악성코드 탐지, 암호화 및 패키징

파일 프로텍터 개발 명세서

문서번호 : DS-102

VER1.1

```python
from pymongo import MongoClient
import gridfs
import downloadDB_Nor  # 파일 저장 모듈
from logging_Utils import setup_logger  # 공통 로그 설정 함수
from dotenv import load_dotenv
import os

# 환경 변수 로드
load_dotenv()
MONGO_URI = os.getenv("MONGO_URI")

# 로그 설정
log_file = r'C:\pymodules\log\db_monitor.log'
logger = setup_logger(log_file)

def monitor_files():
    """MongoDB Change Stream을 사용하여 normal_files.filedata 테이블을 모니터링"""
    try:
        # MongoDB에 연결
        client = MongoClient(MONGO_URI)
        db = client['normal_files']
        fs = gridfs.GridFS(db)  # GridFS 인스턴스 생성

        # Change Streams로 normal_files에서 파일 업로드를 감시
        with db.filedata.watch() as stream:
            logger.info("Started monitoring new files in MongoDB using Change Streams.")
            for change in stream:
                if change["operationType"] == "insert":
                    file_doc = change["fullDocument"]
                    logger.info(f"New file detected: {file_doc['filename']}")

                    # GridFS에서 파일 가져오기 (gridfs_file_id를 사용)
                    gridfs_file_id = file_doc.get('gridfs_file_id')
                    if gridfs_file_id:
                        try:
                            # GridFS에서 파일 데이터를 가져옴
                            grid_out = fs.get(gridfs_file_id)
                            file_data = grid_out.read()

                            # 로컬에 파일 저장 (파일 저장 모듈에 맞게 수정 필요)
                            local_file_path = downloadDB_Nor.store_file(file_doc, file_data)
                            if local_file_path:
                                logger.info(f"File {file_doc['filename']} has been saved to
{local_file_path}")
                            else:
                                logger.error(f"Failed to save file {file_doc['filename']} to
local storage.")
                        except gridfs.errors.NoFile:
                            logger.error(f"File not found in GridFS for file ID:
{gridfs_file_id}")
                    else:
                        logger.error(f"No gridfs_file_id found in the document for file:
{file_doc['filename']}")

                else:
                    logger.info(f"Ignored operation: {change['operationType']}")

    except Exception as e:
        logger.error(f"Error occurred during file monitoring: {e}", exc_info=True)

if __name__ == "__main__":
    monitor_files()
```

