악성코드 탐지, 암호화 및 패키징

악성코드 탐지 개발 명세서

문서번호 : DS-101

VER1.1

```python
import time
import json
import re
from hash_handler import read_hashes_from_csv, load_processed_hashes,
save_processed_hashes
from virustotal_API import search_file_by_hash
from data_converter import convert_data
from DB_handler import upload_to_mongodb
from logger_manager import setup_logger

# 로그 설정
logger = setup_logger(r'C:\VTAPImodules\log\process_hash',
r'C:\VTAPImodules\log\process_hash.log')

# 설정
MAX_EXECUTIONS_PER_MINUTE = 2  # 분당 최대 2 번 실행
MAX_EXECUTIONS_PER_DAY = 250  # 하루 최대 250 번 실행

# 분당 5 번 실행을 위한 제한 함수
def rate_limiter():
    time.sleep(30)  # 30 초마다 1 번 실행 -> 분당 2 번 실행 가능
    logger.info("Rate limiter: 30 초 대기")

# MD5 해시가 유효한지 확인하는 함수
def is_valid_md5(md5_hash):
    """MD5 해시가 32 자리의 16 진수인지 확인"""
    valid = re.match(r"^[a-fA-F0-9]{32}$", md5_hash) is not None
    if not valid:
        logger.warning(f"유효하지 않은 MD5 해시: {md5_hash}")
    return valid

def process_hash(hash_value, processed_hashes):
    if hash_value in processed_hashes:
        logger.info(f"{hash_value} 이미 처리됨. 스킵합니다.")
        return False  # 이미 처리된 경우 False 반환

    # 유효한 MD5 해시인지 확인
    if not is_valid_md5(hash_value):
        logger.warning(f"{hash_value} 유효하지 않은 MD5 해시입니다. 스킵합니다.")
        return False  # 유효하지 않은 해시는 처리하지 않음

    # VirusTotal API 호출
    details = search_file_by_hash(hash_value)
    if details is None:
        logger.error(f"{hash_value} 처리 실패. 파일을 찾을 수 없습니다.")
        return False

    behavior = search_file_by_hash(hash_value, "behaviour_summary")
    if behavior is None:
        logger.warning(f"{hash_value} 행동 분석 정보를 찾을 수 없습니다.")

    if details:
        # 데이터 변환
        logger.info(f"{hash_value} 데이터 변환 중...")
```

```
        converted_data = convert_data(details, behavior)

        # MongoDB 에 업로드
        upload_to_mongodb(converted_data, "info")
        logger.info(f"{hash_value} MongoDB 에 저장 완료.")

        # 처리된 해시 기록
        processed_hashes.append(hash_value)
        save_processed_hashes(processed_hashes)
        logger.info(f"{hash_value} 처리된 해시 기록에 추가.")
        return True  # 성공적으로 처리된 경우 True 반환
    else:
        logger.error(f"{hash_value} 처리 실패.")
        return False  # 처리 실패 시 False 반환

    rate_limiter()

if __name__ == "__main__":
    logger.info("처리된 해시 로드 시작")

    # 처리된 해시 로드
    processed_hashes = load_processed_hashes()

    # 해시값 처리
    execution_count = 0
    hashes = read_hashes_from_csv()

    for hash_value in hashes:
        if execution_count >= MAX_EXECUTIONS_PER_DAY:
            logger.info("오늘의 최대 실행 횟수에 도달했습니다.")
            break

        if process_hash(hash_value, processed_hashes):
            execution_count += 1  # 실제로 처리된 경우에만 증가

