104 – PDF Classification

Team Information

Team Name : LuckyVicky

Team Member : Eungchang Lee, Hyun Yi, Juho Heo, Dongkyu Lee

Email Address : dfc_luckyvicky@googlegroups.com

Instructions

Description Develop a tool to analyze the internal structure of given example PDF files and classify them into normal PDFs, corrupted PDFs, and encrypted PDFs.

Target	Hash (MD5)
1.pdf	c578c2eaeb340372f7ca96ae20c5d6e5
2.pdf	89a5c129333d6aba39befe9b6ce40cf7
3.pdf	d07f21d1a4ef17de5ec7a531f50b9f84

Teams must:

- Develop and document the step-by-step approach used to solve this problem to allow another examiner to replicate team actions and results.
- Specify all tools used in deriving the conclusion(s).

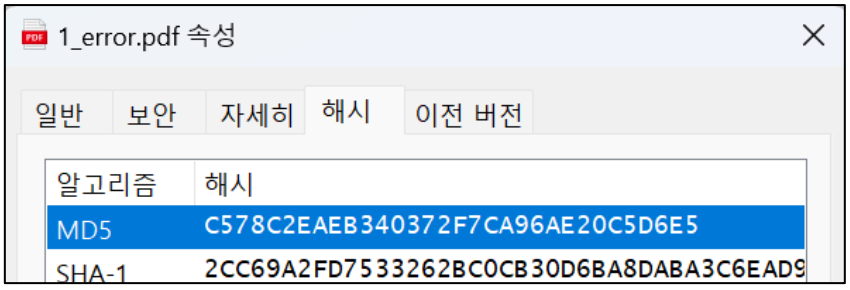
Tools used:

Name:	Python	Publisher:	Python Software Foundation
Version:	3.11.8		
URL:	https://www.python.org/		

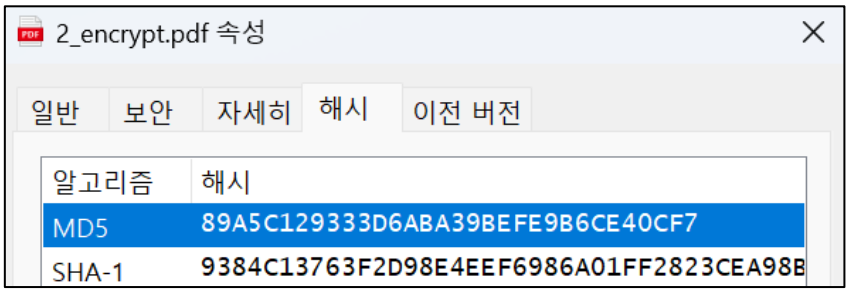
Name:	pypdf	Publisher:	py-pdf
Version:	4.3.1		
URL:	https://github.com/py-pdf/pypdf		

Name:	HxD	Publisher:	Maël Hörz
Version:	1.7.7.0		
URL:	https://mh-nexus.de/en/hxd/		

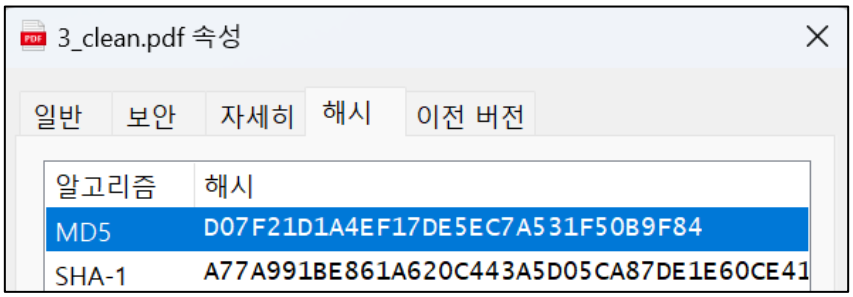
Step-by-step methodology:



[그림 1] 1_error.pdf 파일의 md5 해시 값 확인



[그림 2] 2_encrypt.pdf 파일의 md5 해시 값 확인



[그림 3] 3_clean.pdf 파일의 md5 해시 값 확인

문제로 주어진 파일들의 MD5 해시 값이 일치함을 확인하였습니다.

```
E:\dfc2024\104 - PDF Classification>dir /S
 E 드라이브의 볼륨에는 이름이 없습니다.
 볼륨 일련 번호: 3239-0852

 E:\dfc2024\104 - PDF Classification 디렉터리

2024-08-16  오후 02:44    <DIR>          .
2024-08-11  오후 08:52    <DIR>          ..
2024-08-16  오후 02:44            72,213 104 - PDF Classification.docx
2024-08-16  오후 02:44    <DIR>          pdfs
2024-08-16  오후 02:46             2,544 pdf_analyzer.py
               2개 파일              74,757 바이트

 E:\dfc2024\104 - PDF Classification\pdfs 디렉터리

2024-08-16  오후 02:44    <DIR>          .
2024-08-16  오후 02:44    <DIR>          ..
2024-08-02  오전 02:50           704,228 1_error.pdf
2024-07-22  오후 04:37           687,547 2_encrypt.pdf
2024-07-22  오후 03:55           704,228 3_clean.pdf
               3개 파일           2,096,003 바이트

     전체 파일:
               5개 파일           2,170,760 바이트
               5개 디렉터리  33,530,843,136 바이트 남음
```

[그림 4] 현재 폴더에 존재하는 파일 목록 출력

현재 폴더에는 pdf_analyzer.py 파일과 pdfs 폴더에 3개의 pdf 파일이 존재합니다.

```
E:\dfc2024\104 - PDF Classification>python pdf_analyzer.py
usage: pdf_analyzer.py [-h] --path PATH
pdf_analyzer.py: error: the following arguments are required: --path

E:\dfc2024\104 - PDF Classification>python pdf_analyzer.py -h
usage: pdf_analyzer.py [-h] --path PATH

Analyze PDF files

options:
  -h, --help   show this help message and exit
  --path PATH  Path to the PDF file
```

[그림 5] pdf_analyzer.py 실행 결과

pdf_analyzer.py를 실행했을 때 필요한 argument를 알려주며, -h 옵션을 사용하면 상세한 옵션 설명을 출력해줍니다.

```
E:\dfc2024\104 - PDF Classification>python pdf_analyzer.py --path pdfs
File: 1_error.pdf
  - Corrupted: True
  - Encrypted: False
Total number of pages: 5

Page 1 Structure:
{'/Type': '/Page', '/Parent': IndirectObject(2, 0, 1786596359120), '/Resources': {'/ExtGState': {'/GS5': IndirectObject(
5, 0, 1786596359120), '/GS8': IndirectObject(8, 0, 1786596359120)}, '/Font': {'/F1': IndirectObject(6, 0, 1786596359120)
, '/F2': IndirectObject(9, 0, 1786596359120)}, '/ProcSet': ['/PDF', '/Text', '/ImageB', '/ImageC', '/ImageI']}, '/MediaB
ox': [0, 0, 960, 540], '/Contents': IndirectObject(4, 0, 1786596359120), '/Group': {'/Type': '/Group', '/S': '/Transpare
ncy', '/CS': '/DeviceRGB'}, '/Tabs': '/S', '/StructParents': 0}
Key: /Type, Value: /Page
Key: /Parent, Value: {'/Type': '/Pages', '/Count': 8, '/Kids': [IndirectObject(3, 0, 1786596359120), IndirectObject(11,
0, 1786596359120), IndirectObject(20, 0, 1786596359120), IndirectObject(30, 0, 1786596359120), IndirectObject(38, 0, 178
6596359120), IndirectObject(52, 0, 1786596359120), IndirectObject(60, 0, 1786596359120), IndirectObject(71, 0, 178659635
9120)]}
Key: /Resources, Value: {'/ExtGState': {'/GS5': IndirectObject(5, 0, 1786596359120), '/GS8': IndirectObject(8, 0, 178659
6359120)}, '/Font': {'/F1': IndirectObject(6, 0, 1786596359120), '/F2': IndirectObject(9, 0, 1786596359120)}, '/ProcSet'
: ['/PDF', '/Text', '/ImageB', '/ImageC', '/ImageI']}
Key: /MediaBox, Value: [0, 0, 960, 540]
Key: /Contents, Value: {'/Filter': '/FlateDecode'}
Key: /Group, Value: {'/Type': '/Group', '/S': '/Transparency', '/CS': '/DeviceRGB'}
Key: /Tabs, Value: /S
Key: /StructParents, Value: 0
```