목적	Mongo DB에 올라온 유저가 올린 파일 모니터링 모듈
파일명	monitorDB.py

```python
 import os
from pymongo import MongoClient
from datetime import datetime
import convert_NorToEnc  # 암호화 모듈
import uploadDB_NorPE  # PE 섹션 분석 및 저장 모듈
import check_Enc  # 암호화 확인 모듈
from logging_Utils import setup_logger  # 로그 설정 함수
from dotenv import load_dotenv
import gridfs
load_dotenv()

#env 에서 로드
MONGO_URI=os.getenv("MONGO_URI")
# 로그 파일 설정
log_file = r'C:\pymodules\log\db_file_store.log'
logger = setup_logger(log_file)

# MongoDB 연결 설정
client = MongoClient(MONGO_URI)
db = client['normal_files']
collection = db['filedata']
fs = gridfs.GridFS(db)
def store_file(file_doc, file_data):
    """
    DB 에서 파일을 다운로드하고 signature_id 로 로컬에 저장 (확장자 필드 사용).
    :param file_doc: 파일의 메타데이터 문서
    :param file_data: GridFS 에서 가져온 파일의 바이너리 데이터
    """
    try:
        signature_id = file_doc["signature_id"]
        file_extension = file_doc.get("file_extension", "")

        # 확장자가 없거나 잘못된 형식일 경우 수정
        if not file_extension.startswith("."):
            file_extension = f".{file_extension}"

        # 오늘 날짜를 기준으로 폴더 생성
        today_date = datetime.today().strftime('%Y-%m-%d')
        local_folder_path = os.path.join(r'C:\DBFiles', today_date, 'origin_files')

        # 로컬 폴더가 존재하지 않으면 생성
        if not os.path.exists(local_folder_path):
            os.makedirs(local_folder_path)

        # 파일명을 signature_id 로 하고, 확장자를 적용
        local_file_path = os.path.join(local_folder_path,
f"{signature_id}{file_extension}")

        # 파일 데이터를 로컬에 저장
        with open(local_file_path, "wb") as f:
            f.write(file_data)  # 파일 데이터를 바이너리로 저장

        logger.info(f"File with signature_id {signature_id} has been saved to
{local_file_path}")

        # 3-1: 파일 PE 섹션 분석
        pe_info = check_Enc.analyze_pe_sections(file_data, is_path=False)

        # 3-2: 섹션 암호화 여부 확인
```

```
58.         encryption_status = check_Enc.check_file_encryption(pe_info)
59.
60.         # 3-3: PE 섹션 정보를 MongoDB 에 저장
61.         uploadDB_NorPE.store_normal_file_pe_info(signature_id, file_doc["filename"],
file_doc["upload_ip"], pe_info, encryption_status)
62.
63.         # 3-4: 파일 암호화 작업 실행
64.         encrypted_file_path = convert_NorToEnc.encrypt_with_themida(local_file_path,
file_doc)
65.         if encrypted_file_path:
66.             logger.info(f"File {local_file_path} has been successfully encrypted.")
67.         else:
68.             logger.error(f"Encryption failed for file {local_file_path}.")
69.
70.         return local_file_path
71.
72.     except Exception as e:
73.         logger.error(f"Error storing file with signature_id {file_doc.get('signature_id',
'unknown')}: {e}", exc_info=True)
74.         print(f"Error storing file with signature_id {file_doc.get('signature_id',
'unknown')}. Check log for details.")
75.         return None
76.
```

목적	DB에 올라온 사용자 파일 다운로드 모듈
파일명	downlaodDB_Nor.py

```python
import subprocess
import os
from logging_Utils import setup_logger
from datetime import datetime
import uploadDB_Enc  # MongoDB 에 암호화된 파일을 저장하는 모듈
import uploadDB_EncPE  # MongoDB 에 암호화된 파일 PE 를 저장하는 모듈
import check_Enc

# 로그 파일 설정
log_file = r'C:\pymodules\log\file_encryption.log'
logger = setup_logger(log_file)

def encrypt_with_themida(input_file, original_file_doc):
    """
    Themida 로 파일 암호화하고, Themida 작업 완료 후 암호화된 파일의 PE 를 분석하여 MongoDB 에 저장
    :param input_file: 원본 파일 경로
    :param original_file_doc: 원본 파일의 메타데이터 (DB 에서 가져온 정보)
    :return: 암호화된 파일 경로 또는 None
    """
    try:
        # 암호화된 파일을 저장할 경로 설정
        today_date = datetime.today().strftime('%Y-%m-%d')
        protected_folder_path = os.path.join(r'C:\DBFiles', today_date, 'protected')

        # 폴더가 존재하지 않으면 생성
        if not os.path.exists(protected_folder_path):
            os.makedirs(protected_folder_path)

        # 암호화된 파일명을 설정
        file_name, file_extension = os.path.splitext(os.path.basename(input_file))
        output_file = os.path.join(protected_folder_path, f"{file_name}_protected{file_extension}")

        logger.info(f"Starting encryption for file: {input_file}")

        # 배치 파일 경로 (환경에 맞게 수정)
        bat_file_path = r"C:\pymodules\bat\themida_encrypt.bat"

        # 배치 파일 실행, stdout 과 stderr 를 로그로 기록
        result = subprocess.run([bat_file_path, input_file, output_file], stdout=subprocess.PIPE, stderr=subprocess.PIPE, shell=True)

        # 배치 파일 실행 후 결과 확인
        if result.returncode == 0:
            logger.info(f"File {input_file} has been encrypted successfully to {output_file}")
            logger.info(f"Batch file output: {result.stdout.decode()}")  # 배치 파일 출력 로그

            # 암호화가 완료된 파일을 MongoDB 에 저장
            uploadDB_Enc.store_encrypted_file(output_file, original_file_doc)  # MongoDB 에 저장
            logger.info(f"Encrypted file {output_file} has been saved to MongoDB.")