```

목적	데이터셋에서 Virustoatal API를 통해 나온 정보로 DB에 업로드 하기 위한 모듈
파일명	process_hash.py

```python
import json
from logger_manager import setup_logger
import csv

# 로그 설정
logger = setup_logger(r'C:\VTAPImodules\log\hash_handler',
r'C:\VTAPImodules\log\hash_handler.log')

PROCESSED_HASHES_FILE = r'C:\VTAPImodules\processed_hashes.json'

def read_hashes_from_csv():
    """CSV 파일에서 해시 값을 읽어온다."""
    csv_file = r"C:\VTAPImodules\dataset.csv"
    try:
        with open(csv_file, 'r', encoding='ISO-8859-1') as file:
            reader = csv.DictReader(file)
            hashes = [row['md5_hash'].strip() for row in reader]  # CSV 에서 'md5_hash'
열 읽기
            logger.info(f"CSV 파일 {csv_file}에서 해시 값 로드 성공")
            return hashes
    except FileNotFoundError:
        logger.error(f"CSV 파일 {csv_file}을 찾을 수 없음")
        return []
    except Exception as e:
        logger.error(f"CSV 파일 읽기 중 오류 발생: {e}")
        return []

def load_processed_hashes():
    try:
        with open(PROCESSED_HASHES_FILE, 'r') as f:
            logger.info("처리된 해시 로드 성공")
            return json.load(f)
    except FileNotFoundError:
        logger.error("처리된 해시 파일을 찾을 수 없음")
        return []
    except json.JSONDecodeError:
        logger.error("처리된 해시 파일을 읽는 중 JSON 오류 발생")
        return []

def save_processed_hashes(processed_hashes):
    try:
        with open(PROCESSED_HASHES_FILE, 'w') as f:
            json.dump(processed_hashes, f)
            logger.info("처리된 해시 기록 저장 성공")
    except Exception as e:
        logger.error(f"처리된 해시 저장 중 오류 발생: {e}")

```

목적	csv와 json에서 해시를 읽거나 저장하는 모듈
파일명	hash_handler.py

```python
import time
from pymongo import MongoClient
import os
from dotenv import load_dotenv
import requests
import json
from logger_manager import setup_logger
import gridfs

# 로그 설정
logger = setup_logger(r'C:\VTAPImodules\log\DB_handler',
r'C:\VTAPImodules\log\DB_handler.log')

load_dotenv()
MONGO_URI = os.getenv('MONGO_URI')
DB_NAME = "vsapi"
FILES_COLLECTION = "file"
INFO_COLLECTION = "info"
client = MongoClient(MONGO_URI)
db = client[DB_NAME]
fs = gridfs.GridFS(db)

def watch_for_file_uploads():
    collection = db[FILES_COLLECTION]
    logger.info("파일 업로드 감시 시작")
    with collection.watch() as stream:
        for change in stream:
            if change["operationType"] == "insert":
                file_data = change["fullDocument"]
                file_hash = file_data.get('filehash')  # MD5 해시 가져오기
                gridfs_file_id = file_data.get('gridfs_file_id')  # GridFS 파일 ID 가져오기

                if file_hash and gridfs_file_id:
                    logger.info(f"새로운 파일 업로드 감지: {file_hash}, GridFS 파일 ID:
{gridfs_file_id}")

                    # GridFS 에서 파일 가져오기 (gridfs_file_id 를 사용하여 파일 검색)
                    try:
                        grid_out = fs.get(gridfs_file_id)
                        logger.info(f"GridFS 에서 파일 {file_hash} 로드 완료")
                        return file_hash, grid_out.read()  # 파일의 바이너리 데이터 반환
                    except gridfs.errors.NoFile:
                        logger.error(f"GridFS 에 {gridfs_file_id}에 해당하는 파일이 없습니다.
(file_hash: {file_hash})")
                        return None, None
                else:
                    logger.error(f"file_hash 또는 gridfs_file_id 가 존재하지 않습니다.
file_hash: {file_hash}, gridfs_file_id: {gridfs_file_id}")
                    return None, None



# MongoDB 업로드
def upload_to_mongodb(data, collection_name=FILES_COLLECTION):
    collection = db[collection_name]
    try:
        collection.insert_one(data)
        logger.info(f"Data uploaded to MongoDB: {collection_name}")
    except Exception as e:
        logger.error(f"MongoDB 업로드 실패: {e}")

```

```python
# 해시가 MongoDB 'info' 컬렉션에 있는지 확인하는 함수
def check_hash_in_mongodb(file_hash):
    collection = db[INFO_COLLECTION]
    try:
        existing_data = collection.find_one({"md5": file_hash})
        if existing_data:
            logger.info(f"{file_hash} 이미 MongoDB 'info' 컬렉션에 존재합니다.")
            return existing_data
        else:
            logger.info(f"{file_hash} MongoDB 'info' 컬렉션에 존재하지 않음.")
    except Exception as e:
        logger.error(f"MongoDB 해시 검사 실패: {e}")
    return None

