# 악성코드 탐지, 암호화 및 패키징 프로젝트

데이터셋 가공 보고서

문서 번호 : DSP-002



**목**

**차**

1. **데이터셋 가공 개요**
2. **데이터베이스 구성**
3. **데이터셋 가공 모듈**
4. **결론**
5. 데이터셋 가공 개요

데이터셋 수집에서Malware Bazzar abuse를 최종 선택했습니다. 아래는 Malware Bazzar abuse dataset에 포함된 정보입니다. 저희는 이 데이터를 Virustotal API Module을 동작시킨 후 데이터와의 불균형을 해소하기 위해서 이 데이터 셋에 있는 MD5 HASH를 이용해 Virustotal API의 file hash 검색을 통해 데이터를 가공할 것입니다.

**포함된 정보**

|  |  |
| --- | --- |
| **이름** | **설명** |
| first\_seen\_ufc | 바이러스가 처음 보고된 시간 |
| sha256\_hash | 파일의 SHA-256 해시 값 |
| md5\_hash | 파일의 md5 해시 값 |
| sha1\_hash | 파일의 SHA-1 해시 값 |
| reporter | 바이러스를 보고한 사람 |
| file\_name | 바이러스 파일의 이름 |
| file\_type\_guess | 바이러스 파일의 형식(추정값) |
| mime\_type | 바이러스 파일의 MIME 유형 |
| signature | 바이러스의 정의된 이름(ex. Loki) |
| clamav | Clamav 동작 후 나온 값(ex. RedLineStealer) |
| vtpercent | Virustotal 검사 점수 |
| imphash | PE 파일의 import table 해시 값 |
| ssdeep | 파일 간 유사성을 비교하기 위한 해시 값 |
| tlsh | TLSH 해시 값. 파일의 유사성을 비교하기 위한 해시 |

가공될 데이터의 구조는 VirustotalAPI 데이터베이스에 있는 info Collection입니다. info collection은 md5, Details, Behavior로 구성되어 있습니다. 상세 설명은2. 데이터베이스 구성에 있습니다.

1. 데이터베이스 구성

저희 데이터 베이스는 Virustotal API를 통해 파일을 분석하는 데이터베이스와 파일 프로텍터가 적용될 데이터 베이스가 있습니다. 기존 Malware Bazzar의 데이터셋을 재가공 할 데이터베이스는 Virustotal API의 데이터베이스 안에 있는 info collection입니다.

MongoDB VirustotalAPI 데이터베이스 구성

# 1 files 컬렉션

업로드된 파일에 대한 메타데이터와 파일 데이터

|  |  |
| --- | --- |
| **필드** | **설명** |
| signature\_id | 파일의 고유 서명 ID |
| filehash | MD5 해쉬 정보 |
| filename | 파일의 이름 |
| file\_data | 파일의 전체 내용 |
| upload\_time | 파일이 업로드된 시간 |
| upload\_ip | 파일을 업로드한 사용자의 IP 주소 |

# 2 info 컬렉션

파일의 details 정보와 파일 behavior 정보

## Details

**Hash** :

|  |  |
| --- | --- |
| **필드** | **설명** |
| md5 | 파일의 MD5 해시 값 |
| sha1 | 파일의 SHA1 해시 값 |
| sha256 | 파일의 SHA256 해시 값 |
| vhash | 파일의 VHash 값 |
| auth\_hash | 인증 해시 값 |
| imphash | Import Hash 값 |
| ssdeep | SSDEEP 해시 값 |
| tlsh | TLSH 해시 값 |

**file\_info** :

|  |  |
| --- | --- |
| **필드** | **설명** |
| md5 | 파일의 MD5 해시 값 |
| file\_type | 파일의 유형 (예: 실행 파일, 문서 파일 등) |
| magic | 파일의 매직 넘버 또는 식별자 |
| file\_size | 파일 크기 (바이트 단위) |
| PEID\_packer | 파일에 사용된 패커 정보 |
| first\_seen\_time | 파일이 처음 발견된 시간 |
| name | 파일의 이름 |