```

```
            # Themida 암호화 작업이 완료된 후, PE 섹션 분석 진행
            analyze_and_store_pe(output_file, original_file_doc)

            return output_file
        else:
            logger.error(f"Error in Themida encryption for {input_file}: Return code {result.returncode}")
            logger.error(f"Error message: {result.stderr.decode()}")  # 배치 파일 오류 로그
            return None

    except Exception as e:
        logger.error(f"Error during encryption for {input_file}: {e}", exc_info=True)
        return None

def analyze_and_store_pe(encrypted_file, original_file_doc):
    """
    Themida 암호화 작업이 끝난 후 암호화된 파일의 PE 를 분석하고, MongoDB 에 저장
    :param encrypted_file: 암호화된 파일 경로
    :param original_file_doc: 원본 파일의 메타데이터 (DB 에서 가져온 정보)
    """
    try:
        # 1: 파일 PE 섹션 분석
        pe_info = check_Enc.analyze_pe_sections(encrypted_file, is_path=True)

        # 2: 섹션 암호화 여부 확인
        encryption_status = check_Enc.check_file_encryption(pe_info)

        # 3: MongoDB 에 PE 정보와 암호화 여부 저장
        signature_id = original_file_doc["signature_id"]
        uploadDB_EncPE.store_encrypted_file_pe_info(signature_id, encrypted_file, original_file_doc["upload_ip"], pe_info, encryption_status)

        logger.info(f"PE information and encryption status for encrypted file {encrypted_file} successfully stored in MongoDB.")

    except Exception as e:
        logger.error(f"Error during PE analysis for encrypted file {encrypted_file}: {e}", exc_info=True)
```

목적	사용자가 올린 파일을 프로텍터로 암호화 모듈
파일명	conver_NorToEnc.py

```python
import os
from pymongo import MongoClient
import gridfs
from logging_Utils import setup_logger
from datetime import datetime
from dotenv import load_dotenv

load_dotenv()
#env 에서 로드
MONGO_URI=os.getenv("MONGO_URI")
# 로그 설정
log_file = r'C:\pymodules\log\file_encryption.log'
logger = setup_logger(log_file)

# MongoDB 연결 설정
client = MongoClient(MONGO_URI)
db = client['encrypted_files']  # encrypted_files DB 사용
fs = gridfs.GridFS(db)  # GridFS 를 이용해 파일을 저장

def store_encrypted_file(encrypted_file_path, original_file_doc):
    """
    Themida 로 보호된 파일을 GridFS 에 저장하고, 관련 메타데이터를 MongoDB 의 encrypted_files.filedata 컬렉션에 저장.

    :param encrypted_file_path: 보호된 파일 경로
    :param original_file_doc: 원본 파일의 메타데이터 (MongoDB 에서 가져온 정보)
    """
    try:
        # 암호화된 파일을 읽어서 MongoDB 에 저장
        with open(encrypted_file_path, "rb") as f:
            encrypted_file_data = f.read()

        # GridFS 에 암호화된 파일 저장
        encrypted_file_id = fs.put(
            encrypted_file_data,
            filename=os.path.basename(encrypted_file_path),
            original_filename=original_file_doc["filename"],
            original_signature_id=original_file_doc["signature_id"],
            file_extension=os.path.splitext(encrypted_file_path)[1],
            encrypted=True  # 파일이 암호화되었음을 표시
        )

        # 메타데이터 저장: 암호화된 파일 정보
        encrypted_file_doc = {
            "signature_id": original_file_doc["signature_id"],  # 원본 파일의 signature_id 유지
            "original_filename": original_file_doc["filename"],  # 원본 파일명
            "encrypted_filename": os.path.basename(encrypted_file_path),  # 암호화된 파일명
            "original_upload_time": original_file_doc["upload_time"],  # 원본 파일 업로드 시간
            "encrypted_upload_time": datetime.now(),  # 암호화된 파일 저장 시간
            "upload_ip": original_file_doc["upload_ip"],  # 업로드한 IP 정보
```

```
            "gridfs_file_id": encrypted_file_id  # GridFS 에 저장된 파일의 ID
        }

        # encrypted_files 의 filedata 컬렉션에 메타데이터 저장
        db.filedata.insert_one(encrypted_file_doc)

        logger.info(f"Encrypted file {encrypted_file_path} has been stored in MongoDB successfully.")
        return True

    except Exception as e:
        logger.error(f"Error storing encrypted file {encrypted_file_path} in MongoDB: {e}", exc_info=True)
        return False

