# 102 – Find all graphics

## Team Information

**Team Name : DogeCoin**

**Team Member : Dongbin Oh, Donghyun Kim, Donghyun Kim, Yeongwoong Kim**

**Email Address : dfc-dogecoin@naver.com**

## Instructions

### Description

As a forensic expert, you must identify all graphics in the target image.

| Target | Hash (MD5) |
|---|---|
| 102.ad1 | 996d4a53947afe89c202af0887bf3ad3 |

### Questions

Find all visual images and fill out the *102.csv*.

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: | FTK Imager | Publisher: | AccessData |
|---|---|---|---|
| Version: | 4.5 | | |
| URL: | https://accessdata.com/product-download/ftk-imager-version-4-5 | | |

| Name: | Python | Publisher: | Python Software Foundation |
|---|---|---|---|
| Version: | 3.9.4 | | |
| URL: | https://www.python.org/downloads/ | | |

| Name: | pyad1 | Publisher: | Petter Christian Bjelland |
|---|---|---|---|
| Version: | 0.0.1 | | |
| URL: | https://github.com/pcbje/pyad1 | | |

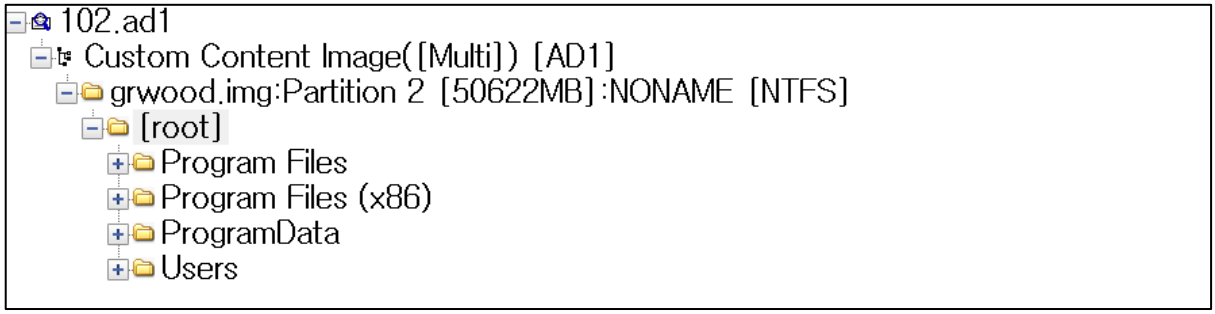
| Name: | binwalk | Publisher: | ReFirmLabs |
|---|---|---|---|
| Version: | 2.3.1 | | |
| URL: | https://github.com/ReFirmLabs/binwalk | | |

| Name: | MD5 File | Publisher: | MD5 File |
|---|---|---|---|
| Version: | - | | |
| URL: | https://md5file.com/calculator | | |

| Name: | Thumb Cache Viewer | Publisher: | Eric Kutcher |
|---|---|---|---|
| Version: | 1.0.3.6 | | |
| URL: | https://thumbcacheviewer.github.io/ | | |

## Step-by-step methodology:

### 1. 요구 사항 파악 및 설정




**[그림 1] 제공받은 이미지 파일 내부 확인 (Input)**

102

| No | Format | Absolute Path | Embebbed | Relative Path | Hash (MD5) |
|---|---|---|---|---|---|
| 1 | BMP | C:\sample.bmp | N | | f5d72bfdf0facd5b90f7e5334bef4f57 |
| 2 | JPEG | C:\sample.pptx | Y | ppt\media\image1.jpeg | 22524d2ec91314fe3a1fcad7054f0848 |
| 3 | JPEG | C:\sample.pptx | Y | ppt\media\image2.jpeg | b4c61d0fa4e652c0135bb51a2cb9afbb |
| 4 | BMP | C:\.....\aaa.db | Y | | be83ab3ecd0db773eb2dc1b0a17836a1 |
| 5 | GIF | C:\.....\bbb.zip | Y | sample.gif | 9806b902e654c2ddd3b611099436037a |
| | | | | | |

**[그림 2] 제출을 요구하는 CSV의 내부 형태 (Output)**

문제에서 제공한 디스크 이미지의 내부를 확인한 결과, Windows 운영체제에 대한 파일 일부가 덤프 되어 있었습니다. 또한 최종적으로 제출을 요구 받은 "102.csv"의 내부 구조는 위의 사진과 같이 정의되어 있었습니다.

이를 토대로 아래와 같은 요구사항을 설정하였습니다.