**signature** **:** 파일 서명 데이터를 저장, 파일이 디지털 서명되어 있거나 인증된 경우 그 정보를 저장

**pe\_info** : PE(Portable Executable) 파일에 대한 정보 PE 파일은 주로 Windows 운영 체제에서 실행되는 파일, PE 구조와 관련된 추가 정보

**dot\_net\_assembly** : .NET 어셈블리 파일에 대한 정보를 저장 .NET 파일이 실행되는 동안 사용되는 메타데이터 및 코드 모듈 정보

## behavior

**mitre** : MITRE ATT&CK 프레임워크에 기반한 공격 기법 분석 정보를 저장 파일의 악성 활동이 MITRE ATT&CK의 어떤 공격 기법에 해당하는지에 대한 정보

**Capabilities** : 파일이 실행될 때 수행할 수 있는 기능에 대한 정보 (예 : 파일이 시스템 권한 상승, 키로깅, 백도어 생성 등의 악성 행동을 수행할 수 있는지에 대한 데이터)

**tags** : 분석된 파일의 행동에 따라 붙여진 태그를 저장 (예 : "ransomware", "spyware")

**network\_communications** : 파일이 실행 중에 수행한 네트워크 통신에 대한 정보HTTP 대화, IP 주소, 도메인, JA3 지문 등의 데이터를 저장

|  |  |
| --- | --- |
| **필드** | **설명** |
| http\_conversations | 파일이 서버와 주고받은 HTTP 요청 및 응답 정보 |
| ja3\_digests | JA3 지문은 TLS 연결에서 클라이언트 측 정보를 해싱한 값, 특정 네트워크 패턴을 식별하는 데 사용 |
| memory\_pattern\_domains | 메모리에서 발견된 악성 도메인 |
| memory\_pattern\_ips | 메모리에서 발견된 IP 주소 |
| memory\_pattern\_urls | 메모리에서 발견된 URL 정보 |

**file\_system\_actions** : 파일이 시스템에서 실행되면서 수행한 파일 시스템 관련 동작

|  |  |
| --- | --- |
| **필드** | **설명** |
| files\_opened | 파일이 열린 기록 |
| files\_written | 파일이 작성된 기록 |
| files\_deleted | 파일이 삭제된 기록 |
| files\_attribute\_changed | 파일 속성(예: 읽기 전용)이 변경된 기록 |
| files\_dropped | 실행 도중 생성되거나 드롭된 파일에 대한 정보 |

**registry\_actions** : 파일이 레지스트리와 관련하여 수행한 동작

|  |  |
| --- | --- |
| **필드** | **설명** |
| registry\_keys\_opened | 파일이 접근한 레지스트리 키 |
| registry\_keys\_set | 파일이 설정한 레지스트리 키 |
| registry\_keys\_deleted | 파일이 삭제한 레지스트리 키 |

**process\_and\_service\_actions** : 프로세스 및 서비스 관련 동작

|  |  |
| --- | --- |
| **필드** | **설명** |
| processes\_created | 파일이 생성한 프로세스 정보 |
| command\_executions | 파일이 실행한 명령어 |
| processes\_injected | 파일이 다른 프로세스에 주입한 내용 |
| processes\_terminated | 파일이 종료한 프로세스 |
| services\_opened | 파일이 실행한 서비스 관련 정보 |
| processes\_tree | 파일이 생성한 프로세스 트리 구조 |

**synchronization\_mechanisms\_signals** : 파일이 시스템에서 동기화 메커니즘과 관련하여 수행한 동작

|  |  |
| --- | --- |
| **필드** | **설명** |
| mutexes\_created | 파일이 생성한 mutex 객체 |
| mutexes\_opened | 파일이 열린 mutex 객체 |

**modules\_loaded** : 파일이 실행 중에 로드한 모듈을 기록 악성 파일이 추가적으로 로드하는 라이브러리나 코드

**highlighted\_actions** : 분석 중에 강조된 주요 행동

|  |  |
| --- | --- |
| **필드** | **설명** |
| calls\_highlighted | 중요하거나 특이한 시스템 호출 |
| text\_decoded | 실행 중에 디코딩된 텍스트 |