[그림 6] pdf_analyzer.py 분석 결과 - 1

--path 옵션에 pdf 파일들이 존재하는 경로를 지정한 후 코드를 실행하면 각 파일들의 손상 및 암호 설정 유무와 페이지 별 구조를 출력해줍니다.

```
File: 2_encrypt.pdf
  - Corrupted: False
  - Encrypted: True
The file is encrypted and cannot be analyzed.
```

[그림 7] pdf_analyzer.py 분석 결과 - 2

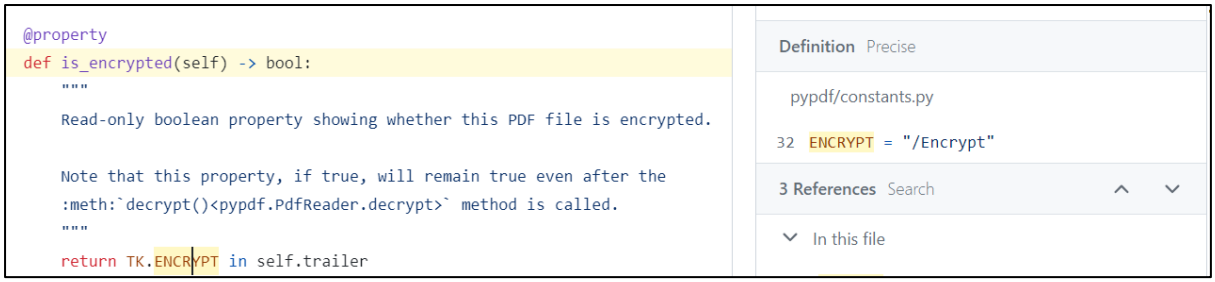
암호화된 파일은 구조를 분석할 수 없기 때문에 위와 같은 문구를 출력합니다.

```
--- Summary Results ---
File: 1_error.pdf, Corrupted: True, Encrypted: False, Type: Corrupted PDF
File: 2_encrypt.pdf, Corrupted: False, Encrypted: True, Type: Encrypted PDF
File: 3_clean.pdf, Corrupted: False, Encrypted: False, Type: Normal PDF
```

[그림 8] pdf_analyzer.py 분석 결과 - 3

마지막으로 분석 결과를 요약하여 문제에서 요구하는 분류 결과를 한 눈에 확인할 수 있으며, Python 에서 실행할 수 있는 스크립트 파일과 라이브러리 설치를 위한 requirements.txt 파일을 압축하여 제출합니다.

다음은 암호화된 파일과 손상된 파일임을 어떻게 판단하는지에 대한 설명입니다.



[그림 9] pypdf에서 제공하는 is_encrypted[1] 함수

[그림 9]는 pypdf에 구현된 is_encrypted 함수의 코드이며, PDF의 Trailer 영역에 /Encrypt가 있는지 확인하여 암호화 유무를 판단합니다.

```
>>> from pypdf import PdfReader
>>> e = PdfReader("2_encrypt.pdf")
>>> e.is_encrypted
True
>>> e.trailer
{'/Root': IndirectObject(687, 0, 1749566681936), '/Encrypt': IndirectObject(686, 0, 1749566681936), '/Info': IndirectObj
ect(684, 0, 1749566681936), '/ID': ['-.\x04ñ£#\x02F‹¤ÞÒþ[Ç8', '¦I–K2Ã‡G•Á…Y‹jÓ|'], '/Size': 702}
>>> "/Encrypt" in e.trailer
True
>>> e = PdfReader("3_clean.pdf")
>>> e.is_encrypted
False
>>> e.trailer
{'/Size': 688, '/Root': IndirectObject(1, 0, 1749555598480), '/Info': IndirectObject(73, 0, 1749555598480), '/ID': ['-.\
x04ñ£#\x02F‹¤ÞÒþ[Ç8', '-.\x04ñ£#\x02F‹¤ÞÒþ[Ç8'], '/Prev': 690125, '/XRefStm': 688365}
>>> "/Encrypt" in e.trailer
False
```