- **그래픽 파일 (이미지 포맷) 에 대한 설정**

  실제로 존재하는 모든 이미지 포맷에 대해 정의하고 이에 대한 정보를 모으는 것은 불가능한 일이므로, CSV **내부에 기재된 이미지의 포맷과 더불어 범용적으로 사용되는 이미지 포맷** (PNG, JPEG, BMP, GIF) 을 가진 이미지들에 대해서만 탐색하도록 설정하였습니다.

- **일반 이미지 파일에 대한 해시 값 구하기**

  위에서 정의한 이미지 파일이 단독으로 존재할 시, 이에 대한 해시 값을 계산하여 기록합니다. 다만 경우에 따라 확장자가 변조되었거나 존재하지 않는 경우도 있으므로 이에 대해서도 예외처리를 해야 합니다.

- **문서 파일 내부에 존재하는 이미지 파일에 대한 해시 값과 상대 경로 구하기**

  문서 파일 내부에는 다양한 이미지가 존재할 수 있으며, 일반적인 MS Office 문서 포맷 (DOCX, PPTX, XLSX) 를 대상으로 내부에 이미지가 존재할 시, 이에 대한 해시 값과 상대 경로를 기록합니다.

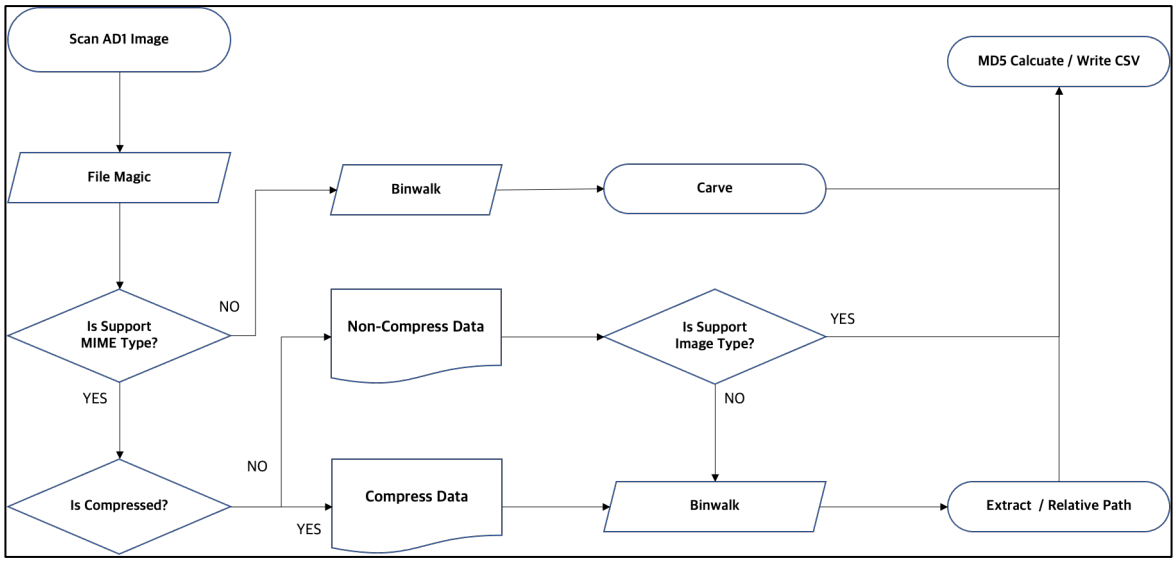
- **비압축 파일에 포함된 이미지 파일에 대한 해시 값 구하기**

  Windows 운영체제에서 사용하는 파일이나 일부 파일은 자체적인 포맷을 사용하여 내부에 이미지를 저장하는 경우가 있습니다. 이에 파일 내부를 탐색하여 존재하는 이미지에 대한 해시 값을 기록합니다.

- **압축 파일 내부에 존재하는 이미지 파일에 대한 해시 값 구하기**

  압축 파일 (ZIP, 7z, RAR 등) 내부에도 다양한 이미지가 존재할 수 있으므로, 문서 파일과 유사하게 압축 파일 내부의 이미지에 대한 해시 값과 상대 경로를 기록합니다.

## 2. 문제 해결 과정




**[그림 3] 디스크 내에 존재하는 이미지 식별 Logic**

파악한 요구사항을 기반으로 이미지들을 식별하기 위해, 위의 그림과 같은 Logic을 설계하였습니다.

위의 이미지는 AD1 이미지 내부에 존재하는 데이터를 순회하면서 발생하는 아래의 과정을 나타냅니다..

