Pre-Vision

프로젝트

보고서

악성코드 분석, 암호화 및 패키징

Malware Detection, Encryption, and Packaging

10월 8일

Pre-Vision

작성자: 우건희

목차

1. 프로젝트 개요
2. 팀원 구성 및 역할
3. 협업 도구
4. 프로젝트 일정
5. FLOW CHART
6. 요구사항 정의
7. 서버 구성
8. 데이터셋 수집 및 가공
9. 데이터베이스 정의
10. 서브보고서
11. 기능 구현
12. 모듈 별 동작도
13. 테스트
14. 결론
15. 프로젝트 과정 중 어려움 및 느낀점
16. 참고자료

프로젝트 개요

팀원 구성 및 역할

우건희

직책 : 팀장(모듈 담당)

역할

1. Protect 모듈 개발
2. Virustotal API 모듈 개발
3. 프로젝트 일정 관리
4. 프로젝트 문서 관리
5. 클라우드 구축
6. 데이터 재가공

김선우

직책 : 팀원(웹 담당)

역할

1. 프론트엔드 개발
2. 백엔드 개발
3. 데이터베이스 서버 연동
4. 프로젝트 문서 관리
5. 테스트 검토 및 검증

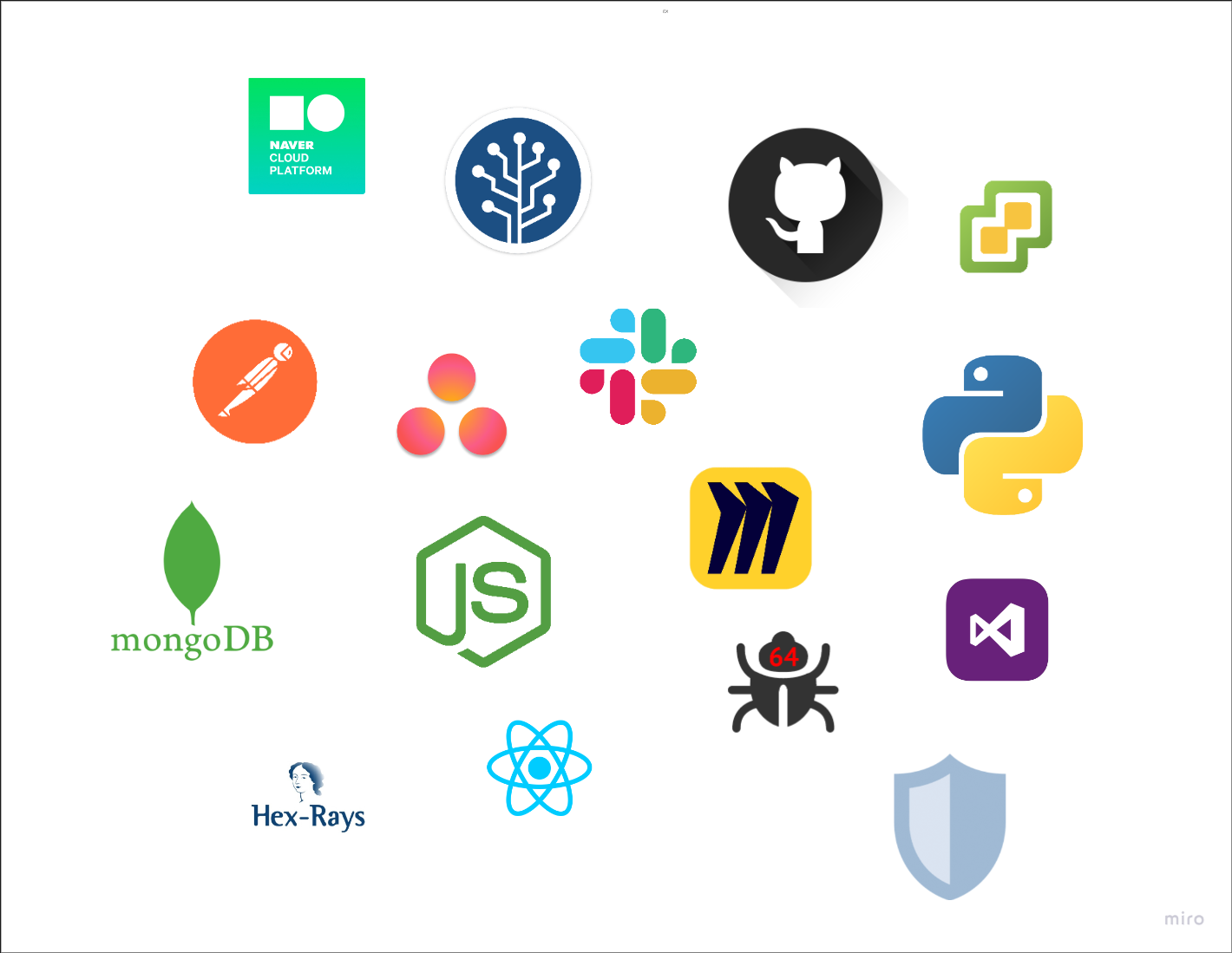
김효진

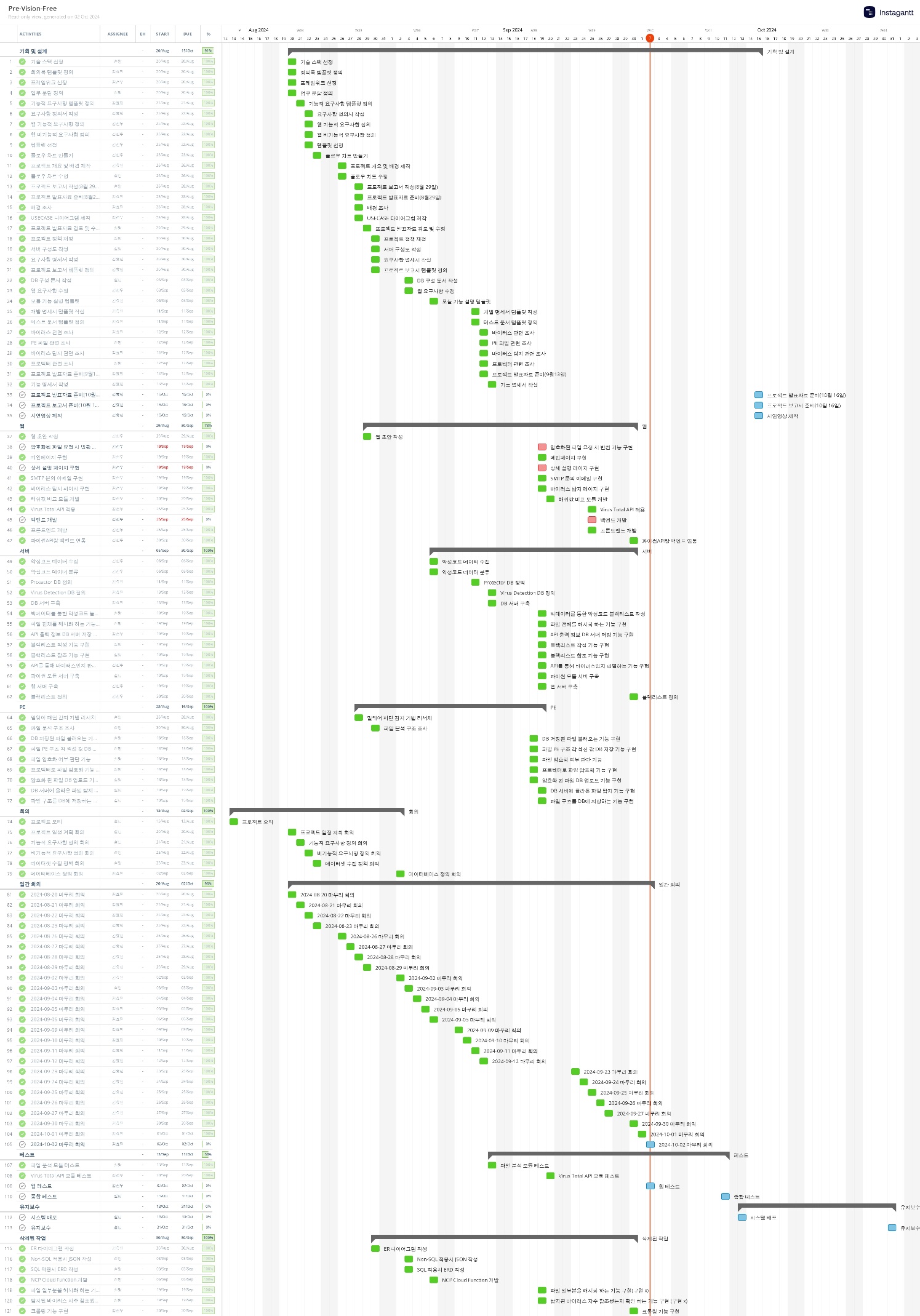
직책 : 팀원(프로젝트 관리 담당)

역할

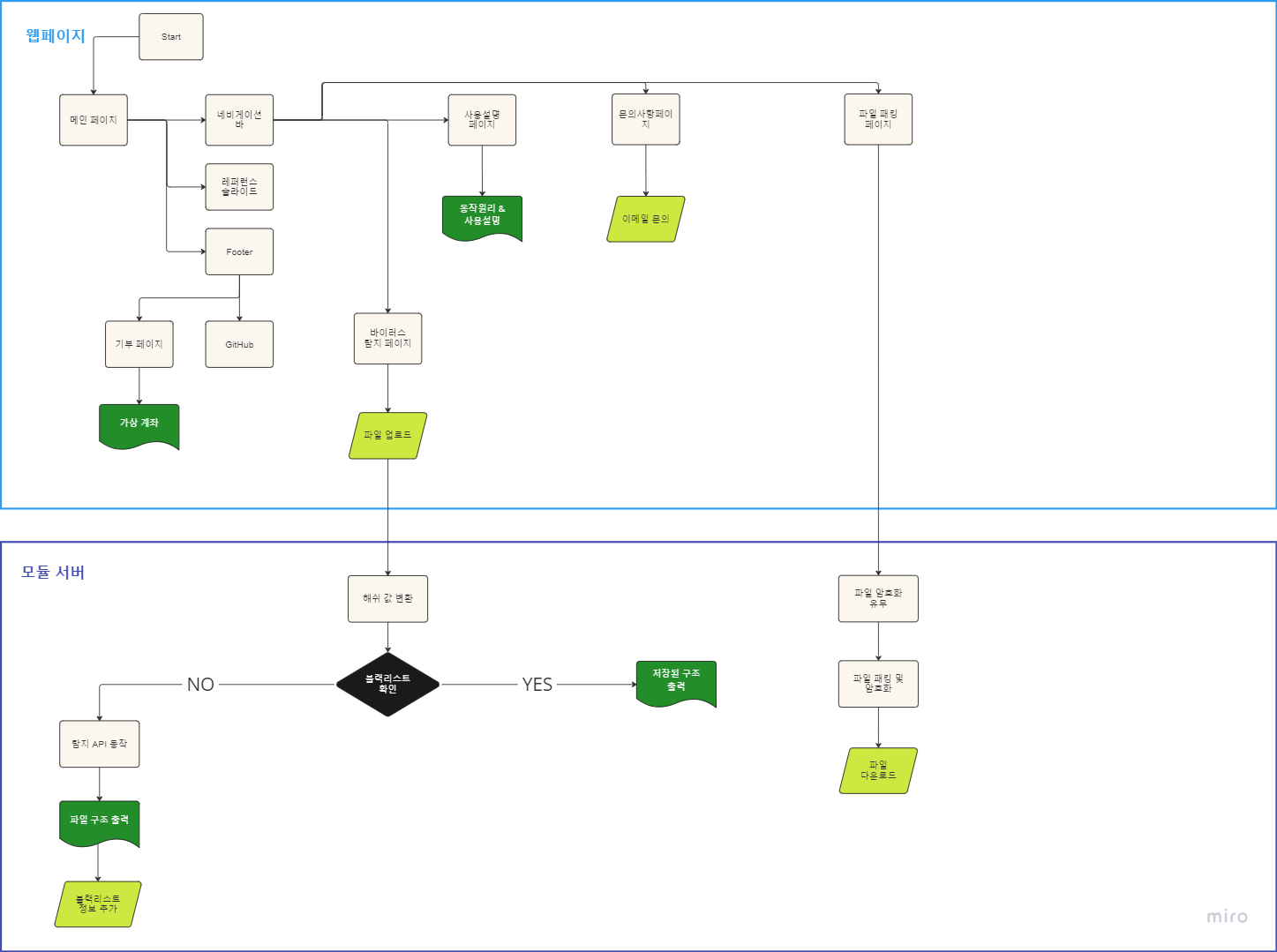
1. 프로젝트 일정 관리
2. 프로젝트 문서 관리
3. 문서 템플릿 정의
4. QA
5. 악성코드 데이터 수집
6. 데이터베이스 서버 구축