```

목적	DB에 올라온 파일을 감시, DB에 업로드, DB에 해당 해시 값이 있는지 확인하는 모듈
파일명	DB_handler.py

```python
import json
import os
from logger_manager import setup_logger

# 로그 설정
logger = setup_logger(r'C:\VTAPImodules\log\data_converter', r'C:\VTAPImodules\log\data_converter.log')

def convert_data(details, behavior):
    logger.info("데이터 변환 시작")

    try:
        template_file_path = r"C:\VTAPImodules\template.json"
        with open(template_file_path, 'r', encoding='utf-8') as template_file:
            json_data = json.load(template_file)

        detail_data_attribute = details.get("data", {}).get("attributes", {})
        json_data["md5"] = detail_data_attribute.get("md5", None)

        # 해시 정보
        json_data["details"]["hash"]["md5"] = detail_data_attribute.get("md5", None)
        json_data["details"]["hash"]["sha1"] = detail_data_attribute.get("sha1", None)
        json_data["details"]["hash"]["sha256"] = detail_data_attribute.get("sha256", None)
        json_data["details"]["hash"]["vhash"] = detail_data_attribute.get("vhash", None)
        json_data["details"]["hash"]["auth_hash"] = detail_data_attribute.get("authentihash", None)
        json_data["details"]["hash"]["imphash"] = detail_data_attribute.get("pe_info", {}).get("imphash", None)
        json_data["details"]["hash"]["ssdeep"] = detail_data_attribute.get("ssdeep", None)
        json_data["details"]["hash"]["tlsh"] = detail_data_attribute.get("tlsh", None)

        # 파일 정보
        json_data["details"]["file_info"]["md5"] = detail_data_attribute.get("md5", None)
        json_data["details"]["file_info"]["file_type"] = detail_data_attribute.get("type_tags", None)
        json_data["details"]["file_info"]["magic"] = detail_data_attribute.get("magic", None)
        json_data["details"]["file_info"]["file_size"] = detail_data_attribute.get("size", None)
        json_data["details"]["file_info"]["PEID_packer"] = detail_data_attribute.get("packers", {}).get("PEiD", None)
        json_data["details"]["file_info"]["first_seen_time"] = detail_data_attribute.get("first_submission_date", None)

        # 이름 정보
        json_data["details"]["file_info"]["name"] = detail_data_attribute.get("names", None)

        # 시그니처 정보
        json_data["details"]["signature"] = detail_data_attribute.get("signature_info", {})

        json_data["details"]["pe_info"] = detail_data_attribute.get("pe_info", {})
        json_data["details"]["dot_net_assembly"] = detail_data_attribute.get("dot_net_assembly", {})
```

```
45.
46.         if behavior is None or behavior.get("data") is None:
47.             logger.info("Behavior 데이터가 없음")
48.             return json_data
49.
50.         # MITRE 공격 기법 정보 추가
51.         mitre_techniques = behavior.get("data", {}).get("mitre_attack_techniques",
{})
52.         for technique in mitre_techniques:
53.             technique_id = technique.get("id")
54.             description = technique.get("signature_description")
55.             severity = technique.get("severity", "")
56.
57.             json_data["behavior"]["mitre"][technique_id] = {
58.                 "description": description,
59.                 "severity": severity
60.             }
61.
62.         # 행동 정보 추가
63.         json_data["behavior"]["modules_loaded"] = behavior.get("data",
{}).get("modules_loaded", {})
64.         json_data["behavior"]["tags"] = behavior.get("data", {}).get("tags", {})
65.
66.         # Capabilities 처리
67.         capabilities_comms = behavior.get("data", {}).get("signature_matches", {})
68.         for comm in capabilities_comms:
69.             if comm.get("format") == "SIG_FORMAT_CAPA":
70.                 capa_name = comm.get("name")
71.                 description = comm.get("description")
72.                 authors = comm.get("authors")
73.                 rule_src = comm.get("rule_src")
74.                 refs = comm.get("refs", {})
75.                 json_data["behavior"]["Capabilities"][capa_name] = {
76.                     "authors": authors,
77.                     "description": description,
78.                     "rule": rule_src,
79.                     "refs": refs
80.                 }
81.
82.         # 네트워크 통신 정보
83.         network_communications = ["ja3_digests", "http_conversations",
"memory_pattern_domains", "memory_pattern_urls",
84.                                   "memory_pattern_ips", "tls"]
85.         for net_comm in network_communications:
86.             col = behavior.get("data", {}).get(net_comm, {})
87.             json_data["behavior"]["network_communications"][net_comm] = col
88.
89.         # 파일 작업 정보
90.         file_actions = ["files_opened", "files_written", "files_deleted",
"files_attribute_changed", "files_dropped"]
91.         for file_comm in file_actions:
92.             col = behavior.get("data", {}).get(file_comm, {})
93.             json_data["behavior"]["file_system_actions"][file_comm] = col
94.
95.         # 레지스트리 작업 정보
96.         registry_actions = ["registry_keys_opened", "registry_keys_set",
"registry_keys_deleted"]
97.         for reg_comm in registry_actions:
98.             col = behavior.get("data", {}).get(reg_comm, {})
99.             json_data["behavior"]["registry_actions"][reg_comm] = col
100.
```

```
        # 프로세스 및 서비스 작업 정보
        process_and_service_actions = ["processes_created", "command_executions",
"processes_injected",
                                       "processes_terminated", "services_opened",
"processes_tree"]
        for pas_comm in process_and_service_actions:
            col = behavior.get("data", {}).get(pas_comm, {})
            json_data["behavior"]["process_and_service_actions"][pas_comm] = col

        # 동기화 메커니즘 정보
        synchronization_mechanisms_signals = ["mutexes_created", "mutexes_opened"]
        for sms_comm in synchronization_mechanisms_signals:
            col = behavior.get("data", {}).get(sms_comm, {})
            json_data["behavior"]["synchronization_mechanisms_signals"][sms_comm] =
col

        # 강조된 작업 정보
        highlighted_actions = ["calls_highlighted", "text_decoded"]
        for high_comm in highlighted_actions:
            col = behavior.get("data", {}).get(high_comm, {})
            json_data["behavior"]["highlighted_actions"][high_comm] = col

        # 시스템 속성 조회
        system_property_lookups = behavior.get("data",
{}).get("system_property_lookups", {})
        json_data["behavior"]["system_property_lookups"] = system_property_lookups

        logger.info("데이터 변환 완료")
        return json_data

    except Exception as e:
        logger.error(f"데이터 변환 중 오류 발생: {e}")
        return None
```