**system\_property\_lookups** : 파일이 조회한 시스템 속성 정보(예 : 파일이 시스템 버전, 사용자 정보, 설치된 소프트웨어 정보를 확인하려는 시도 등)

MongoDB Protect file데이터베이스 구성

# 1. DB명: normal\_files

## 컬렉션: filedata

원본 파일의 메타데이터와 바이너리 파일 데이터

|  |  |
| --- | --- |
| **필드** | **설명** |
| \_id | MongoDB 고유 식별자 (자동 생성) |
| signature\_id | 파일의 고유 서명 ID (예: "20240909-001") |
| filename | 파일명 (예: "PEview.exe") |
| file\_extension | 파일 확장자 (예: ".exe") |
| file\_data | Base64로 인코딩된 바이너리 파일 데이터 |
| upload\_time | 파일 업로드 시간 (ISODate 형식) |
| upload\_ip | 파일을 업로드한 IP 주소 (예: "192.168.0.1") |

## 컬렉션: pe\_info

normal\_files에 저장된 원본 파일의 PE(Portable Executable) 정보

|  |  |
| --- | --- |
| **필드** | **설명** |
| \_id | MongoDB 고유 식별자 (자동 생성) |
| signature\_id | 파일의 고유 서명 ID (예: "20240909-001") |
| filename | 파일명 (예: "PEview.exe") |
| upload\_time | PE 정보가 저장된 시간 |
| upload\_ip | PE 정보를 업로드한 IP 주소 |
| pe\_info | PE 파일 분석 정보가 포함된 객체 |
| encrypted | 파일의 각 섹션이 암호화되었는지 여부가 포함된 객체 |

# 2. DB명: encrypted\_files

## 컬렉션: filedata

암호화된 파일의 메타데이터와 관련 정보 gridfs\_file\_id는 GridFS(대용량 파일 저장 시스템)에서 암호화된 파일이 저장된 위치

|  |  |
| --- | --- |
| **필드** | **설명** |
| \_id | MongoDB 고유 식별자 (자동 생성) |
| signature\_id | 파일의 고유 서명 ID (예: "20240909-001") |
| original\_filename | 암호화되기 전의 원본 파일 이름 (예: "PEview.exe") |
| encrypted\_filename | 암호화된 후의 파일 이름 (예: "20240909-001\_protected.exe") |
| original\_upload\_time | 원본 파일의 업로드 시간 |
| encrypted\_upload\_time | 암호화된 파일의 업로드 시간 |
| upload\_ip | 파일을 업로드한 IP 주소 |
| gridfs\_file\_id | GridFS에 저장된 파일의 ID |

## 컬렉션: pe\_info

암호화된 파일의 PE 정보

|  |  |
| --- | --- |
| **필드** | **설명** |
| \_id | MongoDB 고유 식별자 (자동 생성) |
| signature\_id | 파일의 고유 서명 ID (예: "20240909-002") |
| encrypted\_filename | 암호화된 파일의 파일 경로 (예: "20240909-002\_protected.exe") |
| encrypted\_upload\_time | 암호화된 파일의 업로드 시간 |
| upload\_ip | 파일을 업로드한 IP 주소 |
| pe\_info | 암호화된 파일의 PE 분석 정보가 포함된 객체 |
| encrypted | 암호화된 섹션 정보가 포함된 객체 |

1. 데이터셋 가공 모듈

Malware Bazzar에서 가져온 데이터셋을 가공하기 위한 가공 모듈입니다. Virustotal API가 제한이 있기 때문에 Windows task schedular를 통해서 Main module을 매달 10,11,12일 250번 실행될 수 있게 자동화 설정을 해두었습니다. 모듈은 Python3로 작성되었습니다.

Main module

1. # 블랙리스트에 들어가 있는 해시 값을 불러오는 모듈

2. processed\_hashes = load\_processed\_hashes()

3.

4. # 블랙리스트에 추가될 해시 값을 불러오는 모듈

5. execution\_count = 0

6. hashes = read\_hashes\_from\_csv()

7.

8. for hash\_value in hashes:

9. if execution\_count >= MAX\_EXECUTIONS\_PER\_DAY: # API KEY 하루 사용량 제한

10. logger.info("오늘의 최대 실행 횟수에 도달했습니다.")