### 1) 별도의 압축 해제 과정이나 카빙 과정이 필요하지 않은 이미지 파일 식별

본 풀이에서는 **File Magic**을 이용하였고, 이를 이용해서 파일들의 타입을 검증할 수 있을 뿐만 아니라, 확장자가 없는 이미지 파일 (웹 브라우저 캐시 등) 을 식별할 수 있게 됩니다.

### 2) 압축 및 문서 파일에 대한 압축 해제 및 이미지 파일 식별

압축 및 문서 파일 내부에 존재하는 이미지를 파악 하기 위해 사전에 정의한 파일 타입이 **File Magic**에서 탐지될 경우, **Binwalk** 모듈을 통해서 압축 파일을 해제하여 내부의 이미지 파일들을 식별할 수 있게 됩니다.

### 3) 비압축 파일 내부에 저장된 이미지 파일 식별 및 카빙

비압축 파일은 포맷이 다양하므로, 파일 포맷에 근간한 이미지 탐색 방식보다, 이미지 파일의 시그니처를 활용한 파일 카빙 방식을 채택하였습니다. 이전과 동일하게 **Binwalk** 모듈을 통해서 일반 바이너리 파일에서 정의한 시그니처에 맞게 이미지를 식별할 수 있게 됩니다, 다만 **Binwalk**에서 시그니처를 활용한 파일 추출은 더미 데이터가 포함된 채로 추출될 수 있으므로 각 이미지 타입에 맞게 카빙하는 과정을 거치게 됩니다.

이러한 과정들을 통해서 각 파일 별로 정보를 수집하게 되면 CSV 형태로 저장하는 것으로 식별을 종료합니다.

이에 대한 상세 코드와 내용은 아래와 같습니다.

## (1) AD1 이미지 스캔

```python
def scan_disk_image(self, image_name, csvwr):
    count_no = 0
    with AD1Reader(image_name) as ad1:
        for item_type, folder, filename, metadata, content in ad1:
            if item_type == self.item_folder_value:
                continue
            else:
                absolute_path = self.create_absolute_path(folder, filename)
                print(absolute_path)
                result = self.process_content(content, absolute_path)
```

**[그림 4] 디스크 이미지 Read 코드 (grep.py, Line 32)**

주어진 AD1 이미지 내부의 파일에 접근하기 위해서는 AD1 이미지에 대한 분석이 필요합니다. 다양한 방법 중 “pyad1”이라는 Python 모듈은 AD1 이미지를 분석하여 내부의 파일 엔트리들에 대해 Iteration하게 사용할 수 있습니다. 이에 해당 모듈로 **AD1 이미지 내부의 엔트리 정보를 불러오되, 아이템 타입 (item_type) 의 값이 폴더 (5) 에 해당 할 시, 무시하도록 하여 파일 엔트리에 대해서만 수집**하여 처리하도록 하였습니다.

```python
def process_content(self, content, absolute_path):
    detect_mime = self.magic.check_magic(content)
    result = self.magic.check_domain(self.defined_mime, detect_mime)
    if(result):
        mime = result[0]
        return self.call_content(detect_mime, mime, content, absolute_path)
    else:
        return {}
```

**[그림 5] File Magic을 통해서 File Type을 검사하는 Logic**

수집한 파일 엔트리에서 파일 경로와 파일 데이터를 수집할 수 있습니다. 이를 **“Magic” 모듈의 메서드와 연계하여 해당 데이터가 어떠한 File Type을 가지는지 검사하고 이를 필터링** 할 수 있습니다. 문제에서는 **일반 이미지 파일, 문서 파일 및 압축 파일이 그 대상이므로 “dict.json”에 필터링할 File Type을 사전에 명시**하였습니다.

```python
def check_magic(self, file_buffer):
    magic_handler = Magic(mime=True)
    result = magic_handler.from_buffer(file_buffer)
    return result

def load_dictionary(self):
    with open('./modules/res/dict.json') as dict:
        json_data = load(dict)
    defined_mimes = json_data["filterMIME"]
    return defined_mimes

def check_support_mime(self, defined_mimes, mime):
    defined_mimes_values = defined_mimes.values()
    flatten_func = lambda mimes: [mime for defined_mimes in mimes for mime in defined_mimes]
    return mime in flatten_func(defined_mimes_values)

def check_domain(self,  defined_mimes, mime):
    return [domain for domain, mimes in defined_mimes.items() if mime in defined_mimes[domain]]
```

**[그림 6] File Buffer 데이터에 기반해 File Type을 알아내는 Magic Method**

```
{
 "filterMIME": {
   "image": [
     "image/gif",
     "image/bmp",
     "image/jpeg",
     "image/png",
```