협업 도구



프로젝트 일정

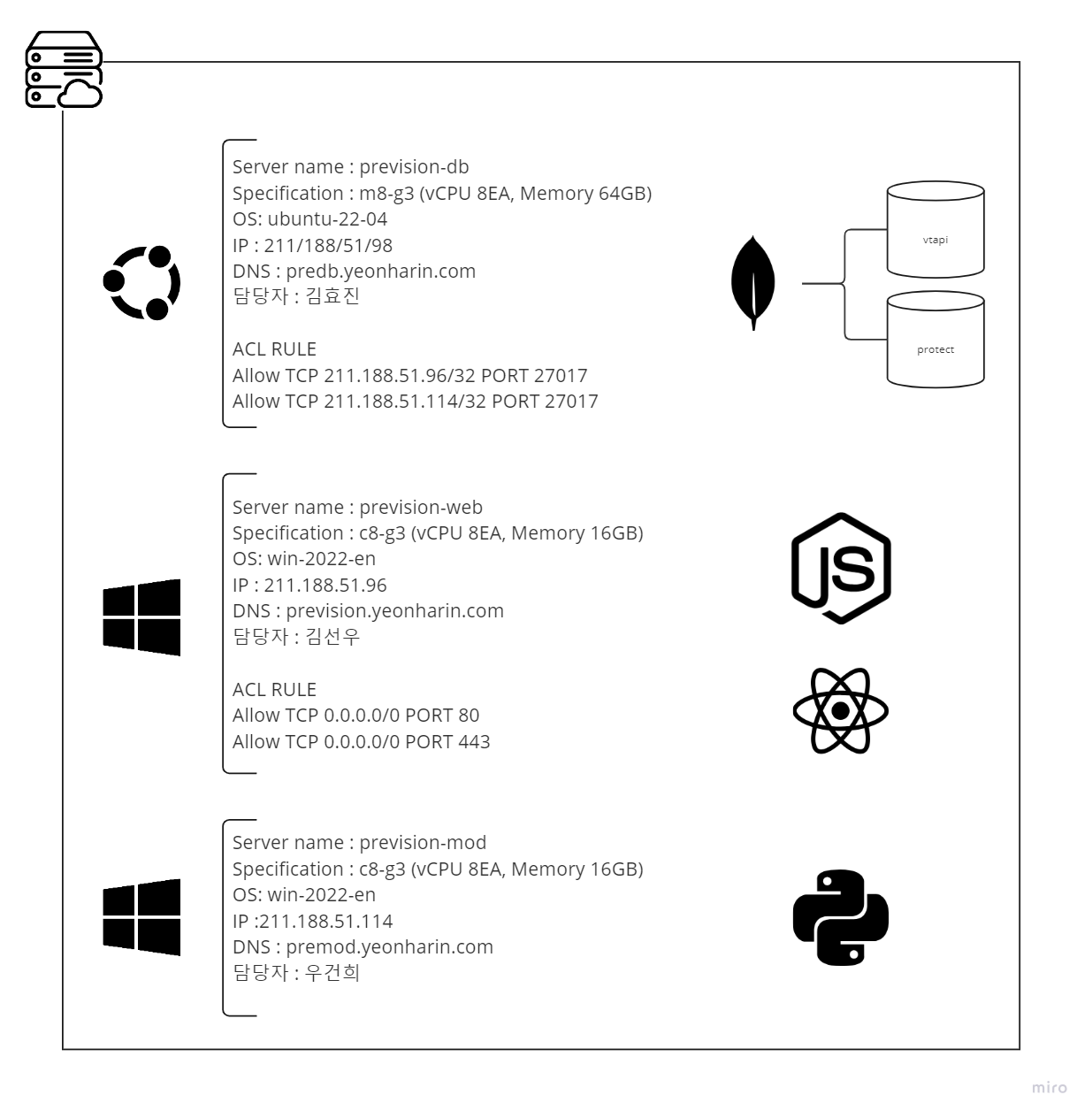
FLOW CHART



요구사항 정의

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 번호 | 업무 영역 | 요구사항 ID | 요구사항 명 | 비고 |
| 1 | 신규기능 선정 | FR-101 | 메인 로고 클릭 시 메인페이지 Redirect기능 |  |
| 2 | FR-102 | 기부 페이지 Redirect기능 |  |
| 3 | FR-103 | 사용설명 페이지 Redirect기능 |  |
| 4 | FR-104 | 문의 페이지 Redirect기능 |  |
| 5 | FR-104-01 | 문의메일 전송 기능 |  |
| 6 | FR-105 | 개발팀Github페이지 Redirect기능 |  |
| 7 | FR-106 | 바이러스탐지 페이지 Redirect기능 |  |
| 8 | FR-106-01 | 바이러스 탐지하고 구조출력기능 |  |
| 9 | FR-107 | 레퍼런스 페이지 Redirect 기능 |  |
| 10 | FR-108 | 패킹 페이지 Redirect 기능 |  |
| 11 | FR-108-01 | 파일 다운로드 기능 |  |
| 12 | FR-201 | DB 저장된 파일 불러오는 기능 |  |
| 13 | FR-202 | 파일 PE 구조 각 섹션 값 DB 저장 기능 |  |
| 14 | FR-203 | 파일 암호화 여부 판단 기능 |  |
| 15 | FR-204 | 프로텍터를 통한 파일 암호화 기능 |  |
| 16 | FR-205 | 암호화 된 파일 DB 업로드 기능 |  |
| 17 | FR-301 | 빅데이터를 통한 악성코드 블랙리스트 작성 |  |
| 18 | FR-302 | 암호화된 파일 요청 시 반환 기능 | 웹에서 DB로 접근 |
| 19 | FR-303 | 파일 전체를 해시화 하는 기능 | 웹 구현 |
| 20 | FR-304 | 파일 일부분을 해시화 하는 기능 | 기능 구현 X |
| 21 | FR-305 | API 출력 정보 DB 서버 저장 기능 |  |
| 22 | FR-306 | 탐지된 바이러스 자주 참조됐는지 확인하는 기능 | 기능 구현 X |
| 23 | FR-307 | 블랙리스트 작성 기능 | FR-301과 통합 |
| 24 | FR-308 | 블랙리스트 참조 기능 |  |
| 25 | FR-309 | API를 통해 바이러스인지 판별하는 기능 | 기능 구현 X |
| 26 | FR-310 | 파일 구조를 DB에 저장하는 기능 | FR-202와 통합 |
| 27 | FR-311 | DB 서버에 올라온 파일 탐지 기능 |  |
| 28 | FR-312 | 파이썬 모듈 서버 구축 |  |
| 29 | FR-313 | DB 서버 구축 |  |
| 30 | FR-314 | 웹 서버 구축 |  |

서버 구성



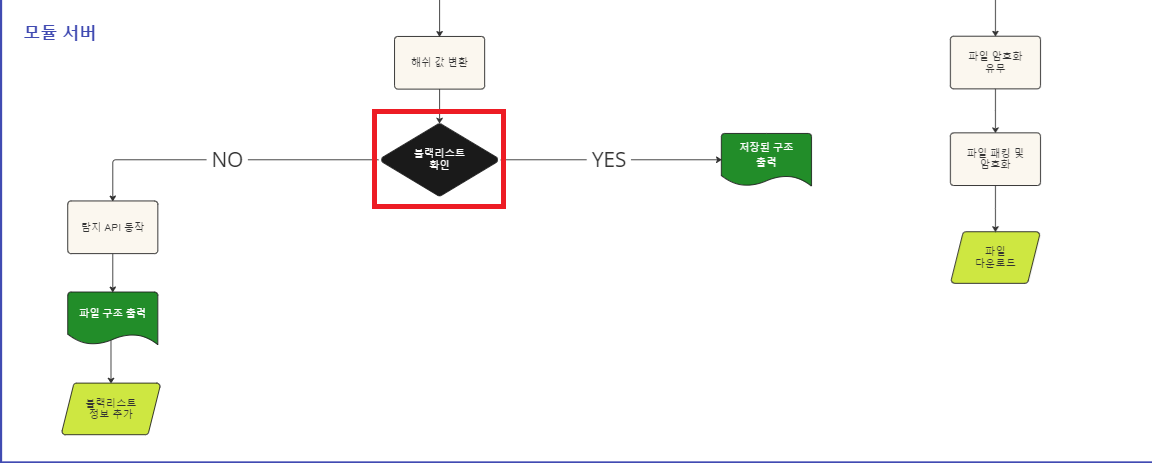
데이터셋 수집 및 가공

악성코드 분석, 암호화 및 패키징 프로젝트

데이터셋 수집 보고서

1. 데이터 수집 정책

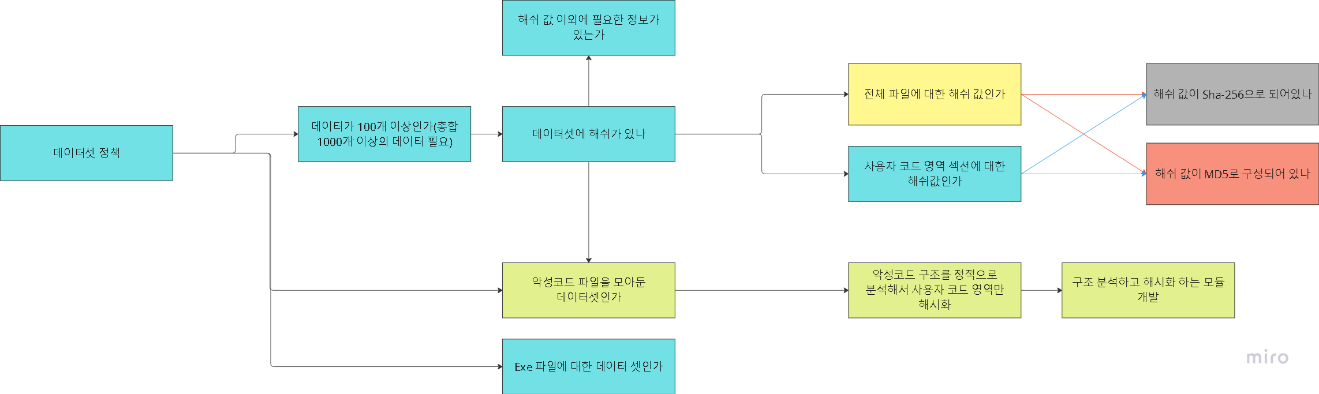
먼저 플로우 차트를 살펴 보면 저희가 만들어야 하는 데이터셋은 블랙리스트에 해당 합니다.



블랙리스트는 사전에 정의된 바이러스 정보가 들어가야 합니다. 바이러스 정보에는 파일의 해시, 파일 타입 등의 정보, 시그니처 정보, pe 정보 등이 포함되어야 합니다. 또한 Virus Total 사이트에서 파일을 업로드 하거나 해시를 통해 검색하면 나오는 정보로 Behavior 정보도 있습니다. 이 또한 API를 통해 받아올 수 있는 정보입니다. 그래서 블랙리스트에는 최종적으로 Detail, Behavior 정보가 포함됩니다.

블랙리스트를 만들기 위해선 사전 정의된 바이러스 정보가 있는 데이터셋을 사용해야 합니다. 가장 좋은 방안으로는 Virus Total에 정의되어 있는 정보를 가져오는 것이지만 서비스에서 따로 제공해주는 데이터가 없었습니다. 그래서 저희는 다른 사이트를 서치하여 데이터셋을 수집하기로 했습니다.

데이터셋 정책



데이터셋을 수집할 때 최종적으로 고려해야 할 사항입니다.

1. 데이터가 100개 이상인가?
2. 데이터셋에 해시 값이 포함되어 있나?
3. 전체 파일에 대한 해시 값인가?
4. MD5 해시가 있는가?

저희가 만들 블랙 리스트는 서비스의 시간 단축이 목적인 데이터 셋입니다. 따라서 사전에 정의된 데이터들이 많으면 좋습니다. 따라서 위에 4개의 정책을 중점으로 데이터셋을 수집했습니다.

1. 선정된 데이터

최종적으로 선정된 데이터셋은 Malware bazzar abuse에서 가져온 csv file입니다. 이 파일은 2020년 2월 13일부터 2024년 8월 29일까지 이 사이트에 보고된 바이러스 파일들에 대한 정보가 포함되어 있습니다.

데이터셋 출처 : <https://bazaar.abuse.ch/export/>

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 |

이 데이터셋에는 이러한 정보가 담긴 데이터가 약 80만개가량 있습니다. 해당 데이터셋을 가공하여 블랙리스트의 정보를 만드는데 적합하다고 판단했습니다. 최종적으로 블랙리스트에 적용되어야 할 정보는 Virustotal 사이트를 통해서 나온 detail과 behavior 섹션의 값이므로 Virustotal API를 적용시켜 가공할 수 있는 데이터셋으로 MD5 hash를 선정했습니다. 이 데이터셋을 API module을 통해 가공하여 블랙리스트를 정의할 것입니다.

1. 선정되지 않은 데이터

루트킷 및 백도어 악성코드 데이터셋

https://www.bigdata-telecom.kr/invoke/SOKBP2603/?goodsCode=KIS0000035

선정되지 않은 이유 : 데이터셋 수집 정책에 따라서 데이터셋을 분석한 결과 블랙리스트를 구축하기에 필요한 필수 정보들(해시 값 등)이 부족했습니다.

Microsoft Malware Classification Challenge (BIG 2015)

https://www.kaggle.com/c/malware-classification

선정되지 않은 이유 : 데이터셋의 데이터가 많았지만 블랙리스트를 구축하기에 필요한 필수 정보들(해시 값, 파일 정보 등)이 부족했습니다.

한국인터넷진흥원 - 정상 및 악성코드 데이터셋

https://www.gimi9.com/dataset/bdp-kt-co-kr-dataset-1367055

선정되지 않은 이유 : 해시 값은 있었지만 정상 파일도 포함되어 있어 블랙리스트를 구성하는데 분류 과정이 늘어나기 때문에 선정에서 제외했습니다.