목적	VirusTotal API를 돌려나온 Details, Behavior를 template에 맞게 변환하는 모듈
파일명	data_converter.py

```python
import requests
import time
import os
from dotenv import load_dotenv
from logger_manager import setup_logger

# 로그 설정
logger = setup_logger(r'C:\VTAPImodules\log\virustotal_api',
r'C:\VTAPImodules\log\virustotal_api.log')

load_dotenv()
API_KEY = os.getenv('VT_API_KEY')

# VirusTotal 에서 해시로 검색하는 함수 (재시도 포함)
def search_file_by_hash_with_retry(file_hash, retries=5, wait_time=60):
    for i in range(retries):
        logger.info(f"해시 검색 시도 {i + 1}/{retries}: {file_hash}")

        response_json = search_file_by_hash(file_hash)
        if response_json:
            logger.info(f"{file_hash} 해시 검색 성공")
            return response_json
        else:
            logger.info(f"{file_hash} 파일이 아직 처리되지 않았습니다. {wait_time}초 대기
후 다시 시도합니다...")
            time.sleep(wait_time)
    logger.error(f"{file_hash} 최대 재시도 횟수 초과")
    return None

# VirusTotal 해시 검색
def search_file_by_hash(hash_value, endpoint=""):
    url = f"https://www.virustotal.com/api/v3/files/{hash_value}"
    if endpoint:
        url = f"https://www.virustotal.com/api/v3/files/{hash_value}/{endpoint}"

    headers = {"accept": "application/json", "x-apikey": API_KEY}
    response = requests.get(url, headers=headers)

    if response.status_code == 200:  # 성공 시
        logger.info(f"{hash_value} 해시 검색 성공")
        return response.json()
    else:
        logger.error(f"{hash_value} 해시 검색 실패: {response.status_code}")
        return None

# VirusTotal 파일 업로드
def upload_file_to_virustotal(file_data):
    logger.info("VirusTotal 파일 업로드 시작")

    upload_url_response =
requests.get("https://www.virustotal.com/api/v3/files/upload_url", headers={"x-apikey":
API_KEY})

    if upload_url_response.status_code == 200:
        upload_url = upload_url_response.json().get('data', None)

        if upload_url:
            files = {'file': file_data}
```

```
55.             upload_response = requests.post(upload_url, files=files, headers={"x-
apikey": API_KEY})
56.
57.             if upload_response.status_code == 200:
58.                 logger.info("파일 업로드 성공")
59.                 return True
60.             else:
61.                 logger.error(f"파일 업로드 실패: {upload_response.status_code} -
{upload_response.text}")
62.         else:
63.             logger.error("업로드 URL 획득 실패")
64.     else:
65.         logger.error(f"업로드 URL 요청 실패: {upload_url_response.status_code} -
{upload_url_response.text}")
66.
67.     return None
68.
```