**[그림 7] 사전에 정의된 Magic Type JSON**

```python
def call_content(self, detect_mime, mime, content, absolute_path):
    if(mime == "image"):
        data = [{
            "format": self.get_format(detect_mime),
            "absolute_path": absolute_path,
            "embedded": "N",
            "relative_path": " ",
            "hash": hashlib.md5(content).hexdigest()
        }]
        return data
    elif(mime == "docs" or mime == "compress"):
        data = self.binary_walk(absolute_path, content)
        return data
    else:
        data = self.binary_carv(absolute_path, content)
        return data
```

**[그림 8] File Magic을 통해 필터링된 File Type에 따라 분기하는 Logic**

File Magic 메서드를 통해서 이미지로 판별될 경우, 별도의 Relative Path을 가지지 않으므로 File Buffer을 통해 MD5 값만 계산하여 Return하도록 하였습니다. 그러나 엔트리의 Magic 결과가 문서나 압축파일로 식별될 경우, Binwalk를 통해서 Extract 과정을 거치게 됩니다.

```python
def run_binwalk(self, file_name, absolute_path):
    binwalk_result = []
    for data in binwalk.scan(file_name, signature=True, quiet=True, extract=True, directory="./temp"):
        for app in data.results:
            if self.check_eof(app.description):
                continue
            if not self.check_metadata(app.description):
                continue
            if app.file.path in data.extractor.output:
                scan_result = self.scan_image(file_name, absolute_path, app.description)
                if(scan_result):
                    binwalk_result.append(scan_result)
    print(binwalk_result)
    self.clean_up_temp()
    return binwalk_result
```

**[그림 9] Binwalk를 통해서 내부의 파일에 대해 탐색하는 Logic**

이전에 Magic을 통해서 필터링한 압축 및 문서 형태의 파일 내부의 이미지를 식별하기 위해 Binwalk을 수행하였습니다. 원래 Binwalk는 바이너리 형태로 존재하나, Binwalk 파이프라인 형태의 Python API을 지원하여 이를 활용하였습니다.

Binwalk을 통해 내부 파일을 압축해제하고, 이를 순회하는 과정에서 이미지 파일이 확인될 경우, 이에 대한 메타 데이터 (Relative Path, MD5 Hash) 을 List에 담아 최종적으로 Return 하게 됩니다. (압축 파일이나 문서 파일은 여러 개의 이미지를 가질 수 있으므로 List을 이용)

```python
def carv_binwalk(self, origin_file, absolute_path):
    carv_datas = []
    binwalk.scan(origin_file, signature=True, quiet=True, extract=True, invalid=True, include=["png","jpeg","jpg","gif", "bmp", "bitmap"], rm=True,
                 dd=["png:png", "jpeg:jpeg", "jpg:jpg", "gif:gif", "bitmap:bmp", "bmp:bmp"])
    file_name = os.path.split(origin_file)[1]
    extract_path = "./temp/" + "_" + file_name + ".extracted/"
    file_list = self.absoluteFilePaths(extract_path)
    for file in file_list:
        carv_datas.append(self.travel_carv(file, absolute_path))
    return carv_datas
```

**[그림 10] 파일 시그니처 탐색 및 카빙을 위한 Binwalk 메서드**

비압축 파일 내부를 탐색하기 위해서도 위의 사례와 동일하게 Binwalk를 사용합니다. 그러나 추출과 탐지의 정확도를 위해 몇가지 옵션을 추가적으로 사용합니다. 주요한 인자에 대한 설명은 다음과 같습니다.

**[표 1] Binwalk에 사용된 인자 값**

| |
|---|
| - **include** : 추출 대상이 되는 확장자 (시그니처)<br>- **rm** : 사이즈가 0 바이트인 것과 같이 파일의 덤프가 불가능한 케이스를 무시한다.<br>- **dd** : 탐지한 추출 대상을 덤프 함에 있어 사용할 확장자 정의 |

해당 인자 값을 통해서 비압축 파일들에 대해서 내부에 정의한 파일 시그니처가 존재할 시, 이미지 정보를 가져옴에 있어 무의미한 데이터 (zlib 등) 를 제외하고 덤프를 수행하게 됩니다.

```python
def carv_get_hash(self, file_path, mime):
    if(mime == "image/jpeg" or mime == "image/jpg"):
        with open(file_path, 'rb') as fs:
            data = fs.read()
            eof = data.find(b'\xff\xd9')
            fs.seek(0)
            carv_data = fs.read(eof+2)
        return hashlib.md5(carv_data).hexdigest()
    if (mime == "image/png"):
        with open(file_path, 'rb') as fs:
            data = fs.read()
            eof = data.find(b'\x49\x45\x4E\x44\xAE\x42\x60\x82')
            fs.seek(0)
            carv_data = fs.read(eof + 8)
```