```

목적	프로텍터로 암호화된 파일 DB에 업로드 모듈
파일명	uploadDB_Enc.py

```python
from pefile import PE
from logging_Utils import setup_logger  # logging_Utils 에서 로거 설정 함수 가져오기
import hashlib

# 로그 파일 설정
log_file = r'C:\pymodules\log\check_enc.log'
logger = setup_logger(log_file)  # logging_utils 의 setup_logger 사용

def analyze_pe_sections(file_input, is_path=False):
    """
    PE 파일의 섹션을 분석하고, 필요한 섹션 정보를 추출
    :param file_input: 파일 경로 또는 파일 데이터
    :param is_path: True 면 파일 경로로 처리, False 면 파일 데이터로 처리
    :return: 섹션 정보가 담긴 딕셔너리
    """
    try:
        # 파일 경로로 분석할지, 파일 데이터로 분석할지 결정
        if is_path:
            pe = PE(file_input)  # 파일 경로를 사용한 경우
        else:
            pe = PE(data=file_input)  # 파일 데이터를 사용한 경우

        sections = {}

        for section in pe.sections:
            section_name = section.Name.decode('utf-8').rstrip('\x00')

            # 섹션의 MD5 해시 계산
            section_data = section.get_data()
            section_md5 = hashlib.md5(section_data).hexdigest()

            # 추가적인 섹션 정보 추출
            sections[section_name] = {
                "virtual_address": section.VirtualAddress,  # 메모리 상의 가상 주소
                "virtual_size": section.Misc_VirtualSize,  # 메모리 로드 시 크기
                "raw_size": section.SizeOfRawData,  # 실제 파일에서의 크기
                "entropy": section.get_entropy(),  # 엔트로피
                "md5_hash": section_md5,  # 섹션의 MD5 해시
                "characteristics": section.Characteristics,  # 섹션의 권한 정보 (읽기/쓰기/실행)
                "permissions": {
                    "readable": bool(section.Characteristics & 0x40000000),
                    "writable": bool(section.Characteristics & 0x80000000),
                    "executable": bool(section.Characteristics & 0x20000000),
                },
                # PE 값 추가
                "pointer_to_raw_data": section.PointerToRawData,  # 파일 내에서 섹션의 시작 위치
                "section_alignment": pe.OPTIONAL_HEADER.SectionAlignment  # 섹션의 메모리 정렬 값
            }