대용량 정상/악성파일 Ⅱ(test set)

https://www.ksecurity.or.kr/kisis/subIndex/375.do

선정되지 않은 이유 : 데이터셋의 데이터가 많았지만 블랙리스트를 구축하기에 필요한 필수 정보들(해시 값, 파일 정보 등)이 부족했습니다.

대용량 정상, 악성파일 5

https://www.ksecurity.or.kr/kisis/subIndex/493.do

선정되지 않은 이유 : 데이터셋의 데이터가 많았지만 블랙리스트를 구축하기에 필요한 필수 정보들(해시 값, 파일 정보 등)이 부족했습니다.

SoReL-20M

https://github.com/sophos/SOREL-20M

선정되지 않은 이유 : Amazon S3를 통해 구성해야 하는 데이터셋으로 NCP를 사용하는 것이 좋기 때문에 선정에서 제외됐습니다. 또한 데이터셋의 용량이 8TB이기 때문에 저희에게 할당된 서비스 요금만으로는 처리가 힘들 것 같아 제외됐습니다.

VirusShare info

https://virusshare.com/

선정되지 않은 이유 : 수집하는 데이터의 문제는 없었으나 홈페이지 이용 및 다운로드의 어려움 때문에 데이터셋을 구할 수 없었습니다.

1. 결론

블랙리스트와 Virus total API를 돌려서 나온 정보가 불균형을 이루면 안되기 때문에 Virustotal API로 데이터를 가공해야 합니다. 따라서 수집해야 할 데이터셋에는 해시 값이 필수로 포함되어 있어야합니다. 해당 해시 값으로 Virustotal API module을 작동시켜 나온 데이터를 데이터베이스에 저장하여 블랙리스트 데이터베이스를 구성해야 합니다.

저희는 최종적으로 Malware Bazzar abuse 의 악성코드 데이터셋을 선정했고 해당 데이터는 약 80만개의 데이터를 포함하고 있습니다. 데이터셋에는 md5, sha-256 등의 중요한 해시정보가 포함되어 있고 저희는 md5 hash 값을 이용해 데이터를 재가공 할 예정입니다.

악성코드의 분석 데이터는 정형화 되어있지 않는 경우가 많기 때문에 데이터베이스 선정에 있어서 No-SQL 방식의 데이터베이스를 선정해야 했습니다. 저희는 BSON 형식으로 데이터를 저장하는 MongoDB를 선정했고 해당 DB에 블랙리스트를 구축할 것입니다.

악성코드 분석, 암호화 및 패키징 프로젝트

데이터셋 가공 보고서

1. 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

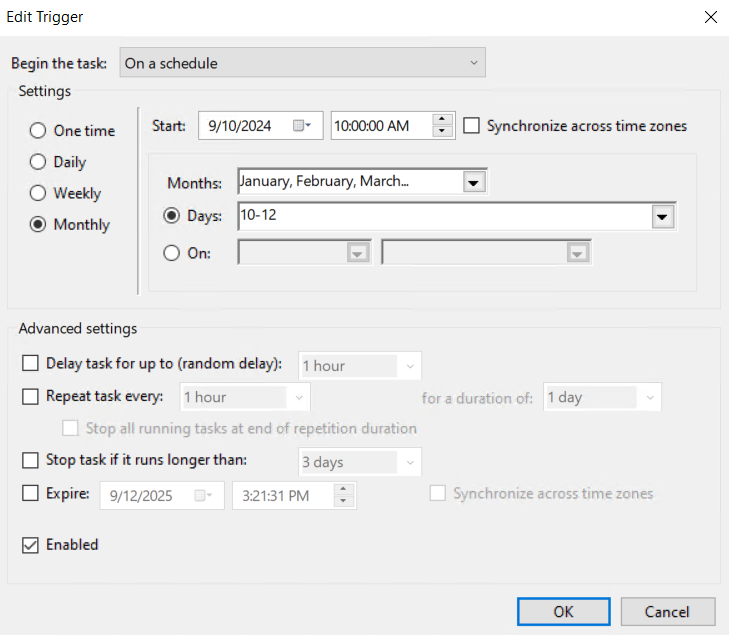
35. else:

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

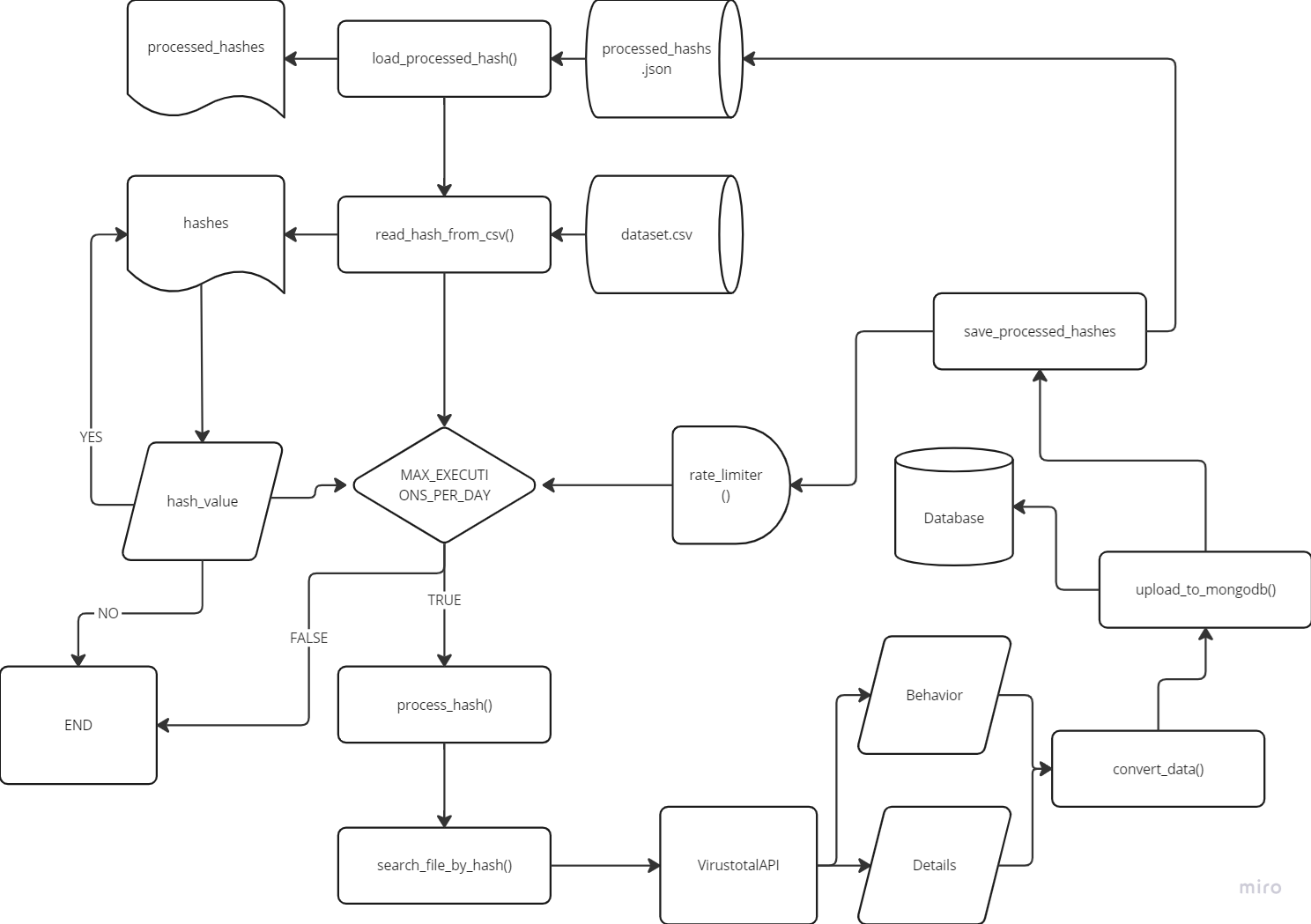
37. return False # 처리 실패 시 False 반환

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.

데이터베이스 정의

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 |

서브 보고서

악성코드 분석, 암호화 및 패키징 프로젝트

컴퓨터 바이러스 탐지 보고서

1. 바이러스 탐지 개요

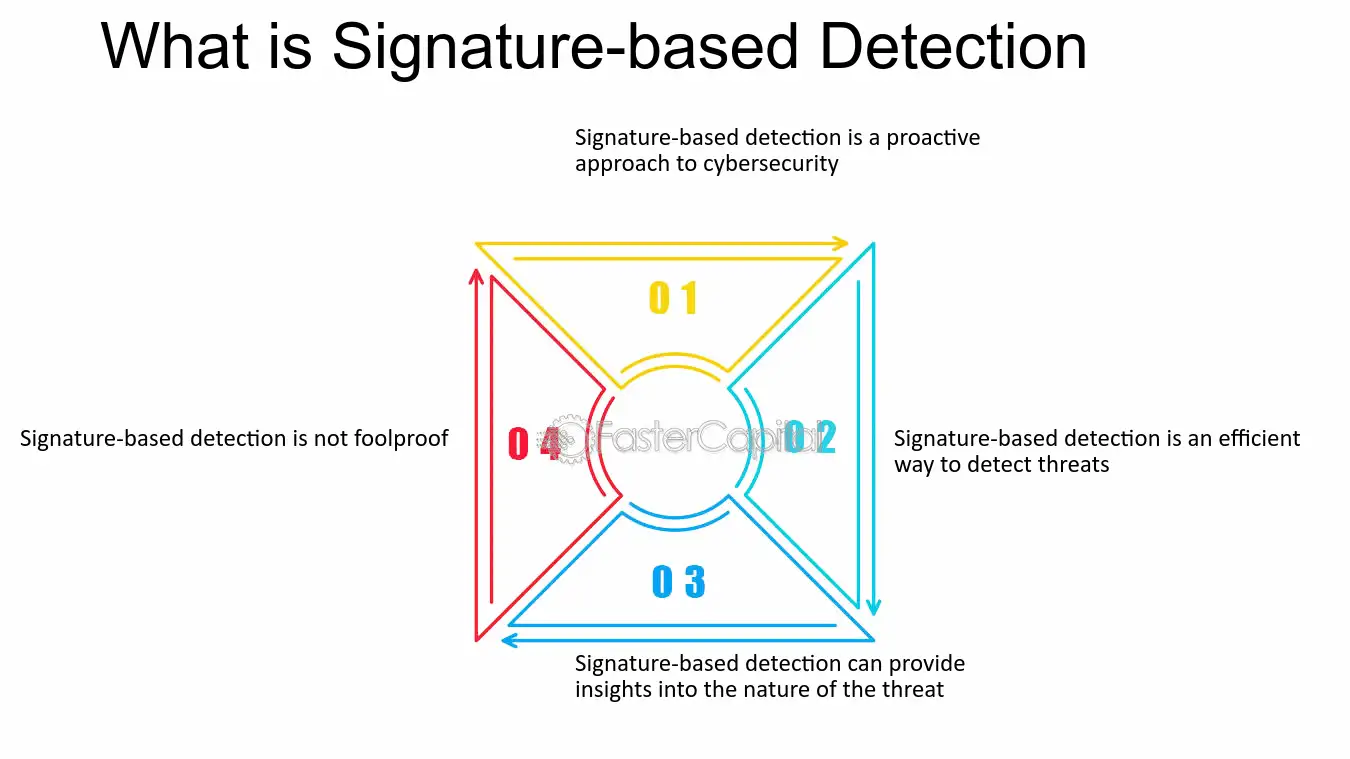
컴퓨터 바이러스는 악의적인 의도로 만들어진 프로그램으로, 사용자도 모르는 사이에 컴퓨터에 침투해 여러 가지 피해를 줍니다. 이런 바이러스는 주로 인터넷이나 USB 같은 저장 장치를 통해 퍼지며, 한번 컴퓨터에 들어가면 자기 복제를 통해 더 많은 악성 프로그램을 퍼뜨립니다. 이로 인해 개인이나 회사는 중요한 정보를 잃거나, 시스템이 멈추거나, 개인 정보가 유출되는 등 심각한 문제를 겪을 수 있습니다. 특히 랜섬웨어 같은 경우는 데이터를 인질로 잡고 돈을 요구하기도 합니다.

이런 위험으로부터 컴퓨터를 지키기 위해서는 바이러스를 찾아내는 것이 매우 중요합니다. 안티바이러스 프로그램은 컴퓨터를 계속 지켜보면서 알려진 바이러스뿐만 아니라 새로운 바이러스도 찾아내고 막아냅니다. 또한 주기적으로 컴퓨터를 검사해서 숨어있던 바이러스도 찾아내 격리하거나 삭제할 수 있습니다. 이렇게 바이러스를 찾아내는 기술은 계속 발전하고 있어서, 바이러스에 감염된 파일을 치료하거나 격리해서 컴퓨터를 안전하게 지키는 데 큰 도움을 주고 있습니다.

이 보고서에서는 VirusTotal이라는 도구에 대해서도 다룹니다. VirusTotal은 여러 안티바이러스 프로그램과 온라인 도구를 한데 모아 바이러스를 빠르게 찾아내는 플랫폼입니다. 의심스러운 파일이나 웹사이트 주소를 분석하고, 이를 알려진 위험 목록과 비교해서 자세한 보고서를 만들어 줍니다. 더 나아가 각 기술의 특징을 살펴보고 바이러스에 대응하는 종합적인 방법을 제안함으로써, 현재 바이러스 탐지 기술의 수준과 앞으로의 발전 방향을 가늠해볼 수 있을 것입니다.