**[그림 11] 시그니처 및 EOF 기반 파일 카빙 메서드**

위의 옵션들을 통해서 대부분의 파일들이 잘 추출되지만, 테스트 과정에서 Dummy Data가 파일에 포함되는 경우도 존재했습니다. 이러한 파일의 경우, MD5 해시 값 계산에 큰 영향을 주게 되므로 이에 대한 카빙을 수행하는 로직을 추가적으로 수행합니다. 각 이미지 타입에 맞게 정의된 카빙 로직은 아래와 같습니다.

**[표 2] 이미지 유형별 카빙 로직**

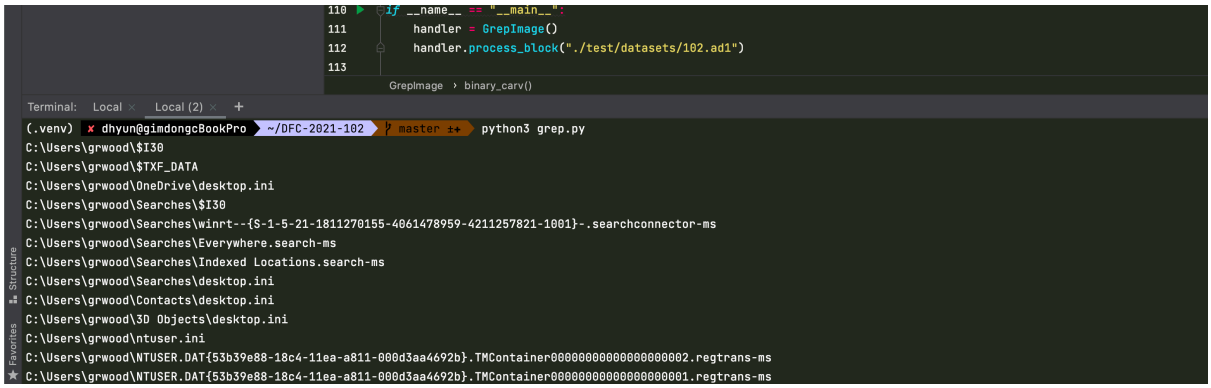
| |
|---|
| - **PNG**<br>Footer에 해당하는 값 **(\x49\x45\x4E\x44\xAE\x42\x60\x82)**의 오프셋을 찾아서<br>해당 크기만큼 버퍼를 읽어 들인다.<br>- **JPG/JPEG**<br>Footer에 해당하는 값 **(\xff\xd9)**의 오프셋을 찾아서 해당 크기만큼 버퍼를 읽어 들인다.<br>- **GIF**<br>Footer에 해당하는 값 **(\x00\x3B)**의 오프셋을 찾아서 해당 크기만큼 버퍼를 읽어 들인다.<br>- **BMP**<br>Header의 2 Byte 지점에 **Size Offset을 Little Endian으로 변환**하여 해당 크기만큼 버퍼를 읽어 들인다. |

```python
if (result):
    for row in result:
        count_no += 1
        csvwr.writerow([
            count_no,
            row["format"],
            row["absolute_path"],
            row["embedded"],
            row["relative_path"],
            row["hash"]
        ])
```

**[그림 12] Return 데이터를 CSV에 Write하는 Logic**

위의 과정들을 통해 이미지라고 판단된 파일 엔트리에 대한 정보를 Return 받았다면, 해당 정보를 CSV에 기록하는 것으로 이미지 식별 코드는 종료됩니다.

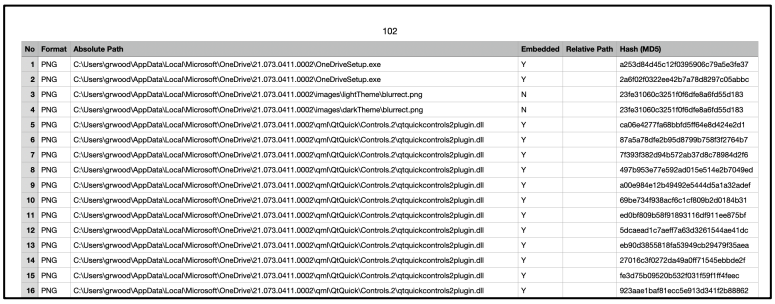
## 3. 실행 및 결과 검증




**[그림 13] 도구 실행 과정**

동봉된 requirements.txt에 기재된 패키지를 PIP로 설치하고 “grep.py”를 실행함으로써 도구를 동작시킬 수 있다. 다른 이미지 파일을 사용하거나 변경을 원할 때는 112번째줄에서 경로를 수정할 수 있습니다.