```

```
        logger.info("PE section analysis completed successfully.")
        return sections

    except Exception as e:
        logger.error(f"Error during PE section analysis: {e}", exc_info=True)
        return {}

# 섹션이 암호화되었는지 확인하는 함수
def is_section_encrypted(section):
    entropy_threshold = 7.0  # 예시 임계값 (일반적으로 7.0 이상이 암호화된 섹션으로 간주됨)
    return section['entropy'] > entropy_threshold

# 파일의 섹션 암호화 상태를 확인하는 함수
def check_file_encryption(pe_info):
    encryption_status = {}

    try:
        for section_name, section_data in pe_info.items():
            encrypted = is_section_encrypted(section_data)
            encryption_status[section_name] = encrypted
            logger.info(f"Section {section_name}: {'Encrypted' if encrypted else 'Not
Encrypted'} (Entropy: {section_data['entropy']})")

        logger.info("File encryption status check completed successfully.")
        return encryption_status

    except Exception as e:
        logger.error(f"Error during file encryption status check: {e}", exc_info=True)
        return {}

```

목적	파일의 PE 분석 및 암호화 확인 모듈
파일명	check_Enc.py

```python
from pymongo import MongoClient
import datetime
from logging_Utils import setup_logger  # 로깅 설정 함수 가져오기
from dotenv import load_dotenv
import os

load_dotenv()
#env 에서 로드
MONGO_URI=os.getenv("MONGO_URI")
# 로그 파일 설정
log_file = r'C:\pymodules\log\upload_db_norpe.log'
logger = setup_logger(log_file)

# 일반 파일의 PE 정보를 MongoDB 에 저장하는 함수
def store_normal_file_pe_info(signature_id, filename, upload_ip, pe_info, encryption_status):
    try:
        # MongoDB 연결
        client = MongoClient(MONGO_URI)
        db = client['normal_files']
        normal_file_pe_info_collection = db['pe_info']

        # 업로드 시간을 현재 시각으로 설정
        upload_time = datetime.datetime.now()

        # MongoDB 에 저장할 PE 데이터 생성
        pe_data = {
            "signature_id": signature_id,
            "filename": filename,
            "upload_time": upload_time,
            "upload_ip": upload_ip,
            "pe_info": pe_info,
            "encrypted": encryption_status
        }

        # MongoDB 에 데이터 삽입
        normal_file_pe_info_collection.insert_one(pe_data)
        logger.info(f"PE information for file {filename} (signature_id: {signature_id}) successfully stored in MongoDB.")

    except Exception as e:
        logger.error(f"Error storing PE information for file {filename} (signature_id: {signature_id}): {e}", exc_info=True)

```

목적	사용자가 업로드 한 파일의 PE를 DB에 업로드 하는 모듈
파일명	uploadDB_NorPE.py

```python
from pymongo import MongoClient
import datetime
from logging_Utils import setup_logger  # 로깅 설정 함수 가져오기
from dotenv import load_dotenv
import os

load_dotenv()
#env 에서 로드
MONGO_URI=os.getenv("MONGO_URI")
# 로그 파일 설정
log_file = r'C:\pymodules\log\upload_db_encpe.log'
logger = setup_logger(log_file)

# 암호화된 파일의 PE 정보를 MongoDB 에 저장하는 함수
def store_encrypted_file_pe_info(signature_id, encrypted_filename, upload_ip, pe_info, encryption_status):
    try:
        # MongoDB 연결 설정
        client = MongoClient(MONGO_URI)
        db = client['encrypted_files']
        encrypted_file_pe_info_collection = db['pe_info']

        # 업로드 시간을 현재 시간으로 설정
        upload_time = datetime.datetime.now()

        # MongoDB 에 저장할 PE 데이터 생성
        pe_data = {
            "signature_id": signature_id,
            "encrypted_filename": encrypted_filename,
            "encrypted_upload_time": upload_time,
            "upload_ip": upload_ip,
            "pe_info": pe_info,  # PE 정보 저장
            "encrypted": encryption_status  # 섹션 암호화 여부 저장
        }

        # MongoDB 에 데이터 삽입
        encrypted_file_pe_info_collection.insert_one(pe_data)
        logger.info(f"PE information and encryption status for encrypted file {encrypted_filename} (signature_id: {signature_id}) successfully stored in MongoDB.")

    except Exception as e:
        logger.error(f"Error storing PE information for encrypted file {encrypted_filename} (signature_id: {signature_id}): {e}", exc_info=True)