1. 바이러스 탐지 기술
2. 시그니처 기반 탐지

시그니처 기반 탐지 기술은 이미 알려진 악성코드의 고유한 특성을 데이터베이스에 저장해 두고, 새로운 파일이나 프로세스와 비교하여 악성코드인지 판단하는 방식입니다. 보안 전문가들이 악성코드를 분석해 해당 코드의 고유한 패턴, 코드 시퀀스, 파일 해시 등의 시그니처를 추출하여 데이터베이스에 등록해 두면, 사용자는 이를 주기적으로 업데이트해야 합니다.



안티바이러스 프로그램이 시스템을 정기적으로 스캔할 때, 파일이나 실행 중인 프로세스를 데이터베이스와 대조하여 일치하는 시그니처가 있으면 악성코드로 탐지합니다. 이 경우 사용자에게 경고가 뜨거나 파일이 자동으로 격리 또는 삭제됩니다.

이 방법의 장점은 이미 알려진 악성코드를 정확하게 탐지할 수 있다는 점이며, 비교적 간단한 연산으로 빠르게 처리된다는 것입니다. 하지만 새로운 변종이나 데이터베이스에 없는 악성코드는 탐지할 수 없다는 한계가 있습니다. 또한 너무 일반적인 시그니처를 사용하면 정상적인 파일을 잘못 탐지할 가능성도 있습니다.

1. 휴리스틱 기반 탐지

휴리스틱 기반 탐지 기술은 악성코드의 특정 패턴이 아닌, 프로그램의 실행 행동과 패턴을 분석하여 잠재적인 악성 활동을 찾아내는 방식입니다. 예를 들어, 프로그램이 시스템 자원에 접근하거나 레지스트리 키를 변경하는 등의 행동을 모니터링하여 미리 정의된 규칙에 맞는지 비교하는 방식입니다.

이 방식은 시그니처 기반 탐지와 달리 새롭게 등장한 악성코드도 탐지할 수 있다는 장점이 있습니다. 규칙을 계속 업데이트함으로써 탐지 범위도 넓힐 수 있죠. 하지만 규칙이 너무 넓으면 정상적인 프로그램까지 악성코드로 잘못 판단할 수 있고, 복잡한 분석이 필요하기 때문에 시스템 자원을 많이 사용할 수 있습니다.

1. 행위 기반 탐지

행위 기반 탐지 기술은 프로그램이 실행되는 동안 지속적으로 행동을 모니터링하여 악성코드를 찾아내는 방식입니다. 예를 들어, 레지스트리 키 변경, 민감한 데이터 유출 시도 등의 행위가 발생하면 악성코드로 의심하여 차단하는 식이죠.

이 방법의 장점은 새로운 변종 악성코드까지 탐지할 수 있다는 점입니다. 하지만 정상 프로그램이 악성 행위와 비슷한 행동을 할 경우, 잘못 탐지할 가능성이 있습니다. 또한 프로그램의 행동을 실시간으로 모니터링하기 때문에 시스템 자원을 많이 소모할 수 있습니다.

1. 새로운 탐지 기술
2. *머신러닝 기반 탐지*

머신러닝을 활용한 탐지 기술은 방대한 양의 데이터를 학습한 모델을 통해 프로그램의 행동 패턴을 분석해 악성코드를 예측하는 방식입니다. 덕분에 기존 시그니처 기반 탐지보다 더 넓은 범위의 위협을 탐지할 수 있습니다.

1. 클라우드 기반 탐지

클라우드 기반 탐지 기술은 사용자의 시스템에서 수집한 데이터를 클라우드 서버로 전송해 분석하는 방식입니다. 강력한 컴퓨팅 자원을 활용하여 대규모 데이터를 실시간으로 처리하며, 신속한 업데이트가 가능하다는 장점이 있습니다.

1. 인공지능 기반 탐지

인공지능을 이용한 탐지 기술은 머신러닝 모델을 통해 프로그램의 실행 행동 패턴을 분석합니다. 변종 악성코드까지 효과적으로 대응할 수 있으며, 오탐지율을 줄여 시스템 자원 낭비를 최소화할 수 있습니다.

1. 바이러스 토탈 API
2. 소개

VirusTotal은 여러 안티바이러스 엔진과 도구를 결합하여 파일, URL, IP 주소, 도메인을 분석해 악성코드를 탐지하는 온라인 서비스입니다. 이 서비스는 보안 전문가, 연구원, 그리고 위협을 식별하고 평가해야 하는 조직들에게 중요한 리소스 입니다.

VirusTotal API는 이 서비스를 확장하여, 사용자가 VirusTotal의 기능을 자체 소프트웨어에 통합할 수 있게 해줍니다.

1. 바이러스 토탈 작동방식

|  |  |
| --- | --- |
|  | VirusTotal은 파일이나 URL을 여러 안티바이러스 엔진에 제출하여 악성코드를 검사합니다. 이 플랫폼은 Kaspersky, McAfee, Symantec과 같은 주요 보안 업체를 포함해 70개 이상의 보안 벤더와 협력하여 제출된 콘텐츠의 안전성을 평가합니다. 그 결과는 통합되어 사용자에게 제공되며, 악성코드로 감지된 파일과 위협 유형에 대한 자세한 보고서가 포함됩니다. |

1. 바이러스 토탈 API 쿼리문

바이러스 토탈 API에는 endpoint를 통한 웹 액션을 쿼리문으로 사용합니다. 주요 쿼리문은 아래와 같습니다.

1. 파일 업로드 및 분석 요청

Endpoint: /file

기능: 사용자가 파일을 업로드하여 멀티 엔진 분석을 요청합니다.

쿼리 예시: POST https://www.virustotal.com/api/v3/files

1. 해시 또는 파일에 대한 분석 결과 조회

Endpoint: /file/id

리소스 ID를 사용하여 파일의 분석 결과를 조회합니다.

쿼리 예시: GET https://www.virustotal.com/api/v3/files/id

1. URL 스캔 및 분석 요청

Endpoint: /url

기능: 특정 URL에 대한 멀티 엔진 분석을 요청합니다.

쿼리 예시: POST https://www.virustotal.com/api/v3/urls

1. URL 분석 결과 조회

Endpoint: /urls/id

기능: URL에 대한 분석 결과를 확인합니다.

쿼리 예시: GET https://www.virustotal.com/api/v3/urls/id

1. IP 주소 정보 조회

Endpoint: /ip\_addresses/ip

기능: 특정 IP 주소와 관련된 보안 정보를 조회합니다.

쿼리 예시: GET https://www.virustotal.com/api/v3/ip\_addresses/ip

1. 도메인 정보 조회

Endpoint: domains/domain

기능: 특정 도메인과 관련된 보안 정보를 조회합니다.

쿼리 예시: GET https://www.virustotal.com/api/v3/domains/domain

1. 파일 재분석 요청

Endpoint: /files/id/analyse

기능: 이전에 분석된 파일을 다시 분석합니다.

쿼리 예시: POST https://www.virustotal.com/api/v3/files/id/analyse

1. 댓글 추가

Endpoint: / files/id/comments

기능: 특정 파일에 대해 의견을 추가합니다.

쿼리 예시: POST https://www.virustotal.com/api/v3/files/id/comments

1. 대용량 파일에 대한 upload 주소 얻기

Endpoint: /files/upload\_url

기능: 크기가 큰 파일을 업로드 하기 위한 url주소를 얻습니다.

쿼리 예시 : GET https://www.virustotal.com/api/v3/files/upload\_url

아래 이미지는 API를 통해 파일을 스캔하는 과정과 보고서로 나오는 구조 입니다.

|  |  |
| --- | --- |
|  |  |

1. 결론

컴퓨터 바이러스 및 악성코드의 위협은 날로 증가하고 있으며, 이에 대응하기 위한 탐지 기술의 발전이 필수적입니다. 시그니처 기반 탐지, 휴리스틱 탐지, 행위 기반 탐지와 같은 전통적인 방법은 여전히 유용하지만, 악성코드의 고도화에 따라 새로운 기술의 도입이 요구되고 있습니다. 특히 머신러닝과 클라우드 기반 탐지 기술은 다양한 변종 악성코드를 더 효과적으로 탐지하고 분석하는 데 중요한 역할을 하고 있습니다.

이 보고서에서 다룬 VirusTotal API는 다수의 보안 엔진을 통합하여 파일, URL, IP 주소 및 도메인에 대한 멀티 엔진 분석을 제공하는 강력한 도구입니다. 이를 통해 보안 위협에 대한 신속하고 포괄적인 평가가 가능해지며, API를 통해 자동화된 위협 탐지 및 대응 시스템을 구축할 수 있습니다.

1. 참고문헌

“안티바이러스 소프트웨어”,

https://namu.wiki/w/%EC%95%88%ED%8B%B0%EB%B0%94%EC%9D%B4%EB%9F%AC%EC%8A%A4%20%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4

“안티바이러스 소프트웨어 원리”, (2024.02.18) form

https://dasima.co.kr/767/%EC%95%88%ED%8B%B0%EB%B0%94%EC%9D%B4%EB%9F%AC%EC%8A%A4-%EC%86%8C%ED%94%84%ED%8A%B8%EC%9B%A8%EC%96%B4-%EC%9B%90%EB%A6%AC/

“안티바이러스 기능 및 성능 분석 1부”, (2023.01.27) from

https://csrc.kaist.ac.kr/blog/2023/01/27/%EC%95%88%ED%8B%B0%EB%B0%94%EC%9D%B4%EB%9F%AC%EC%8A%A4-%EA%B8%B0%EB%8A%A5-%EB%B0%8F-%EC%84%B1%EB%8A%A5-%EB%B6%84%EC%84%9D-1%EB%B6%80/

“안티바이러스 기능 및 성능 분석 2부” , (2023.03.27) from

https://csrc.kaist.ac.kr/blog/2023/03/27/%EC%95%88%ED%8B%B0%EB%B0%94%EC%9D%B4%EB%9F%AC%EC%8A%A4-%EA%B8%B0%EB%8A%A5-%EB%B0%8F-%EC%84%B1%EB%8A%A5-%EB%B6%84%EC%84%9D-2%EB%B6%80/

“정적 분석을 이용한 다형성 스크립트 바이러스의 탐지기법 설계”, (2003.4) form

https://www.dbpia.co.kr/Journal/articleDetail?nodeId=NODE00622344

“바이러스 탐지를 위한 휴리스틱 스캐닝 기법 및 행위 제한 기법 분석”

(2002.10) from

https://www.dbpia.co.kr/journal/articleDetail?nodeId=NODE00616165

“파일 바이러스 복제 특성을 이용한 바이러스 탐지 및 복구”, (2001.10) from

https://www.dbpia.co.kr/journal/articleDetail?nodeId=NODE00615587

바이러스 토탈 API 쿼리

https://blog.virustotal.com/2024/08/VT-S1-EffectiveResearch.html

악성코드 분석, 암호화 및 패키징 프로젝트

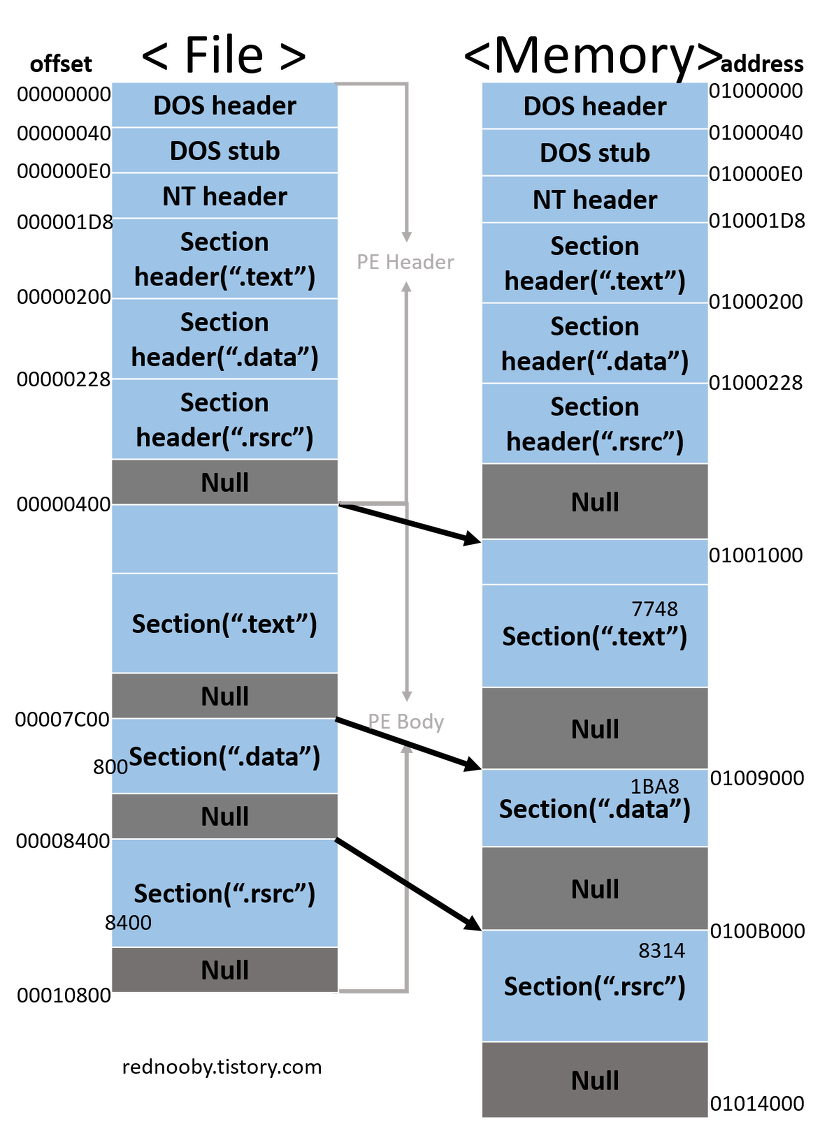
PE파일에 대한 보고서

1. PE 파일 소개

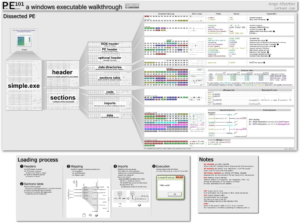
PE(Portable Executable) 파일 형식은 Windows 운영 체제에서 실행 파일(.exe)이나 DLL 파일을 지원하는 기본 형식입니다. 쉽게 말해, PE 파일은 프로그램이 메모리에 로드되어 실행될 수 있도록 도와주는 중요한 정보를 담고 있습니다. 이 파일 형식은 다양한 컴퓨터 아키텍처, 예를 들어 x86, x64,ARM등에서 사용할 수 있도록 설계되었습니다.