프로그램을 실행할 시, 주어진 AD1 이미지를 기반으로 “102.csv”를 생성할 수 있었으며 본 하이퍼링크를 통해 사용된 코드와 CSV을 다운로드 할 수 있습니다.



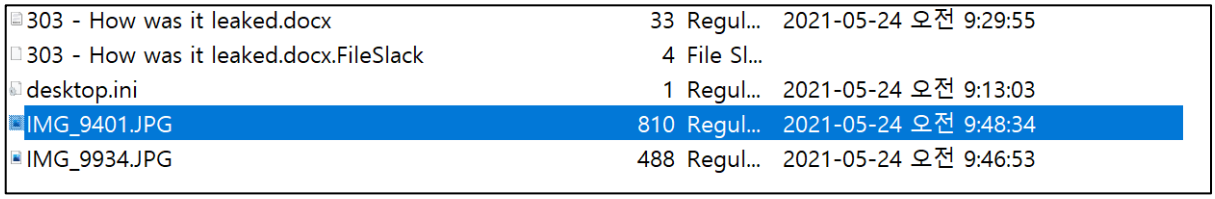
102

| No | Format | Absolute Path | Embedded | Relative Path | Hash (MD5) |
|---|---|---|---|---|---|
| 1 | PNG | C:\Users\grwood\AppData\Local\Microsoft\OneDrive\21.073.0411.0002\OneDriveSetup.exe | Y | | a253d84d45c12f0395906c79a5e3fe37 |
| 2 | PNG | C:\Users\grwood\AppData\Local\Microsoft\OneDrive\21.073.0411.0002\OneDriveSetup.exe | Y | | 2a6f02f0322ee42b7a78d8297c05abbc |
| 3 | PNG | C:\Users\grwood\AppData\Local\Microsoft\OneDrive\21.073.0411.0002\images\lightTheme\blurrect.png | N | | 23fe31060c3251f0f6dfe8a6fd55d183 |
| 4 | PNG | C:\Users\grwood\AppData\Local\Microsoft\OneDrive\21.073.0411.0002\images\darkTheme\blurrect.png | N | | 23fe31060c3251f0f6dfe8a6fd55d183 |
| 5 | PNG | C:\Users\grwood\AppData\Local\Microsoft\OneDrive\21.073.0411.0002\qml\QtQuick\Controls.2\qtquickcontrols2plugin.dll | Y | | ca06e4277fa68bbfd5ff64e8d424e2d1 |
| 6 | PNG | C:\Users\grwood\AppData\Local\Microsoft\OneDrive\21.073.0411.0002\qml\QtQuick\Controls.2\qtquickcontrols2plugin.dll | Y | | 87a5a78dfe2b95d8799b758f3f2764b7 |
| 7 | PNG | C:\Users\grwood\AppData\Local\Microsoft\OneDrive\21.073.0411.0002\qml\QtQuick\Controls.2\qtquickcontrols2plugin.dll | Y | | 7f393f382d94b572ab37d8c78984d2f6 |
| 8 | PNG | C:\Users\grwood\AppData\Local\Microsoft\OneDrive\21.073.0411.0002\qml\QtQuick\Controls.2\qtquickcontrols2plugin.dll | Y | | 497b953e77e592ad015e514e2b7049ed |
| 9 | PNG | C:\Users\grwood\AppData\Local\Microsoft\OneDrive\21.073.0411.0002\qml\QtQuick\Controls.2\qtquickcontrols2plugin.dll | Y | | a00e984e12b49492e5444d5a1a32adef |
| 10 | PNG | C:\Users\grwood\AppData\Local\Microsoft\OneDrive\21.073.0411.0002\qml\QtQuick\Controls.2\qtquickcontrols2plugin.dll | Y | | 69be734f938acf6c1cf809b2d0184b31 |
| 11 | PNG | C:\Users\grwood\AppData\Local\Microsoft\OneDrive\21.073.0411.0002\qml\QtQuick\Controls.2\qtquickcontrols2plugin.dll | Y | | ed0bf809b58f91893116df911ee875bf |
| 12 | PNG | C:\Users\grwood\AppData\Local\Microsoft\OneDrive\21.073.0411.0002\qml\QtQuick\Controls.2\qtquickcontrols2plugin.dll | Y | | 5dcaead1c7aeff7a63d3261544ae41dc |
| 13 | PNG | C:\Users\grwood\AppData\Local\Microsoft\OneDrive\21.073.0411.0002\qml\QtQuick\Controls.2\qtquickcontrols2plugin.dll | Y | | eb90d3855818fa53949cb29479f35aea |
| 14 | PNG | C:\Users\grwood\AppData\Local\Microsoft\OneDrive\21.073.0411.0002\qml\QtQuick\Controls.2\qtquickcontrols2plugin.dll | Y | | 27016c3f0272da49a0ff71545ebbde2f |
| 15 | PNG | C:\Users\grwood\AppData\Local\Microsoft\OneDrive\21.073.0411.0002\qml\QtQuick\Controls.2\qtquickcontrols2plugin.dll | Y | | fe3d75b09520b532f031f59f1ff4feec |
| 16 | PNG | C:\Users\grwood\AppData\Local\Microsoft\OneDrive\21.073.0411.0002\qml\QtQuick\Controls.2\qtquickcontrols2plugin.dll | Y | | 923aae1baf81ecc5e913d3412b88862 |

