_valid_fname_list = [file.split("\\")[1].split(".")[0] for file in peminer_valid_file_list]

    peminer_X_train, peminer_y_train = [], []
    peminer_X_valid, peminer_y_valid = [], []

    for fname in tqdm(peminer_fname_list):
        feature_vector = []
        label = label_table[fname]
        path = f"./data/PEMINER/학습데이터/{fname}.json"
        feature_vector += PeminerParser(path).process_report()
        peminer_X_train.append(feature_vector)
        peminer_y_train.append(label)

    for fname in tqdm(peminer_valid_fname_list):
        feature_vector = []
        label = test_table[fname]
        path = f"./data/PEMINER/검증데이터/{fname}.json"
        feature_vector += PeminerParser(path).process_report()
        peminer_X_valid.append(feature_vector)
        peminer_y_valid.append(label)

    if manual_test:
        models = []
        for model in ["rf"]:
            clf = train(peminer_X_train, peminer_y_train, model)
            models.append(clf)

        for model in models:
            evaluate(peminer_X_valid, peminer_y_valid, model)

    return [peminer_X_train, peminer_y_train, peminer_X_valid, peminer_y_valid]
```

```python
def ember_dataset(manual_test=True):
    ember = "./data/EMBER/학습데이터/*"
    ember_valid = "./data/EMBER/검증데이터/*"
    ember_test = "./data/EMBER/테스트데이터/*"

    ember_file_list = glob.glob(ember)
    ember_valid_file_list = glob.glob(ember_valid)
    ember_fname_list = [file.split("\\")[1].split(".")[0] for file in ember_file_list]
    ember_valid_fname_list = [file.split("\\")[1].split(".")[0] for file in ember_valid_file_list]

    ember_X_train, ember_y_train = [], []
    ember_X_valid, ember_y_valid = [], []

    for fname in tqdm(ember_fname_list):
        feature_vector = []
        label = label_table[fname]
        path = f"./data/EMBER/학습데이터/{fname}.json"
        feature_vector += EmberParser(path).process_report()
        ember_X_train.append(feature_vector)
        ember_y_train.append(label)

    for fname in tqdm(ember_valid_fname_list):
        feature_vector = []
        label = test_table[fname]
        path = f"./data/EMBER/검증데이터/{fname}.json"
        feature_vector += EmberParser(path).process_report()
        ember_X_valid.append(feature_vector)
        ember_y_valid.append(label)

    if manual_test:
        models = []
        for model in ["rf"]:
            clf = train(ember_X_train, ember_y_train, model)
            models.append(clf)

        for model in models:
            evaluate(ember_X_valid, ember_y_valid, model)

    return [ember_X_train, ember_y_train, ember_X_valid, ember_y_valid]
```

```python
def pestudio_dataset(manual_test=True):
    pestudio = "./data/PESTUDIO/학습데이터/*"
    pestudio_valid = "./data/PESTUDIO/검증데이터/*"
    pestudio_test = "./data/PESTUDIO/테스트데이터/*"

    pestudio_file_list = glob.glob(pestudio)
    pestudio_valid_file_list = glob.glob(pestudio_valid)
    pestudio_fname_list = [file.split("\\")[1].split(".")[0] for file in pestudio_file_list]
    pestudio_valid_fname_list = [file.split("\\")[1].split(".")[0] for file in pestudio_valid_file_list]

    pestudio_X_train, pestudio_y_train = [], []
    pestudio_X_valid, pestudio_y_valid = [], []

    for fname in tqdm(pestudio_fname_list):
        feature_vector = []
        label = label_table[fname]
        path = f"./data/PESTUDIO/학습데이터/{fname}.json"
        feature_vector += PestudioParser(path).process_report()
        pestudio_X_train.append(feature_vector)
        pestudio_y_train.append(label)

    for fname in tqdm(pestudio_valid_fname_list):
        feature_vector = []
        label = test_table[fname]
        path = f"./data/PESTUDIO/검증데이터/{fname}.json"
        feature_vector += PestudioParser(path).process_report()
        pestudio_X_valid.append(feature_vector)
        pestudio_y_valid.append(label)

    if manual_test:
        models = []
        for model in ["rf"]:
            clf = train(pestudio_X_train, pestudio_y_train, model)
            models.append(clf)

        for model in models:
            evaluate(pestudio_X_valid, pestudio_y_valid, model)

    return [pestudio_X_train, pestudio_y_train, pestudio_X_valid, pestudio_y_valid]
```

1번에서 생성한 특징 벡터를 각 PEMINER, EMBER, PESTUDIO 별로 feature vector를 만들어서 모델 학습에 필요한 train으로 사용한다.
feature vector와 함께 label 값도 함께 불러와서 모델 학습에 필요한 정답 값으로 주고 학습을 진행한다.
마지막으로 load_model 함수에 존재하는 스트링 분류기를 통해 RandomForest 모델로 학습을 진행하였고 그에 대한 결과를 검증 데이터를 불러와서 evaluate 함수를 통해 정확도를 계산해본다.

3) 결과 확인
PEMINER)

```
100%|██████████| 20000/20000 [00:05<00:00, 3923.36it/s]
100%|██████████| 10000/10000 [00:02<00:00, 3965.11it/s]
정확도 0.9518
```

EMBER)

```
100%|██████████| 20000/20000 [00:08<00:00, 2274.20it/s]
100%|██████████| 10000/10000 [00:04<00:00, 2278.77it/s]
정확도 0.9309
```

PESTUDIO)

```
100%|██████████| 19440/19440 [05:06<00:00, 63.42it/s]
100%|██████████| 9798/9798 [02:23<00:00, 68.40it/s]
정확도 0.9548887528066953
```