[그림 10] is_encrypted 함수의 수행 과정

[그림 10]은 is_encrypted 함수가 동작하는 과정을 확인한 결과입니다. 2_encrypt.pdf 파일의 경우 /Encrypt가 존재하지만, 3_clean.pdf 파일은 /Encrypt 값이 존재하지 않습니다.

```
>>> from pypdf import PdfReader
>>> e = PdfReader("1_error.pdf")
Ignoring wrong pointing object 46 0 (offset 271320)
Ignoring wrong pointing object 47 0 (offset 271574)
Ignoring wrong pointing object 48 0 (offset 272144)
Ignoring wrong pointing object 49 0 (offset 348690)
Ignoring wrong pointing object 50 0 (offset 348947)
Ignoring wrong pointing object 51 0 (offset 349542)
```

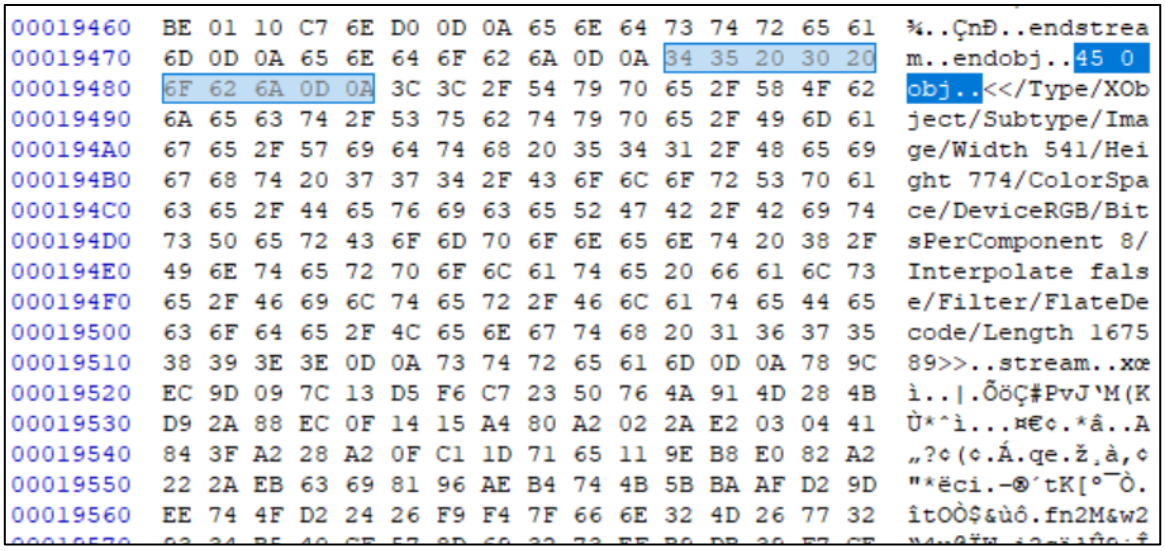
[그림 11] pypdf를 사용하여 1_error.pdf 파일을 분석한 결과

[1] https://github.com/py-pdf/pypdf/blob/82eac7e316f8f785d00ed600f8ba4aba3296a4a8/pypdf/_reader.py#L1075

```python
# remove wrong objects (not pointing to correct structures) - cf #2326
if not self.strict:
    loc = stream.tell()
    for gen, xref_entry in self.xref.items():
        if gen == 65535:
            continue
        ids = list(xref_entry.keys())
        for id in ids:
            stream.seek(xref_entry[id], 0)
            try:
                self.read_object_header(stream)
            except ValueError:
                logger_warning(
                    f"Ignoring wrong pointing object {id} {gen} (offset {xref_entry[id]})",
                    __name__,
                )
                del xref_entry[id]  # we can delete the id, we are parsing ids
    stream.seek(loc, 0)  # return to where it was
```

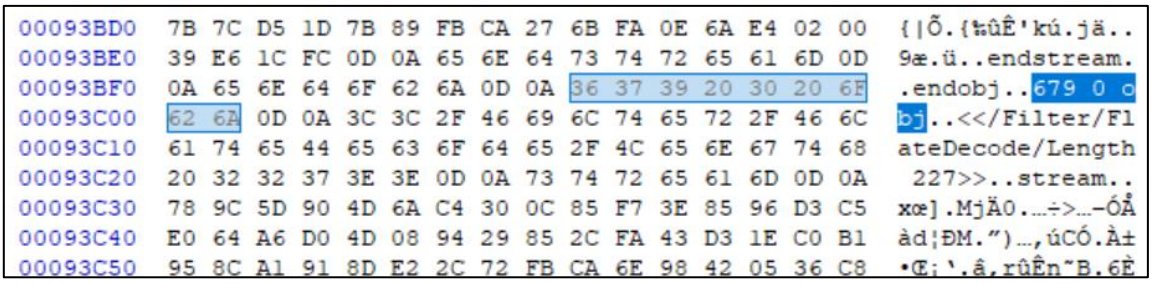
[그림 12] pypdf에서 손상된 파일을 처리하는 코드[2]