**[그림 14] 최종적으로 생성된 102.CSV**

해당 CSV에 저장된 데이터가 정확하게 덤프 된 것인지 확인하기 위해, 디스크 이미지 내에 여러 파일을 임의 선별하여 실제 이미지 파일 내부의 여러 케이스와 비교하였습니다.

### 1) 별도의 압축 해제 과정이나 카빙 과정이 필요하지 않은 이미지 파일



| | | | |
|---|---|---|---|
| 303 - How was it leaked.docx | 33 | Regul... | 2021-05-24 오전 9:29:55 |
| 303 - How was it leaked.docx.FileSlack | 4 | File Sl... | |
| desktop.ini | 1 | Regul... | 2021-05-24 오전 9:13:03 |
| IMG_9401.JPG | 810 | Regul... | 2021-05-24 오전 9:48:34 |
| IMG_9934.JPG | 488 | Regul... | 2021-05-24 오전 9:46:53 |

**[그림 15] AD1 이미지 내부에 존재하는 일반 이미지 파일**

| | |
|---|---|
| **IMG_9401.JPG** | (image/jpeg) - 829000 bytes |
| MD5 | 4ce4a1a49739ec6339a58b3f1db27334 |

**[그림 16] 추출 및 MD5 계산 결과**

| | | | | | | |
|---|---|---|---|---|---|---|
| 334 | JPEG | C:\Users\grwood\Downloads\IMG_9934.JPG | N | | 66a3d07352399d8cd6af2f70e0ce0c70 |
| 334 | JPEG | C:\Users\grwood\Downloads\IMG_9401.JPG | N | | 4ce4a1a49739ec6339a58b3f1db27334 |
| 334 | PNG | C:\Users\grwood\Downloads\301 - What is the secret information.docx | Y | word/media/image1.png | a7c235ddfef8582a31d4f1511cc56eed |
| 334 | JPEG | C:\ProgramData\Microsoft\Windows NT\MSScan\WelcomeScan.jpg | N | | 73d4281e46a68222934403627e5b4e19 |
| 335 | BMP | C:\ProgramData\Microsoft\Windows Defender\Scans\mpcache-092C61B417250F9CB3B47BC1066C308123328FE6.bin.67 | Y | | f3371e427ea0e64b5be4e91b0f1efa32 |
| 335 | BMP | C:\ProgramData\Microsoft\Windows Defender\Scans\mpcache-092C61B417250F9CB3B47BC1066C308123328FE6.bin.67 | Y | | abd779c30b066539262231b828855485 |

[그림 17] CSV에 기록된 일반 이미지 파일 정보

## 2) 문서 파일 내부에 존재하는 이미지 파일에 대한 해시 값과 상대 경로 구하기

| | | | |
|---|---|---|---|
| 20201112_085547.jpg | 3,532 | Regul... | 2021-05-24 오전 9:53:26 |
| 301 - What is the secret information.docx | 101 | Regul... | 2021-05-24 오전 9:25:47 |
| 301 - What is the secret information.docx.FileSlack | 4 | File Sl... | |
| 302 - User Behavior Analysis.docx | 29 | Regul... | 2021-05-24 오전 9:26:16 |

[그림 18] AD1 이미지 내부에 존재하는 문서 파일


[그림 19] 문서 파일 내부에 존재하는 이미지 파일

| | |
|---|---|
| **image1.png** | (image/png) - 72403 bytes |
| MD5 | a7c235ddfef8582a31d4f1511cc56eed |