처음에는 MS-DOS 실행 파일을 확장한 형태로 시작되었지만, 지금은 현대적인 Windows 시스템과 DOS 사이의 호환성을 유지하는 역할을 하고 있습니다. . PE 파일은 효율성과 유연성이 뛰어나, 필요에 따라 메모리에 적재될 데이터를 최적화하여 프로그램의 성능과 보안을 높일 수 있습니다. 예를 들어, 실행할 코드와 데이터는 따로따로 저장되어 프로그램이 더 빠르고 안전하게 작동하도록 도와줍니다.

1. PE 파일의 구조



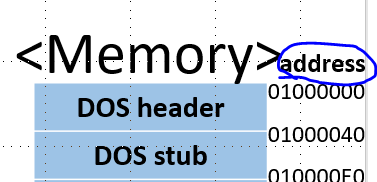
PE 구조



[PE파일(메모장.exe)이 메모리에 로딩되는 모습]

PE header: DOS header ~ Section header

PE Body: 그밑 Section

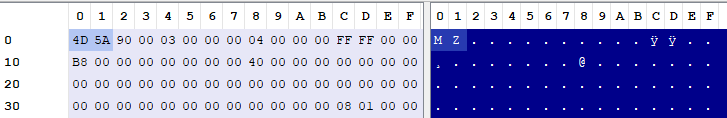


[VA(Virtual Address, 절대주소)]

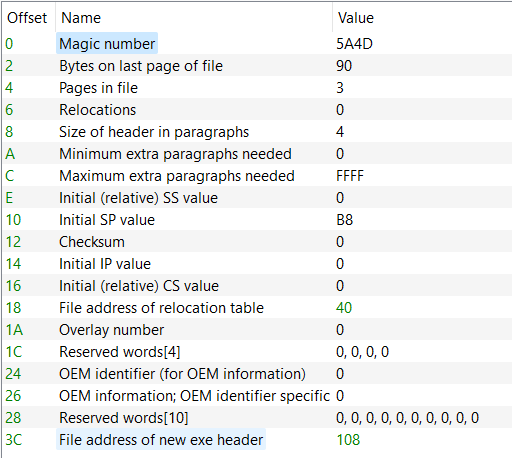
1. DOS 헤더 (MZ 헤더)

파일의 맨 앞부분에 위치한 MZ 헤더는 오래된 MS-DOS 형식의 흔적입니다. 이 부분에는 작은 DOS 프로그램이 포함되어 있어, DOS 환경에서 실행될 경우 "이 프로그램은 DOS 모드에서 실행될 수 없습니다"라는 메시지를 출력합니다. 현대 시스템에서는 필수적이지 않지만, 이전 시스템과의 호환성을 위해 여전히 유지되고 있습니다.

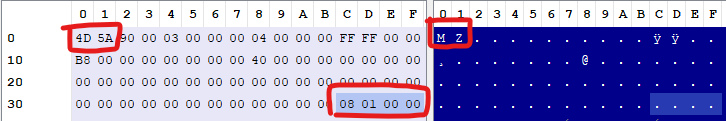
실행 파일(.exe)을 16진수 편집기로 열어보면 가장 먼저 MZ 헤더가 보입니다. 이는 운영 체제에서 유효한 실행 파일인지 확인하는 서명(또는 식별자)으로 사용됩니다. PE 헤더의 "PE" 문자열과 함께 파일의 앞부분에 위치합니다.



하지만 단순히 "MZ" 서명만 있는 것이 아닙니다. 그 뒤에 추가 정보가 이어집니다:



Hex Editor로 자세히 살펴보면 파일 시작 부분에 Hex 형식으로 정보가 표시되어 있습니다.

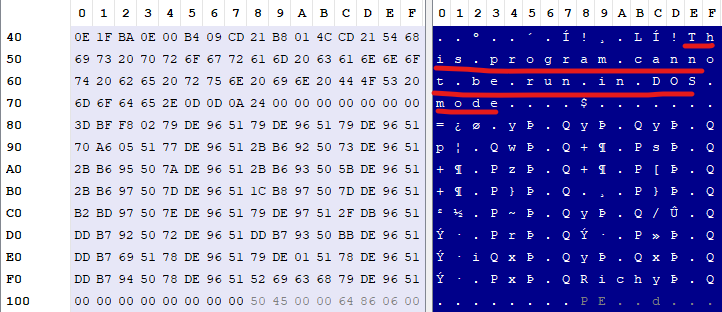


표에서 "매직 넘버" 0x5A4D는 "MZ" 문자열의 16진수 값과 정확히 일치합니다. 또한 0x108은 int값(4바이트, 32비트)으로 역순 설정됩니다: 0x08 0x01 0x00 0x00. 이는 16진수 편집기가 낮은 주소에서 높은 주소로 16진수 바이트를 출력하고, 리틀 엔디언(예: x86/ia32) 기계에서는 다중 바이트 엔터티의 낮은 자릿수를 낮은 주소에 저장하기 때문입니다.

DOS Stub

이는 단 하나의 목적을 가진 작은 DOS 프로그램입니다. 프로그램이 DOS에서 실행될 경우 "이 프로그램은 DOS 모드에서 실행할 수 없습니다"라는 텍스트를 출력하고 종료하는 것입니다.

DOS Stub은 MZ 헤더 바로 뒤에 위치합니다.



1. PE 헤더(또는 NT헤더)

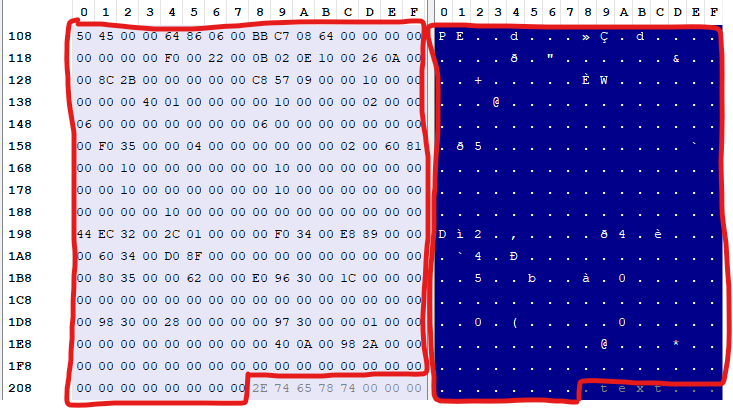
PE 헤더는 MZ 헤더 다음에 위치하며, 현대 Windows 실행 파일의 시작점입니다. 주요 구성 요소는 다음과 같습니다:

1. COFF 파일 헤더: 대상 플랫폼(예: 머신 타입), 섹션 수, 타임스탬프 등의 메타데이터를 포함합니다.

2. 옵션 헤더: 이름과 달리 실행 파일에서 필수적인 헤더로, 프로그램의 로드 및 실행에 중요한 정보를 담고 있습니다. 여기에는 프로그램의 시작 지점(엔트리 포인트), 코드 크기, 데이터 위치 등의 정보가 포함됩니다.

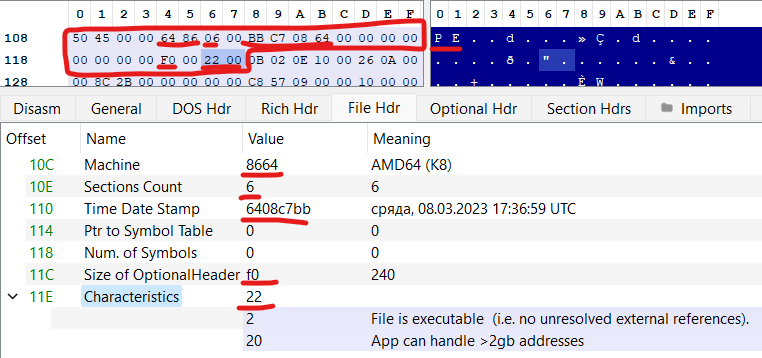
PE 헤더에는 애플리케이션에 대한 중요한 정보가 들어 있으며, 구체적으로 Windows 로더가 프로그램을 로드하고 실행하는 데 사용할 정보가 포함되어 있습니다.

"PE" 문자열로 시작하는데, 이는 Portable Executable 파일이라는 서명이기도 합니다.



1. 이미지 파일 및 선택 헤더

이미지 파일 헤더의 개요는 다음과 같습니다:

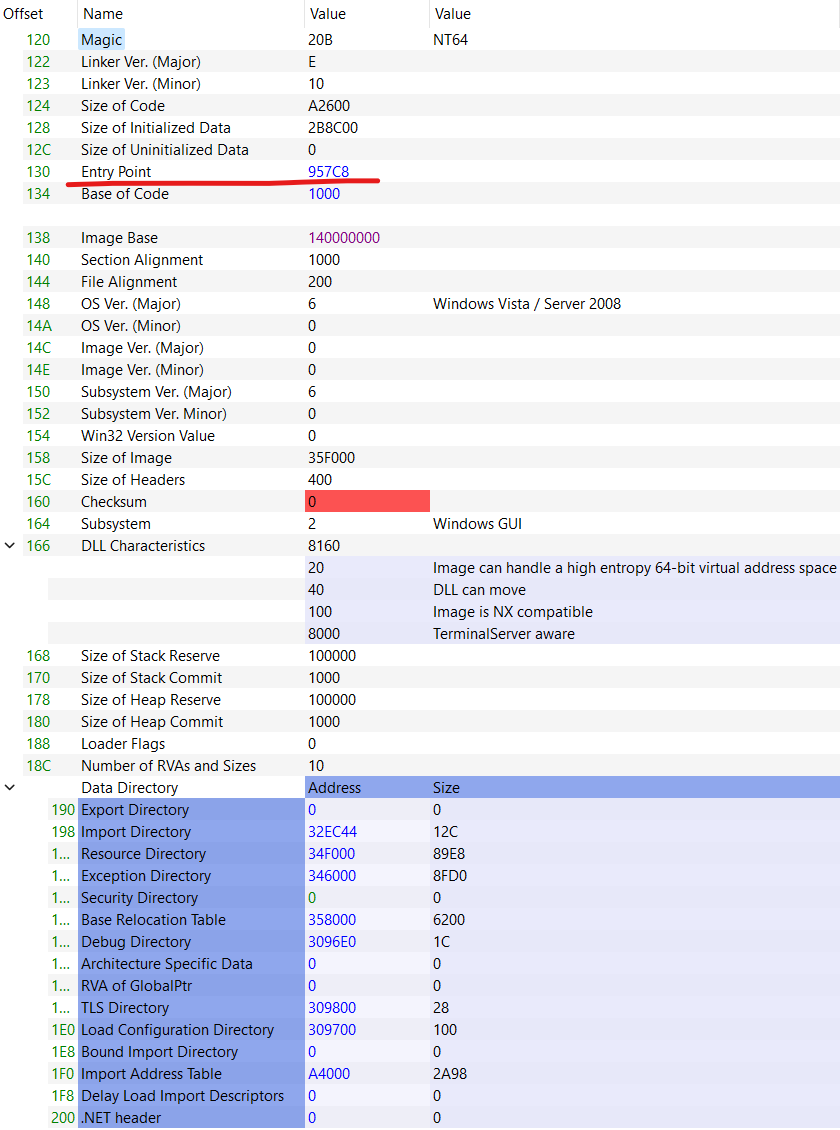


여기에는 실행 파일이 컴파일된 대상 머신 정보, 섹션 수, 타임스탬프 등이 포함되어 있습니다.

그 다음은 이미지 선택 헤더입니다. 이 헤더에는 이미지 파일 헤더보다 훨씬 더 많은 정보가 들어 있습니다. Entry Point는 실행 코드의 진입점으로, 사실상 애플리케이션의 "main()" 함수입니다. Windows 로더가 실행을 시작하는 지점입니다.

이 값 0x957C8은 RVA(상대 가상 주소)로, 실제 파일 오프셋이 아닙니다. Windows는 Image Base 값을 더해 VA(가상 주소)로 변환하고, Windows 로더가 로드하고 매핑한 후 이 코드의 가상 메모리 주소를 얻습니다. 이 주소를 "VA"라고 부르는 이유는 Windows가 각 프로세스에 대해 물리적 메모리(RAM)와 독립적인 고유한 VA 공간을 생성하기 때문입니다.

즉, 애플리케이션을 로드할 때 Windows는 메모리에 프로세스 공간을 할당한 다음 파일을 로드하고 확장합니다. 그런 다음 이 Entry Point 주소를라 메모리에서 코드 실행을 시작합니다.