목적	VirusTotal API를 활용해 파일을 업로드, Details, Behavior 정보를 불러오는 모듈
파일명	virustotal_api.py

```python
from DB_handler import watch_for_file_uploads, upload_to_mongodb, check_hash_in_mongodb
from virustotal_API import search_file_by_hash_with_retry, upload_file_to_virustotal, search_file_by_hash
from hash_handler import save_processed_hashes, load_processed_hashes
from data_converter import convert_data
from logger_manager import setup_logger
import time
# 로그 설정
logger = setup_logger(r'C:\VTAPImodules\log\process_new_file', r'C:\VTAPImodules\log\process_new_file.log')

# 새 파일이 업로드되었을 때 VirusTotal 에서 처리하는 함수
def process_new_file(file_hash, file_data, processed_hashes):
    logger.info(f"새로운 파일 업로드 감지: {file_hash}")

    # DB 에 hash 가 있는 경우 종료
    if check_hash_in_mongodb(file_hash):
        logger.info(f"{file_hash} 이미 DB 에 존재함. 처리 종료.")
        return

    # VirusTotal 에서 해시 검색
    details = search_file_by_hash(file_hash)
    if not details:
        logger.info(f"{file_hash} VirusTotal 에 없음, 파일 업로드 중...")
        flag = upload_file_to_virustotal(file_data)  # 파일 내용 업로드
        if flag:
            # 파일 해시로 검색 (없으면 반복)
            search_result = search_file_by_hash_with_retry(file_hash)
            if search_result:
                details = search_result
                behavior = search_file_by_hash(file_hash, "behaviour_summary")
    else:
        logger.info(f"{file_hash} 이미 VirusTotal 에 존재함. 데이터 가져오는 중...")
        behavior = search_file_by_hash(file_hash, "behaviour_summary")

    # 데이터 변환 후 DB 에 추가
    if details:
        logger.info(f"{file_hash} 데이터 변환 중...")
        converted_data = convert_data(details, behavior)

        # MongoDB 에 업로드
        upload_to_mongodb(converted_data, 'info')
        # 처리된 해시 기록 추가
        processed_hashes.append(file_hash)
        save_processed_hashes(processed_hashes)
        logger.info(f"{file_hash} 분석 및 행동 분석 완료 후 MongoDB 에 저장.")
    else:
        logger.error(f"{file_hash} 처리 실패.")

if __name__ == "__main__":
    logger.info("파일 업로드 감시 시작")
    filehash, filedata = watch_for_file_uploads()
    processed_hashes = load_processed_hashes()

    if filehash:
        process_new_file(filehash, filedata, processed_hashes)
        filehash = None

```

목적	사용자가 DB에 파일을 업로드하면 해당 파일 정보를 DB에 저장해주는 모듈
파일명	process_new_file.py

```python
import logging
import os

def setup_logger(logger_name, log_file, level=logging.INFO):


    logger = logging.getLogger(logger_name)
    logger.setLevel(level)

    # 파일 핸들러 설정 (UTF-8 인코딩 추가)
    file_handler = logging.FileHandler(log_file, encoding='utf-8')
    file_handler.setFormatter(logging.Formatter('%(asctime)s - %(levelname)s
- %(message)s'))

    if not logger.hasHandlers():
        logger.addHandler(file_handler)

    return logger