11. break

12. # Virustotal API를 통해 가공되는 모듈

13. if process\_hash(hash\_value, processed\_hashes):

14. execution\_count += 1 # 실제로 처리된 경우에만 증가

process\_hash module

1. def process\_hash(hash\_value, processed\_hashes):

2. if hash\_value in processed\_hashes:

3. logger.info(f"{hash\_value} 이미 처리됨. 스킵합니다.")

4. return False # 이미 처리된 경우 False 반환

5.

6. # 유효한 MD5 해시인지 확인

7. if not is\_valid\_md5(hash\_value):

8. logger.warning(f"{hash\_value} 유효하지 않은 MD5 해시입니다. 스킵합니다.")

9. return False # 유효하지 않은 해시는 처리하지 않음

10.

11. # VirusTotal API 호출

12. details = search\_file\_by\_hash(hash\_value)

13. if details is None:

14. logger.error(f"{hash\_value} 처리 실패. 파일을 찾을 수 없습니다.")

15. return False

16.

17. behavior = search\_file\_by\_hash(hash\_value, "behaviour\_summary")

18. if behavior is None:

19. logger.warning(f"{hash\_value} 행동 분석 정보를 찾을 수 없습니다.")

20.

21. if details:

22. # 데이터 변환

23. logger.info(f"{hash\_value} 데이터 변환 중...")

24. converted\_data = convert\_data(details, behavior)

25.

26. # MongoDB에 업로드

27. upload\_to\_mongodb(converted\_data, "info")

28. logger.info(f"{hash\_value} MongoDB에 저장 완료.")

29.

30. # 처리된 해시 기록

31. processed\_hashes.append(hash\_value)

32. save\_processed\_hashes(processed\_hashes)

33. logger.info(f"{hash\_value} 처리된 해시 기록에 추가.")

34. return True # 성공적으로 처리된 경우 True 반환

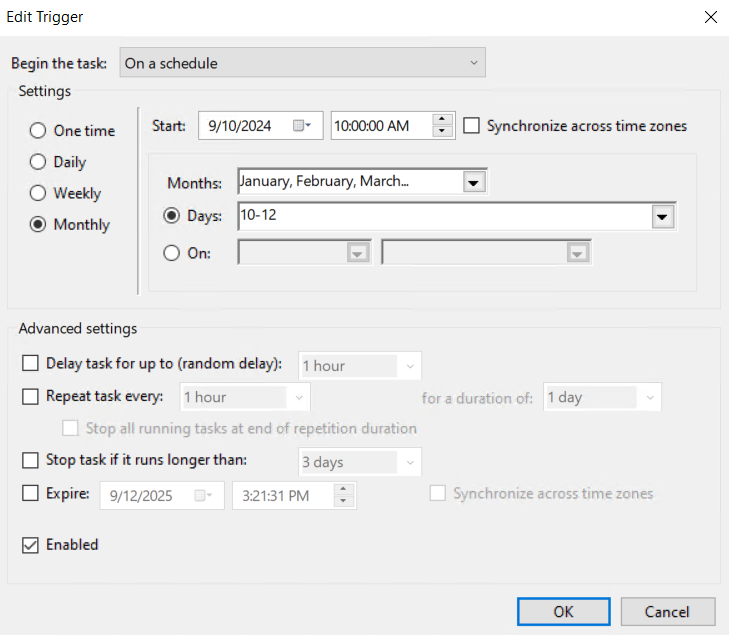
35. else:

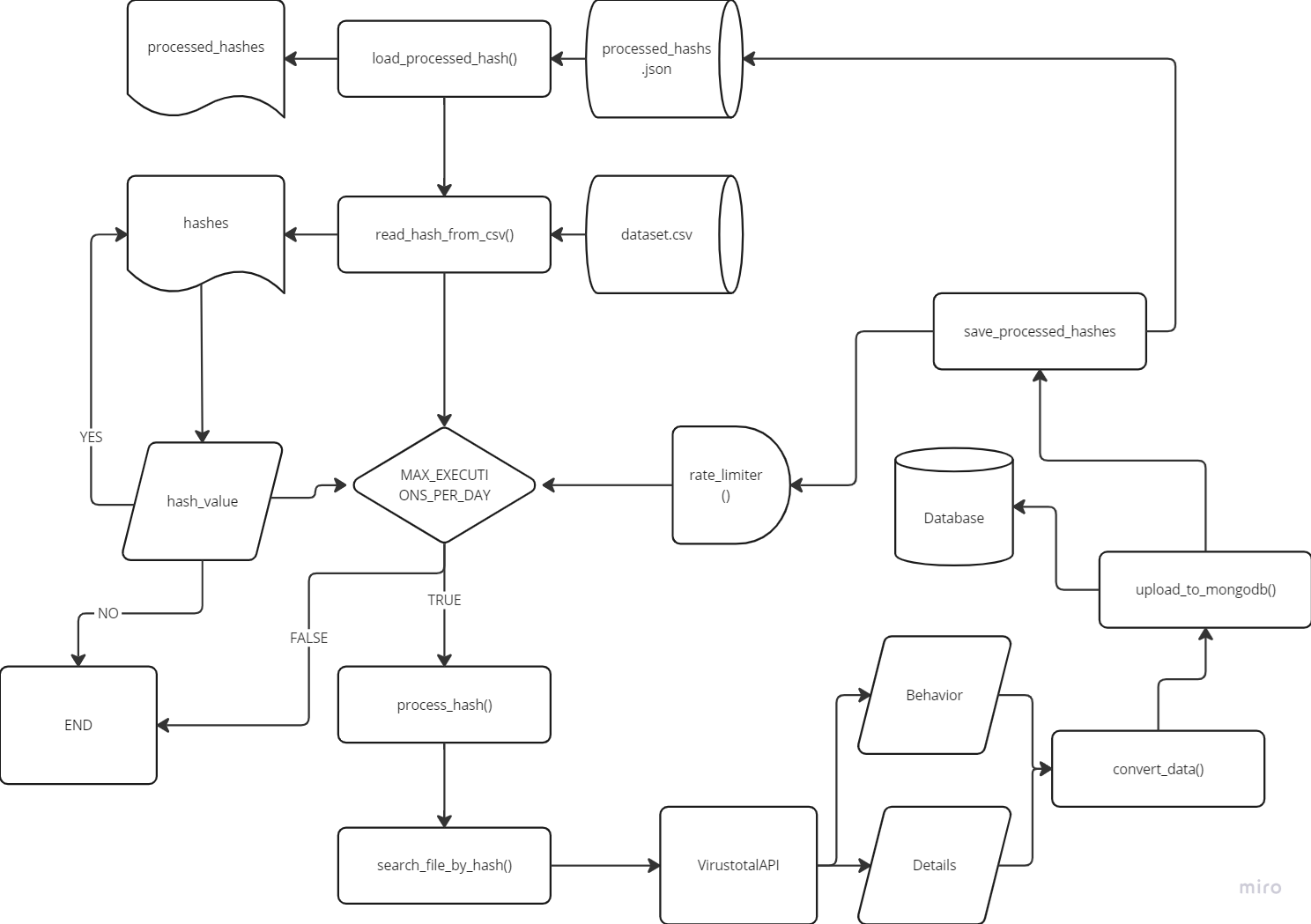
36. logger.error(f"{hash\_value} 처리 실패.")

37. return False # 처리 실패 시 False 반환

38.

39. rate\_limiter()



모듈 동작 구성도

1. 결론

저희는 Malware Bazzar에서 가져온 데이터셋과 Virustotal API를 통해 나온 데이터의 정보 불균형을 해결하기 위해 Virustotal API File hash search 기능을 사용하여 데이터를 재가공해야합니다.

현재 일반 사용자에게 제공된 Virustotal API KEY를 통해 Malware Bazzar에서 가져온 데이터셋을 가공하고 있습니다. 하지만 일반 사용자에게 제공된 API KEY는 API 쿼리를 보낼 수 있는 제한이 상당히 적기 때문에 80만개가량의 데이터를 전부 가공하기에는 시간이 걸립니다. 그래서 저희는 windows task schedular를 사용하여 가공하는 작업을 자동화했습니다. 또한 Virustotal 팀에 연구 목적으로 연구용 API KEY를 메일로 요청한 상태입니다. 연구용 API KEY를 받으면 최종적으로 블랙리스트를 가공하는 기간이 상당히 줄 것으로 예상됩니다.