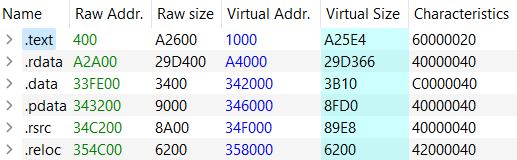


1. 섹션 헤더(또는 섹션 테이블)

PE 헤더 이후 섹션 테이블은 파일의 다른 영역을 정의합니다. 일반적인 섹션은 다음과 같습니다

|  |  |
| --- | --- |
| .text | 프로그램의 실행 가능한 코드가 들어 있는 섹션. |
| .rdata | 문자열 또는 상수와 같은 읽기 전용 데이터가 포함됩니다. |
| .data | 초기화된 변수들이 들어 있습니다. |
| .rsrc | 아이콘, 비트맵, 대화 상자와 같은 리소스. |
| .reloc | 동적 연결을 위한 코드 재배치 정보 |
| .bss | 초기화되지 않은 데이터 |
| .idata | 애플리케이션에서 호출하는 모든 API에 대한 데이터 가져오기 |
| .edata | 데이터 내보내기- 애플리케이션에서 코딩한 모든 공개 API |

섹션 헤더(또는 섹션 테이블)에는 파일의 섹션에 대한 정보가 들어 있습니다. 이는 다양한 요인에 따라 파일마다 다를 수 있습니다.



여기에는 주로 파일의 각 섹션에 대한 주소, 크기, 특성이 포함됩니다. 바이너리 코드 주입을 수행하려면 새로운 섹션을 추가하거나 기존 섹션을 변경하여 크기를 변경해야 하므로 여기 값을 변경해야 합니다.

1. 결론

PE(Portable Executable) 파일은 Windows 시스템에서 실행 파일과 라이브러리의 중요한 형식으로, 그 구조와 작동 원리를 이해하는 것은 보안과 성능 최적화에 필수적입니다. PE 파일의 헤더, 섹션, 엔트리 포인트 등 구성 요소는 프로그램이 메모리에 적재되고 실행되는 방식에 큰 영향을 미칩니다.

이 보고서에서는 PE 파일의 구조를 상세히 분석하고, 악성코드 탐지와 보안 취약점 분석에 있어서 PE 파일의 중요성을 강조했습니다. 특히 PE 파일의 특정 섹션에 바이너리 코드를 주입하거나 변조하는 기술은 악성코드가 자주 사용하는 방법이므로, PE 파일 분석은 보안 전문가에게 필수적인 작업입니다.

결론적으로, PE 파일의 구조를 깊이 이해하는 것은 악성코드 탐지 및 보안 강화를 위해 매우 중요합니다. 이러한 지식을 바탕으로 더 강력하고 신뢰할 수 있는 보안 솔루션을 개발하고, 악성 행위로부터 시스템을 보호하는 데 기여할 수 있을 것입니다.

1. 참고문헌

“System.Reflection.PortableExecutable Namespace”

https://learn.microsoft.com/en-us/dotnet/api/system.reflection.portableexecutable?view=net-8.0

” System.Reflection.PortableExecutable API”

https://apisof.net/catalog/044f0ee831a1a64e9a2cbdfd1884b985?fx=net46

“The Windows Portable Executable (PE) File Format”, (2023.12.17) form

https://yuriygeorgiev.com/2023/12/18/windows-portable-executable-pe-file-format/

“PE 구조와 인공신경망을 사용한 패커 식별 방안 연구”, (2020.02) form

https://repository.hanyang.ac.kr/handle/20.500.11754/123806

“윈도우 PE 포맷 바이너리 데이터를 활용한 Bidirectional LSTM 기반 경량 악성코드 탐지모델”, (2022.02.23)

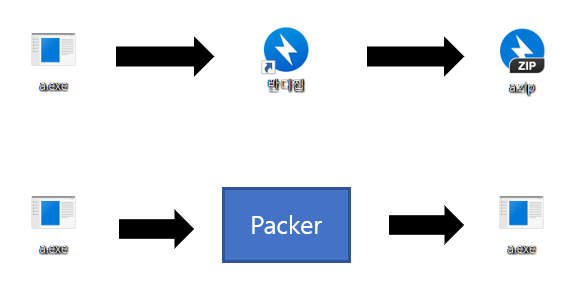
http://www.jics.or.kr, 한국 인터넷 정보학회(23권 1호)

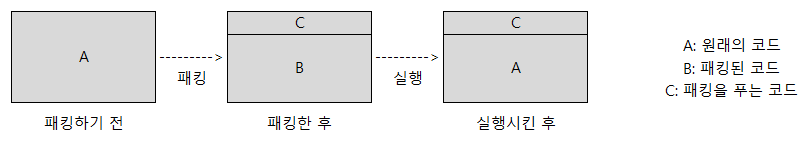
악성코드 분석, 암호화 및 패키징 프로젝트

Protector(프로텍터)에 대한 보고서

1. 패킹의 소개

패킹은 프로그램을 압축하는 방법 중 하나로, 압축을 푸는 과정 없이도 바로 실행이 가능한 압축 방식을 의미합니다. 흔히 사용하는 ZIP, 7Z 같은 압축 파일은 압축을 풀어야 실행할 수 있지만, 패킹된 파일은 그럴 필요가 없습니다. 이는 파일 내부에 패킹을 푸는 코드가 포함되어 있기 때문에, 파일을 실행하면 자동으로 압축이 풀리고 프로그램이 실행됩니다.



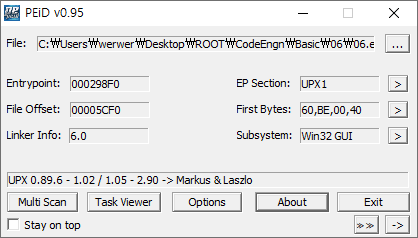


패킹을 사용하는 이유

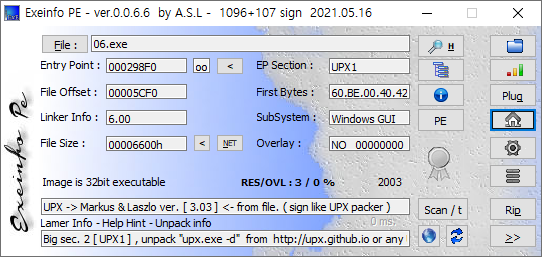
패킹을 하는 이유는 크게 두 가지입니다. 첫 번째로, 프로그램을 분석하기 어렵게 만들 수 있습니다. 패킹을 하면 코드를 분석하기 전에 먼저 압축을 풀어야 하므로 악성코드처럼 분석이 어려워져 백신 개발이 지연될 수 있습니다. 두 번째 이유는 프로그램 크기를 줄이기 위해서입니다. 패킹을 통해 데이터를 압축하여 용량을 줄일 수 있습니다.

패킹 관련 도구

대표적인 패킹 도구로는 UPX, ASPack, Themida 등이 있으며, PEiD나 Exeinfo PE 같은 도구는 프로그램이 패킹되었는지 여부를 확인하는 데 사용됩니다. PEiD는 64비트 파일 분석을 지원하지 않지만, Exeinfo PE는 다양한 정보를 제공하여 더 많은 분석이 가능합니다.



PEiD 실행 화면 https://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/PEiD-updated.shtml



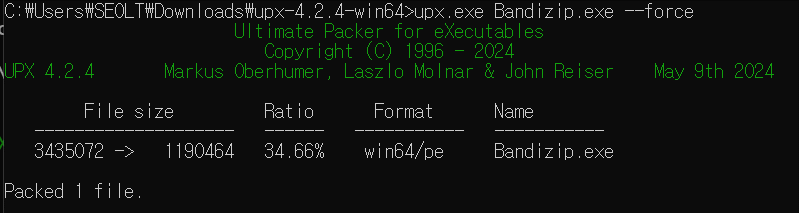
Exeinfo PE 실행 화면 http://exeinfo.booomhost.com/?i=1

1. UPX

UPX(Universal Packer for Executables)는 실행 파일을 압축하여 크기를 줄이는 무료 도구로, 안전하고 휴대성이 뛰어나며 성능이 우수합니다. 주로 소프트웨어 개발자들이 파일 크기를 최적화하고 배포를 효율적으로 하기 위해 사용됩니다.

UPX는 프로그램 실행 파일이나 DLL 파일의 크기를 약 50%에서 70%까지 줄여주며, 이로 인해 배포 파일 크기를 줄이고 다운로드 속도를 높이는 데 매우 유용합니다. UPX로 압축된 파일은 기존과 동일한 방식으로 동작하며, 추가적인 메모리나 성능 상의 불이익 없이 압축 해제가 이루어집니다.

또한 Windows, macOS, Linux 등의 다양한 운영체제에서 실행 파일을 압축할 수 있으며, GNU General Public License(GPL) v2+ 하에 배포되는 오픈 소스 프로젝트입니다. 이를 통해 누구나 소스 코드를 열람하고 수정할 수 있으며, 상업적 용도로도 자유롭게 사용할 수 있습니다.



UPX로 Bandizip.exe를 압축하는 모습

|  |  |
| --- | --- |
|  |  |
| 기존 반디집 PE | UPX 압축 후 반디집 PE |

기능과 특징

1. 뛰어난 압축 성능

UPX는 다른 일반적인 압축 도구보다 더 높은 압축률을 자랑합니다. 실행 파일 크기를 절반 이상 줄여 파일을 빠르게 다운로드하고, 네트워크 대역폭을 절감하는 데 도움이 됩니다. UPX는 ZIP과 같은 포맷보다 더 나은 압축률을 제공하며, 여러 상황에서 효율적인 파일 관리가 가능합니다.

1. 빠른 압축 해제

UPX로 패킹된 파일은 실행 시 자동으로 해제되며, 성능 저하 없이 즉각적인 실행이 가능합니다. 최신 시스템에서 UPX로 압축된 파일은 초당 500MB 이상의 속도로 해제되며, 실행 파일 크기를 줄이는 데도 불구하고 프로그램의 성능에 영향을 미치지 않습니다.

1. 다양한 파일 형식 지원

UPX는 Windows의 PE 파일, Linux의 ELF 파일, macOS의 Mach-O 파일을 포함한 여러 운영체제의 실행 파일 형식을 지원합니다. 이처럼 다양한 형식의 파일을 압축하고 관리할 수 있기 때문에 개발자들은 여러 환경에서 UPX를 사용해 소프트웨어를 최적화할 수 있습니다.

1. 오픈 소스 및 자유로운 배포

UPX는 오픈 소스로 제공되며, 누구나 소스 코드를 열람하고 수정할 수 있습니다. 또한 상업용 응용 프로그램에서도 자유롭게 사용할 수 있어, 배포 및 활용 측면에서 유연성이 높습니다. GPL v2+ 라이선스 하에 배포되어 있어 상업용 소프트웨어에서도 무제한적으로 사용할 수 있습니다.

1. 안전성과 투명성

UPX는 오픈 소스이기 때문에 보안 소프트웨어에서 패킹된 파일의 내부를 분석할 수 있으며, 이를 통해 악성코드 여부를 판별할 수 있습니다. 또한 UPX는 파일 무결성을 유지하는 체크섬 기능을 제공하여, 압축 전후의 파일이 손상되지 않았음을 확인할 수 있습니다.

1. 프로텍터(Protector)의 소개

소프트웨어 프로텍터(Protector)는 프로그램의 무단 복제, 역공학(리버스 엔지니어링), 디버깅 등을 방지하기 위해 다양한 보안 기술을 사용하는 도구입니다. 이는 프로그램의 코드 흐름 분석을 방해하고, 악의적인 공격자가 프로그램을 분석하거나 변조하는 것을 막기 위한 목적으로 사용됩니다. 주로 상용 소프트웨어, 게임, 민감한 데이터를 다루는 프로그램에서 소프트웨어 보호 및 라이선스 관리를 위해 사용되며, 악성코드에서도 악용될 수 있는 기술입니다.

프로텍터는 기본적인 파일 패커 기술을 바탕으로 다양한 분석 방해 기술을 추가하여 프로그램을 보호합니다. 알려진 대표적인 프로텍터로는 ASProtect, Enigma, Obsidium, Themida, VMProtect 등이 있으며, 이들 모두 프로그램 분석을 지연시키고 역공학을 어렵게 만드는 것을 목표로 합니다.

프로텍터의 주요 기능

1. 코드 난독화 (Code Obfuscation)

코드 난독화는 프로그램의 코드를 알아보기 어렵게 변환하여 공격자가 역공학을 시도할 때 이를 이해하기 어렵게 만드는 기술입니다. 변수명, 함수명, 제어 흐름 등이 복잡하게 바뀌며, 이는 코드 분석 툴을 사용하더라도 원래의 의미를 파악하기 어렵게 만듭니다.

1. 안티 디버깅 (Anti-Debugging)

안티 디버깅 기술은 프로그램이 실행되는 동안 디버거를 사용하여 프로그램을 분석하려는 시도를 탐지하고 차단하는 기능입니다. 디버거가 감지되면 프로그램이 비정상적으로 종료되거나 오작동을 유도할 수 있습니다.

1. 무결성 검증 (Integrity Check)

무결성 검증은 프로그램의 데이터를 변조하거나 수정하려는 시도를 탐지하는 기술입니다. 분석 도구를 사용해 프로그램의 코드를 패치하거나 소프트웨어 브레이크포인트를 삽입하는 등의 시도를 방지합니다. 이를 통해 프로그램의 무결성을 보장하고, 변조된 경우 프로그램이 정상적으로 작동하지 않도록 합니다.