```

목적	프로텍터로 암호화된 파일 PE를 DB에 업로드 하는 모듈
파일명	uploadDB_EncPE.py

```python
import logging
import os

def setup_logger(log_file_path):
    """로그 파일 경로를 받아 로거를 설정"""
    logger = logging.getLogger(log_file_path)
    logger.setLevel(logging.INFO)

    # 중복 로그 방지
    if not logger.hasHandlers():
        # 로그 핸들러 설정
        file_handler = logging.FileHandler(log_file_path)
        formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
        file_handler.setFormatter(formatter)
        logger.addHandler(file_handler)

    return logger

```

목적	각 모듈 로그 관리하는 모듈
파일명	logging_Utils.py

```
@echo off
echo Starting Themida encryption...
set input_file=%1
set output_file=%2

REM sigcheck.exe 로 파일 아키텍처 확인
C:\pymodules\sigcheck\sigcheck64.exe -a %input_file% > temp_arch.txt

REM temp_arch.txt 에서 아키텍처 정보를 읽어옴
findstr /i "32-bit" temp_arch.txt > nul
if %errorlevel% == 0 (
    echo 32 비트 파일입니다. Themida 32 비트 버전으로 보호 작업을 실행합니다.
    "C:\Program Files (x86)\Themida Full Activated\Themida.exe" /protect "C:\Program Files (x86)\Themida Full Activated\Prevision.tmd" /inputfile %input_file% /outputfile %output_file%
    goto done
)

findstr /i "64-bit" temp_arch.txt > nul
if %errorlevel% == 0 (
    echo 64 비트 파일입니다. Themida 64 비트 버전으로 보호 작업을 실행합니다.
    "C:\Program Files (x86)\Themida Full Activated\Themida64.exe" /protect "C:\Program Files (x86)\Themida Full Activated\Prevision64.tmd" /inputfile %input_file% /outputfile %output_file%
    goto done
)

echo 파일 아키텍처를 확인할 수 없습니다. sigcheck 결과를 확인하세요.

:done
REM sigcheck 프로세스 종료 후 임시 파일 삭제

del temp_arch.txt
echo 작업 완료.

```

목적	Themida Protector 암호화 작동 배치 파일
파일명	themida_encrypt.bat

```python
from pymongo import MongoClient
from datetime import datetime
import os
import gridfs
from dotenv import load_dotenv

# 환경 변수 로드
load_dotenv()
MONGO_URI = os.getenv("MONGO_URI")

# MongoDB 연결
client = MongoClient(MONGO_URI)

# 데이터베이스 및 GridFS 설정
db = client['normal_files']
fs = gridfs.GridFS(db)  # GridFS 인스턴스 생성
collection = db['filedata']

# signature_id 생성 함수
def generate_signature_id():
    # 오늘의 날짜 (YYYYMMDD 형식)
    today = datetime.now().strftime("%Y%m%d")

    # 오늘 날짜로 시작하는 signature_id의 개수를 확인하여 번호를 매김
    count = collection.count_documents({"signature_id": {"$regex": f"^{today}-"}})
    next_id = count + 1

    # signature_id를 YYYYMMDD-번호 형식으로 생성
    signature_id = f"{today}-{next_id:03d}"
    return signature_id

# 파일 경로 지정
file_path = r'C:\Users\Administrator\Desktop\sample_data\Bandizip.exe'

# 파일명과 확장자 추출
filename = os.path.basename(file_path)  # 파일명 추출
file_extension = os.path.splitext(filename)[1]  # 확장자 추출 (예: .exe)

# 파일을 읽어서 바이너리 데이터로 변환
with open(file_path, 'rb') as file:
    file_data = file.read()

# signature_id 생성
signature_id = generate_signature_id()

# GridFS에 파일 저장
gridfs_file_id = fs.put(file_data, filename=filename)

# 파일 정보를 MongoDB의 메타데이터 컬렉션에 삽입
file_document = {
    "signature_id": signature_id,
    "filename": filename,
    "file_extension": file_extension,  # 확장자 추가
    "gridfs_file_id": gridfs_file_id,  # GridFS 파일 ID
    "upload_time": datetime.now(),
    "upload_ip": "192.168.0.1"  # 적절한 IP를 입력하세요
```

```
57. }
58.
59. # 메타데이터를 MongoDB 컬렉션에 삽입
60. collection.insert_one(file_document)
61.
62. print(f"파일이 GridFS 와 MongoDB 메타데이터 컬렉션에 성공적으로 업로드되었습니다. Signature ID: {signature_id}")
63.
```