```

목적	모듈 동작 로그를 관리하기 위한 모듈
파일명	logger_manager.py

```
{
      "md5":"",
    "details": {
        "hash": {
            "md5": "",
            "sha1": "",
            "sha256": "",
            "vhash": "",
            "auth_hash": "",
            "imphash": "",
            "ssdeep": "",
            "tlsh": ""
        },
        "file_info": {
            "md5": "",
            "file_type": "",
            "magic": "",
            "file_size": "",
            "PEID_packer": "",
            "first_seen_time": "",
            "name": ""
        },
        "signature": {
        },
        "pe_info": {
        },
        "dot_net_assembly": {
        }
    },
    "behavior": {
        "mitre": {

        },
        "Capabilities": {

        },
        "tags": {

        },
        "network_communications": {
            "http_conversations": {},
            "ja3_digests": {},
            "memory_pattern_domains": {},
            "memory_pattern_ips": {},
            "memory_pattern_urls": {},
            "tls": {}
        },
        "file_system_actions": {
            "files_opened": {},
            "files_written": {},
            "files_deleted": {},
            "files_attribute_changed": {},
            "files_dropped": {}
        },
        "registry_actions":{
            "registry_keys_opened": {},
            "registry_keys_set": {},
            "registry_keys_deleted": {}
        },
        "process_and_service_actions": {
            "processes_created": {},
            "command_executions": {},
            "processes_injected": {},
            "processes_terminated": {},
```

```
            "services_opened": {},
            "processes_tree": {}
        },
        "synchronization_mechanisms_signals": {
            "mutexes_created": {},
            "mutexes_opened": {}
        },
        "modules_loaded": {
        },
        "highlighted_actions": {
            "calls_highlighted": {},
            "text_decoded": {}
        },
        "system_property_lookups": {}

    }
}
```

목적	DB에 저장할 구조 template
파일명	template.json

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

# .env 파일에서 MongoDB URI 로드
load_dotenv()
MONGO_URI = os.getenv('MONGO_URI')
DB_NAME = "vsapi"
FILES_COLLECTION = "file"

# MongoDB 연결 및 GridFS 인스턴스 생성
client = MongoClient(MONGO_URI)
db = client[DB_NAME]
fs = gridfs.GridFS(db)  # GridFS 인스턴스 생성
collection = db[FILES_COLLECTION]

# MD5 해시 계산 함수
def calculate_md5(file_path):
    hash_md5 = hashlib.md5()
    with open(file_path, "rb") as f:
        while chunk := f.read(8192):  # 파일을 8192 바이트씩 읽어서 해시 계산
            hash_md5.update(chunk)
    return hash_md5.hexdigest()

# 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 업로드 함수 (GridFS 적용)
def upload_file_to_mongodb(file_path, upload_ip):
    filehash = calculate_md5(file_path)
    filename = os.path.basename(file_path)

    with open(file_path, 'rb') as f:
        file_data = f.read()

    # 고유한 signature_id 생성
    signature_id = generate_signature_id()

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

    # 파일 메타데이터 저장 (GridFS에서 파일 ID 저장)
    file_metadata = {
        "signature_id": signature_id,
        "filehash": filehash,
        "filename": filename,
        "gridfs_file_id": gridfs_file_id,  # GridFS 파일 ID
        "upload_time": datetime.now(),
```

```
        "upload_ip": upload_ip
    }

    # MongoDB 의 file 컬렉션에 메타데이터 저장
    collection.insert_one(file_metadata)
    print(f"파일 {filename}가 MongoDB 에 GridFS 로 업로드되었습니다. MD5: {filehash},
Signature ID: {signature_id}")

# 테스트 파일 경로와 IP
test_file_path = r"C:\Users\Administrator\Desktop\sample_data\Bandizip_protected.exe"
test_upload_ip = "192.168.0.1"

# 파일을 MongoDB 에 업로드
upload_file_to_mongodb(test_file_path, test_upload_ip)

```

목적	모듈 테스트를 위한 DB 파일 업로드 모듈
파일명	uploadDB_sam.py