.

1. API 난독화 (API Obfuscation)

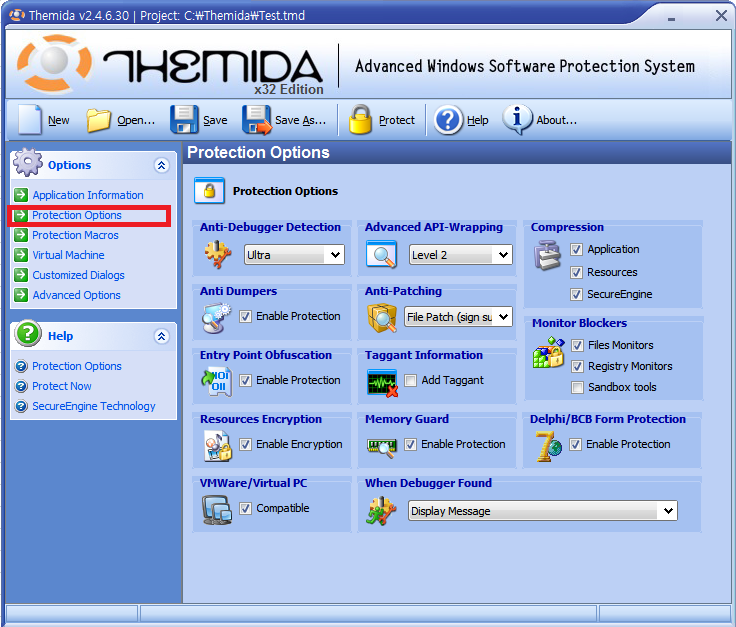
프로그램이 사용하는 API 호출 정보를 숨기거나 난독화하여 API 기반의 분석을 어렵게 만듭니다. 공격자는 보통 프로그램이 사용하는 외부 라이브러리나 API를 통해 프로그램의 동작을 분석하려 하기 때문에, 이를 난독화하여 분석이 어렵게 만듭니다.

1. 코드 가상화 (Code Virtualization)

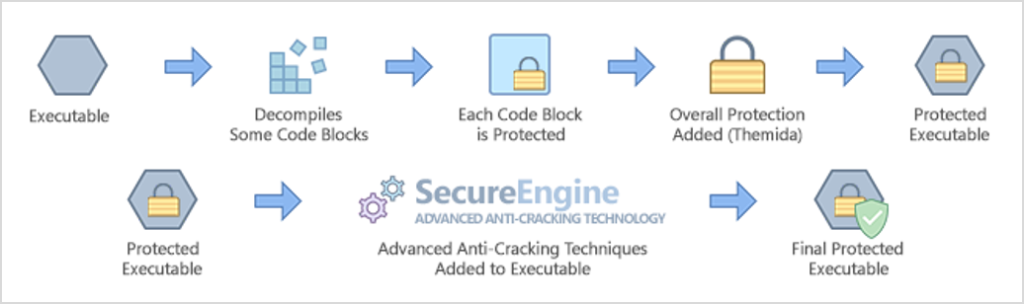
코드 가상화는 프로그램의 중요한 코드 부분을 가상의 CPU에서 실행되도록 변환하는 기술입니다. 이는 원래의 코드를 바이너리 코드로 변환한 후, 가상 CPU가 이 코드를 해석하여 실행하도록 만들어 일반적인 역공학 도구로 분석하기 매우 어렵게 만듭니다.

1. Themida

Themida는 소프트웨어를 크랙과 역공학으로부터 보호하기 위해 개발된 상용 소프트웨어 보호 도구입니다. 주로 게임과 상업용 소프트웨어에서 사용되며, 다양한 난독화 및 역공학 방지 기법을 제공하여 프로그램의 무결성을 유지하고 분석을 어렵게 만듭니다. Themida는 원본 코드의 제어 흐름을 보호하는 동시에 분석을 방해하는 코드를 함께 실행해 역공학을 방지하는 방식을 채택하고 있습니다. 이로 인해 파일 크기가 늘어나고 실행 속도가 다소 느려질 수 있지만, 강력한 보호 성능 덕분에 소프트웨어 보안에 널리 사용되고 있습니다.



Themida 화면



Themida 작동 방식

Themida의 주요 기능

Themida는 소프트웨어 보호를 위해 다양한 보호 옵션을 제공합니다. 이 옵션들은 소프트웨어를 역공학, 크랙, 디버깅, 코드 패치 등 다양한 공격으로부터 보호하며, 각 보호 옵션을 독립적으로 적용하거나 결합해 사용 가능합니다.

1. 난독화 (Obfuscation)

Themida의 핵심 기능 중 하나는 코드 난독화입니다. 난독화는 프로그램의 원본 코드가 분석자에게 쉽게 파악되지 않도록 복잡하게 변경하는 기술입니다. Themida는 원본 코드의 제어 흐름을 유지하면서도, 역공학을 방해하는 다양한 난독화 기술을 적용해 분석을 어렵게 만듭니다. 대표적인 난독화 기법으로는 Entrypoint Obfuscation이 있으며, 이를 통해 역공학 도구가 프로그램의 진입점을 찾기 어렵게 합니다.

Entrypoint Obfuscation: 프로그램이 실행되는 진입점을 난독화하여 역공학 도구가 이를 추적하기 어렵게 만듭니다.

API Wrapping: 프로그램이 외부 API를 호출하는 방식을 숨겨, 역공학 도구가 API 호출을 분석하지 못하도록 만듭니다.

1. 암호화 및 압축 (Encryption & Compression)

Themida는 프로그램의 중요한 코드와 데이터를 암호화하고 압축하는 기능을 제공합니다. 이를 통해 프로그램이 실행되지 않는 동안에는 코드가 보호되며, 실행 시점에만 복호화되어 메모리에서만 실행되기 때문에 역공학 도구로 분석하기 매우 어렵습니다. 또한, 암호화와 압축을 통해 파일 크기를 줄이거나 보호 레벨을 강화할 수 있습니다.

Encryption: 중요한 코드와 데이터를 암호화하여 프로그램이 실행될 때에만 해당 코드가 복호화되도록 합니다.

Compression: 실행 파일 크기를 줄이고 보호 레벨을 향상시키기 위해 코드와 리소스를 압축합니다.

1. 안티 디버깅 및 안티 분석 (Anti-Debugging & Anti-Analysis)

Themida는 소프트웨어가 디버거와 같은 분석 도구로부터 보호될 수 있도록 다양한 안티 디버깅 및 안티 분석 기술을 제공합니다. 이러한 보호 옵션들은 디버깅 시도가 감지되면 프로그램을 종료하거나 알람 메시지를 띄워주며, 해커나 분석자가 소프트웨어를 수정하거나 분석하는 것을 방지합니다.

Anti-Debugging: 디버거가 감지되면 프로그램이 종료되거나 오류가 발생하도록 합니다.

Anti-Analysis: 모니터링 도구나 분석 도구가 감지될 때 프로그램을 종료하거나 방해합니다.

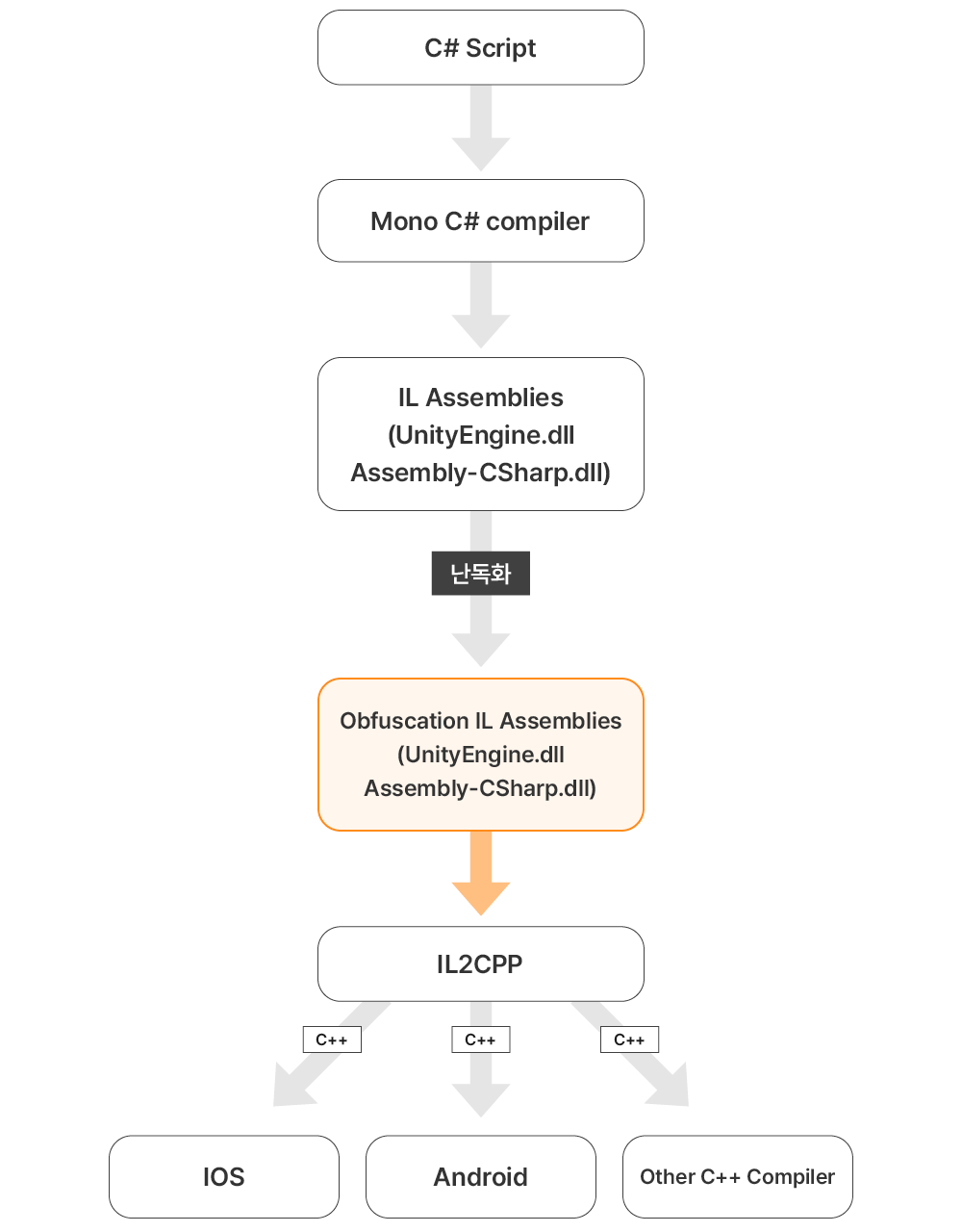
VM 탐지: Themida는 프로그램이 가상 머신 환경(VMware, VirtualPC)에서 실행 중인지 탐지하여 가상 머신에서의 분석 시도를 방지합니다.

1. IAT 제거 및 API 보호 (IAT Removal & API Wrapping)

Themida는 소프트웨어 보호를 위해 Import Address Table(IAT)을 제거하는 기능을 제공합니다. IAT는 프로그램이 실행 중에 사용하는 API 목록을 관리하는데, 이를 제거함으로써 공격자가 API 호출을 추적하는 것을 어렵게 만듭니다. 또한, API 호출을 숨기는 API Wrapping 기능도 제공하여 역공학 도구가 API 기반으로 분석하지 못하게 합니다.

IAT Removal: Import Address Table을 제거해 프로그램이 호출하는 API 정보를 숨깁니다.

API Wrapping: 외부 API 호출을 감춰 API 분석을 방해합니다.



난독화 과정

출처 : https://story.cookapps.com/articles/189

1. 컴파일러에서 Assembly-CSharp.dll, UnityEngine.dll 등이 생성

2. 생성된 .dll을 Mono Cecil을 이용하여 수정

3. 수정된 .dll을 il2cpp를 통해 C++로 변환되어 빌드

// 난독화 전

1. [Token(Token = "0x2000002")]

2. public class GoldManager

3. {

4. [Token(Token = "0x4000001")]

5. [FieldOffset(Offset = "0x8")]

6. private int \_gold;

7. [Token(Token = "0x6000001")]

8. [Address(Offset = "0x1E2C14", RVA = "0x1E2C14", VA = "0x1E2C14")]

9. public bool UseGold(int gold) => new bool();

10. [Token(Token = "0x6000002")]

11. [Address(Offset = "0x1E2C34", RVA = "0x1E2C34", VA = "0x1E2C34")]

12. public GoldManager() {}

13. }

14.

15. // 난독화 후

16. [Token(Token = "0x2000002")]

17. public class CCOOFEENFCA

18. {

19. [Token(Token = "0x4000001")]

20. [FieldOffset(Offset = "0x8")]

21. private int JODFLMBMOHC;

22. [Token(Token = "0x6000001")]

23. [Address(Offset = "0x1E2C64", RVA = "0x1E2C64", VA = "0x1E2C64")]

24. public bool JMPNPBAGNFF(int BEMMJODKCFC) => new bool();

25. [Token(Token = "0x6000002")]

26. [Address(Offset = "0x1E2C84", RVA = "0x1E2C84", VA = "0x1E2C84")]

27. public CCOOFEENFCA() {}

28. }

29.