목적	모듈 테스트를 위해 DB에 샘플 파일 업로드 하는 모듈
파일명	uploadDB_Sam.py

```python
import os
import gridfs
from pymongo import MongoClient
from bson import ObjectId
from logging_Utils import setup_logger
from dotenv import load_dotenv

load_dotenv()
#env 에서 로드
MONGO_URI=os.getenv("MONGO_URI")
# 로그 설정
log_file = r'C:\pymodules\log\file_retrieval.log'
logger = setup_logger(log_file)

# MongoDB 연결 설정
client = MongoClient(MONGO_URI)
db = client['encrypted_files']  # 'encrypted_files' 데이터베이스 사용
fs = gridfs.GridFS(db)  # GridFS 객체 생성

def retrieve_encrypted_file(file_id):
    """
    GridFS 에서 파일 ID 를 사용해 파일을 다운로드합니다.
    :param file_id: GridFS 에 저장된 파일의 ID (ObjectId)
    :return: 파일 데이터 (바이너리) 또는 None
    """
    try:
        # GridFS 에서 파일 ID 를 이용해 파일을 가져옴
        file_data = fs.get(file_id).read()  # 파일 데이터를 읽음
        logger.info(f"File with ID {file_id} has been retrieved from MongoDB.")
        return file_data
    except Exception as e:
        logger.error(f"Error retrieving file with ID {file_id}: {e}", exc_info=True)
        return None

def save_file_to_local(file_data, save_path):
    """
    파일 데이터를 로컬에 저장합니다.
    :param file_data: 가져온 파일의 바이너리 데이터
    :param save_path: 파일을 저장할 경로
    """
    try:
        # 로컬 경로에 파일 저장
        with open(save_path, 'wb') as f:
            f.write(file_data)
        logger.info(f"File has been saved to {save_path}")
        print(f"File saved successfully to {save_path}")
    except Exception as e:
        logger.error(f"Error saving file to {save_path}: {e}", exc_info=True)

if __name__ == "__main__":
    # 사용자로부터 파일 ID 입력 받음 (실제 MongoDB GridFS 에 저장된 파일의 ObjectId)
    file_id_str = input("Enter the GridFS file ID (ObjectId format): ")

    try:
        # 입력된 문자열을 ObjectId 로 변환
        file_id = ObjectId(file_id_str)
```

```
    except Exception as e:
        print(f"Invalid ObjectId format: {e}")
        logger.error(f"Invalid ObjectId format: {e}")
        exit(1)

    # GridFS 에서 파일 가져오기
    file_data = retrieve_encrypted_file(file_id)

    if file_data:
        # 가져온 파일 데이터를 로컬 파일로 저장
        save_path = input("Enter the path where you want to save the file: ")
        save_file_to_local(file_data, save_path)
    else:
        print("Failed to retrieve the file from MongoDB.")

```

목적	모듈 테스트를 위해 DB에서 암호화된 파일 다운로드 하는 모듈
파일명	downloadDB_Enc.py