[그림 20] 추출 및 MD5 계산 결과

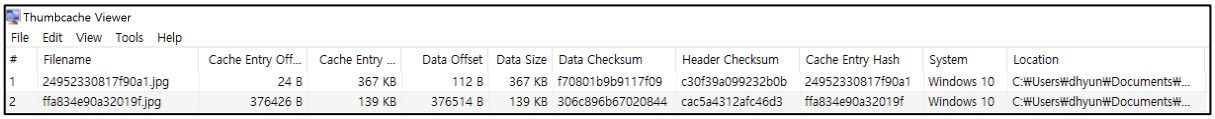
| | | | | | | |
|---|---|---|---|---|---|---|
| 334 | JPEG | C:\Users\grwood\Downloads\IMG_9934.JPG | N | | 66a3d07352399d8cd6af2f70e0ce0c70 |
| 334 | JPEG | C:\Users\grwood\Downloads\IMG_9401.JPG | N | | 4ce4a1a49739ec6339a58b3f1db27334 |
| 334 | PNG | C:\Users\grwood\Downloads\301 - What is the secret information.docx | Y | word/media/image1.png | a7c235ddfef8582a31d4f1511cc56eed |
| 334 | JPEG | C:\ProgramData\Microsoft\Windows NT\MSScan\WelcomeScan.jpg | N | | 73d4281e46a68222934403627e5b4e19 |
| 335 | BMP | C:\ProgramData\Microsoft\Windows Defender\Scans\mpcache-092C61B417250F9CB3B47BC1066C308123328FE6.bin.67 | Y | | f3371e427ea0e64b5be4e91b0f1efa32 |

[그림 21] CSV에 기록된 문서 내부의 이미지 파일 정보

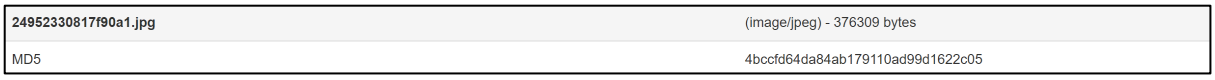
## 3) 비압축 파일 내부에 저장된 이미지 파일 식별 및 카빙

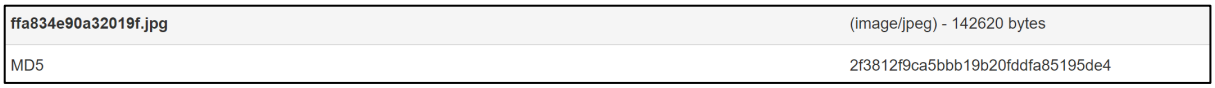
| | | | |
|---|---|---|---|
| thumbcache_1280.db | 1,024 | Regul... | 2021-05-24 오전 11:34:09 |
| thumbcache_1280.db | | $I30 I... | |
| thumbcache_1280.db.FileSlack | 516 | File Sl... | |
| thumbcache_16.db | 1,024 | Regul... | 2021-05-24 오전 9:13:10 |
| thumbcache_16.db.FileSlack | 1,020 | File Sl... | |
| thumbcache_1920.db | 1 | Regul... | 2021-05-24 오전 9:13:10 |
| thumbcache_256.db | 1,024 | Regul... | 2021-05-24 오전 9:13:10 |

[그림 22] AD1 이미지 내부에 존재하는 이미지가 내장된 비압축형 파일

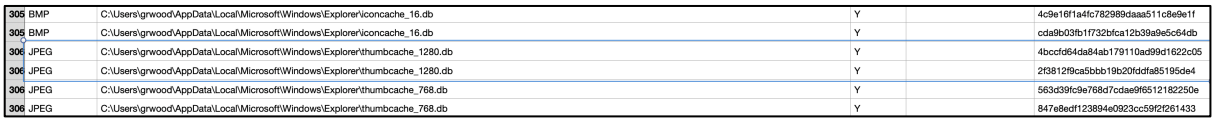



**[그림 23] Thumbcache Viewer로 확인한 비압축형 파일 내부에 존재하는 이미지 파일**




**[그림 24] 추출 및 MD5 계산 결과**




**[그림 25] CSV에 기록된 비압축형 파일 내부의 이미지 파일 정보**

해당 검증 과정을 통해 정의한 시나리오에 맞게, 디스크 이미지 내부의 여러 이미지 파일들을 수집하고 정상적으로 정보가 취합 되었음을 확인하였습니다.