난독화 전(위)과 후(아래) decompile 비교

출처 : https://story.cookapps.com/articles/189

1. 결론

프로텍터는 소프트웨어 보호에서 매우 중요한 역할을 담당합니다. 특히, Themida와 같은 고급 보호 도구는 코드 난독화, 안티 디버깅, 암호화 등의 다양한 기술을 활용하여 소프트웨어를 역공학 및 크랙 시도로부터 보호합니다. Themida는 상업용 소프트웨어와 게임에서 자주 사용되며, 이러한 난독화 및 보호 기법을 통해 해커가 프로그램의 내부 구조를 파악하거나 악의적으로 변경하는 것을 방지합니다.

문서에서 설명한 바와 같이, Themida는 보호의 수준이 매우 높지만, 그만큼 파일 크기가 증가하거나 실행 속도가 저하될 수 있다는 단점도 존재합니다. 또한, 여러 보호 옵션을 조합해 사용하는 과정이 복잡할 수 있습니다. 하지만, 소프트웨어의 무결성을 보장하고 역공학을 방지하는 데 있어 중요한 역할을 한다는 점에서 Themida는 소프트웨어 보호를 위한 매우 유용한 도구라 할 수 있습니다.

Ⅵ. 참고 문헌

“프로텍터 유형의 분석방해 기술 우회방안에 관한 연구” (2017.06),

https://dcollection.korea.ac.kr/srch/srchDetail/000000076259

“실행 파일 형태로 복원하기 위한 Themida 자동 역난독화 도구 구현” (2017.08),

https://scienceon.kisti.re.kr/srch/selectPORSrchArticle.do?cn=JAKO201711656578275

“가상화 난독화 기법이 적용된 실행 파일 분석 및 자동화 분석 도구 구현”, (2013.08) from

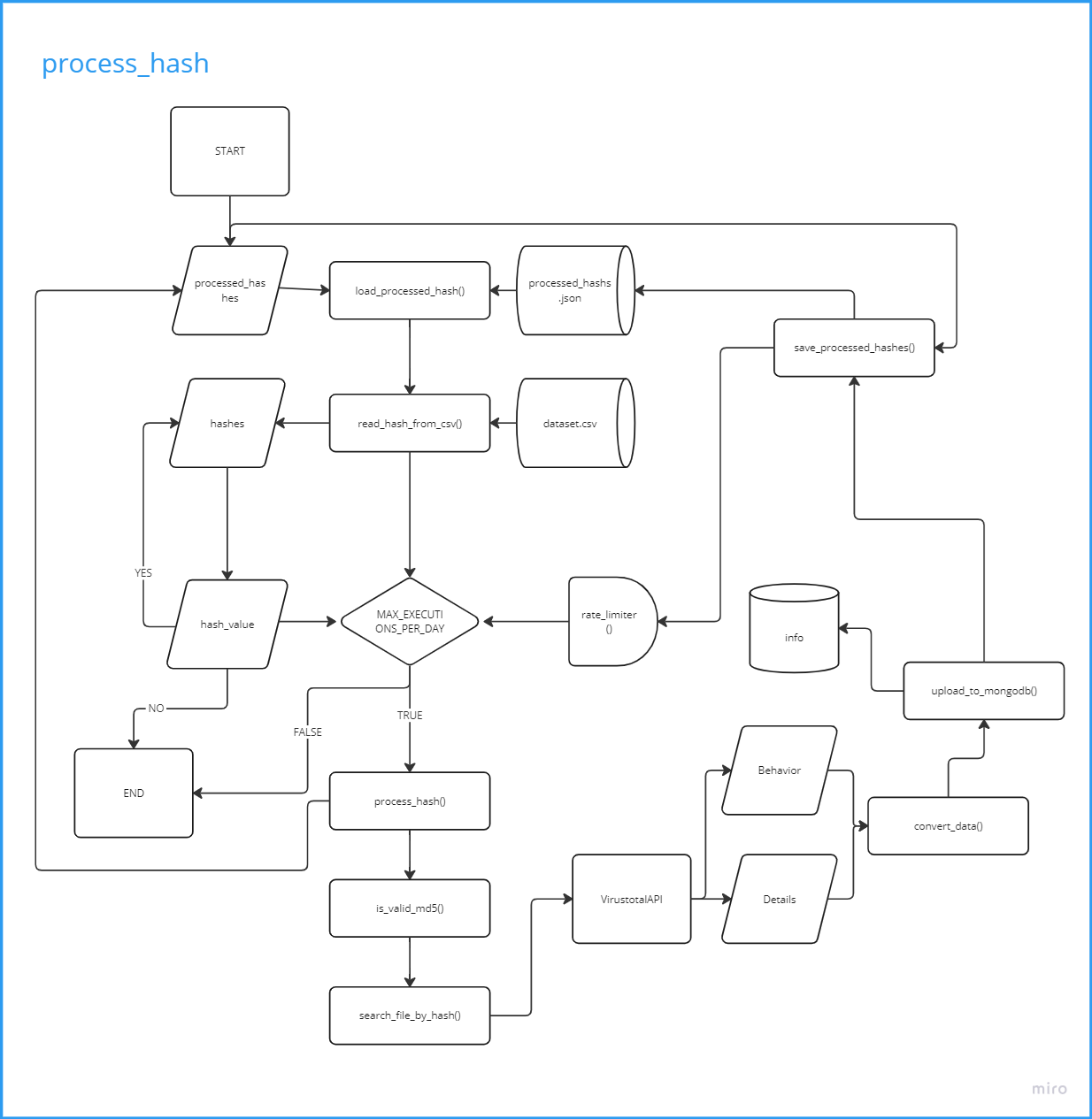
https://koreascience.or.kr/article/JAKO201326940560978.page

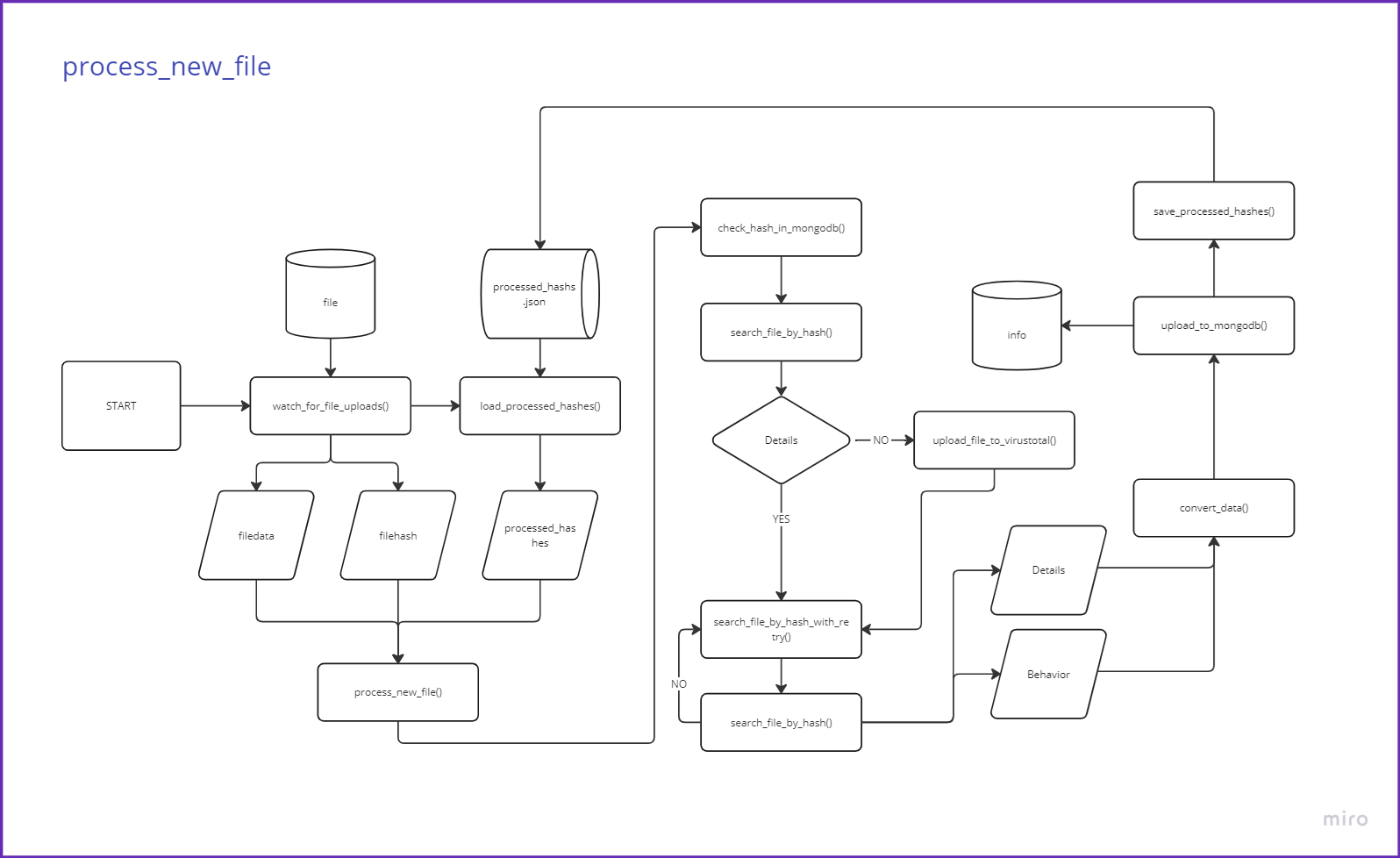
“Themida의 API 난독화 분석과 복구방안 연구”, (2017.02) from

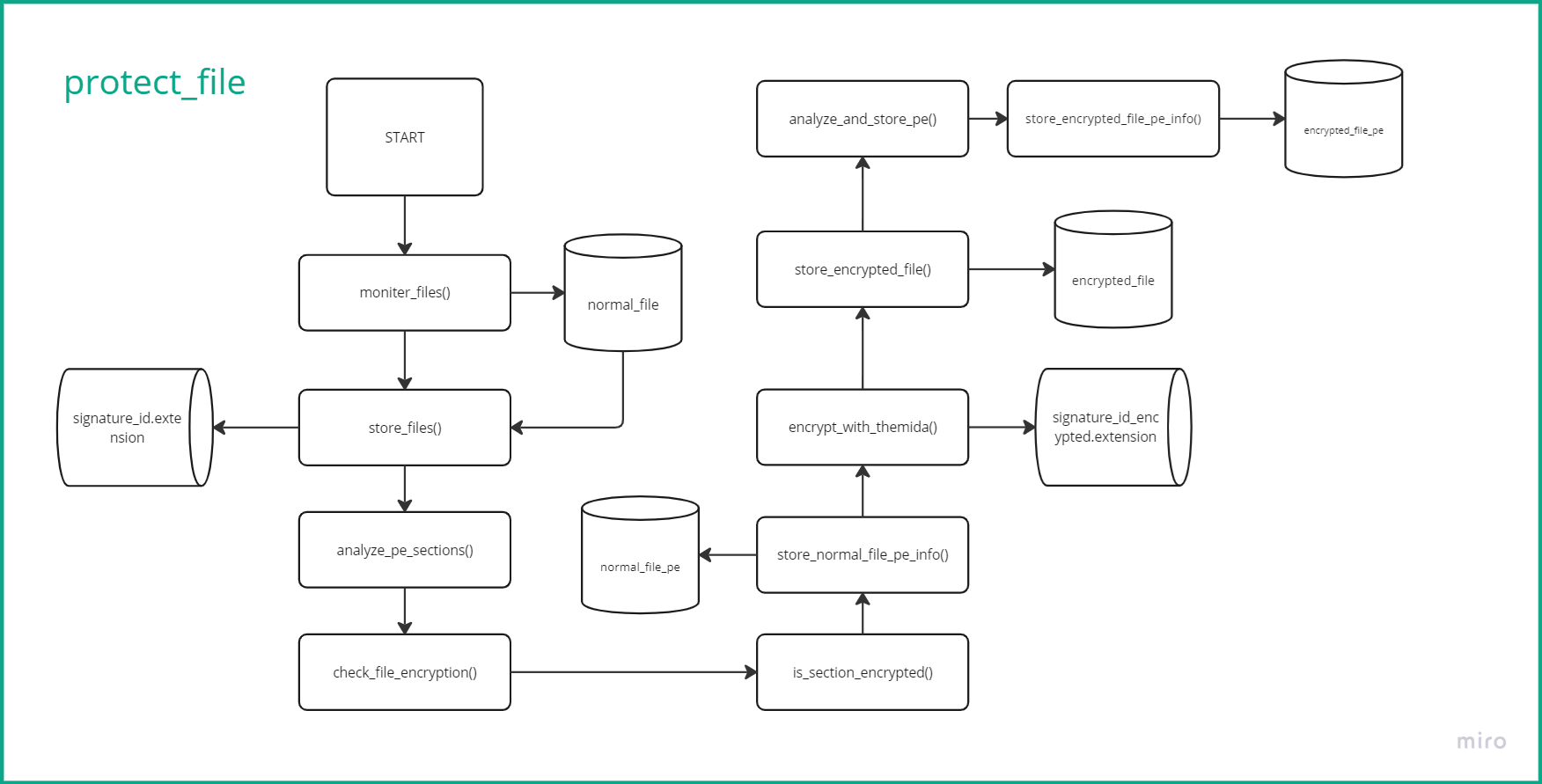
https://www.kci.go.kr/kciportal/ci/sereArticleSearch/ciSereArtiView.kci?sereArticleSearchBean.artiId=ART002200969

기능 구현

모듈 별 동작도







테스트

결론

프로젝트 과정 중 어려움 및 느낀점

참고자료