[그림 12]의 코드를 보면 xref entry 별 header를 읽어오는 과정에서 에러가 발생할 경우 [그림 11]과 같은 에러가 발생하게 됩니다. [그림 12] 코드의 주석에 적혀 있듯이 object가 삭제되어 구조가 올바르지 않은 상태입니다.



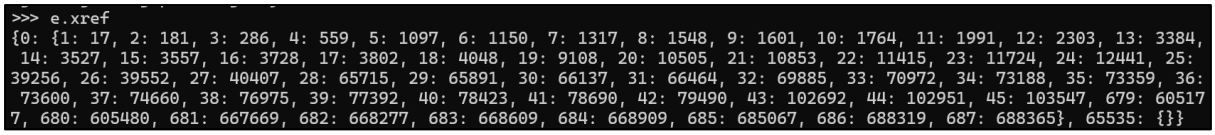
[그림 13] obj 문자를 기준으로 검색한 결과 – 1

[2] https://github.com/py-pdf/pypdf/blob/82eac7e316f8f785d00ed600f8ba4aba3296a4a8/pypdf/_reader.py#L614

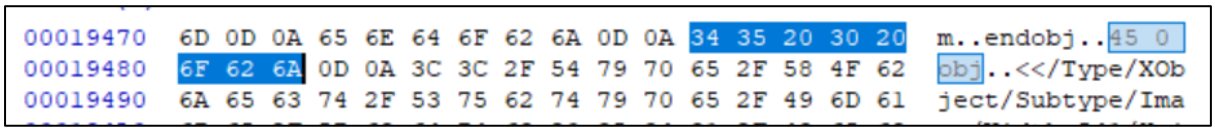


[그림 14] obj 문자를 기준으로 검색한 결과 – 2

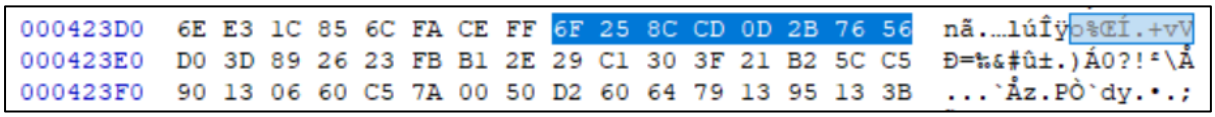
실제로 HxD를 사용하여 obj 문자를 기준으로 검색했을 때 45 0 obj 이후 679 0 obj가 나오는 것을 확인하였으며, 비정상적인 패턴임을 확인할 수 있습니다.



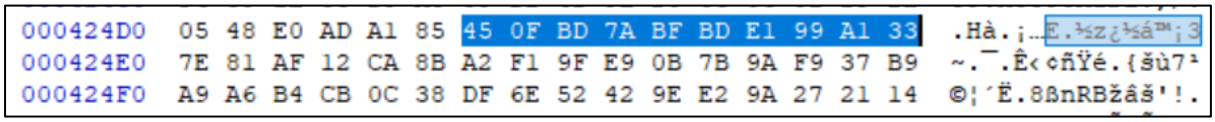
[그림 15] pypdf 라이브러리가 파싱한 xref table 정보



[그림 16] 103547 오프셋에 존재하는 데이터



[그림 17] 271320 오프셋에 존재하는 데이터



[그림 18] 271574 오프셋에 존재하는 데이터

또한 [그림 15]에서 pypdf 라이브러리가 정상적으로 파싱한 xref table의 주소를 따라갔을 때 [그림 16]처럼 보이지만, [그림 11]에서 Ignoring 문구가 뜬 offset으로 가보면 [그림 17], [그림 18]과 같이 이상한 값이 존재함을 확인하였습니다.