FIT BOOL

PE구조·이미지·OPCODE 융합을 통한 악성코드 탐지

kisia 한국정보보호산업협회
Korea Information Security Industry Association

강민성 김민수 김서현 이도협 홍태경

FIT BOOL

CONTENTS

kisia 한국정보보호산업협회
Korea Information Security Industry Association

01 WHY 서론

- 악성코드란?
- 악성코드의 발전과 피해현황
- 기존 탐지 기법의 한계

악성코드란?

사용자 컴퓨터에 악의적인 영향을 끼칠 수 있는 모든 소프트웨어의 총칭

바이러스
정상적인 프로그램 및 파일에 기생해 복제되며 사용자의 클릭, 다운로드 등 직접적인 행위가 있을 시 전파되는 악성코드

웜
다른 파일을 감염시키지 않고 독립적으로 실행되며 스스로를 복제해 네트워크를 통해 자동 전파하는 악성코드

트로이 목마
정상적인 소프트웨어로 위장하여 타깃의 직접 설치를 유도해, 시스템에 잠입하여 원격 제어·정보 탈취 등에 악용되는 악성코드

랜섬웨어
감염된 시스템의 파일을 암호화하고 복호화 키 제공 대가로 금전을 요구하는 악성코드로, 최근엔 RaaS, 표적형 공급망 공격 등으로 사업화·조직화

2014. KISA 지식플랫폼, https://www.kisa.or.kr/204/form?postSeq=021162&lang_type=KO&page=
2025/06, SK쉴더스, “트로이목마란? 주요 멀웨어 종류 5가지부터 대응법까지!”,
https://www.skshieldus.com/blog-security/security-trend-idx-55

IT의 발전에 따른 악성코드 전파 경로와 수법의 진화

1990s — 저장매체를 통한 전파

- 플로피·CD 같은 물리적 매체 바이러스가 주류
- 자가복제·부트섹터 변조가 주요 유포 방식

2000s — 인터넷·이메일 시대

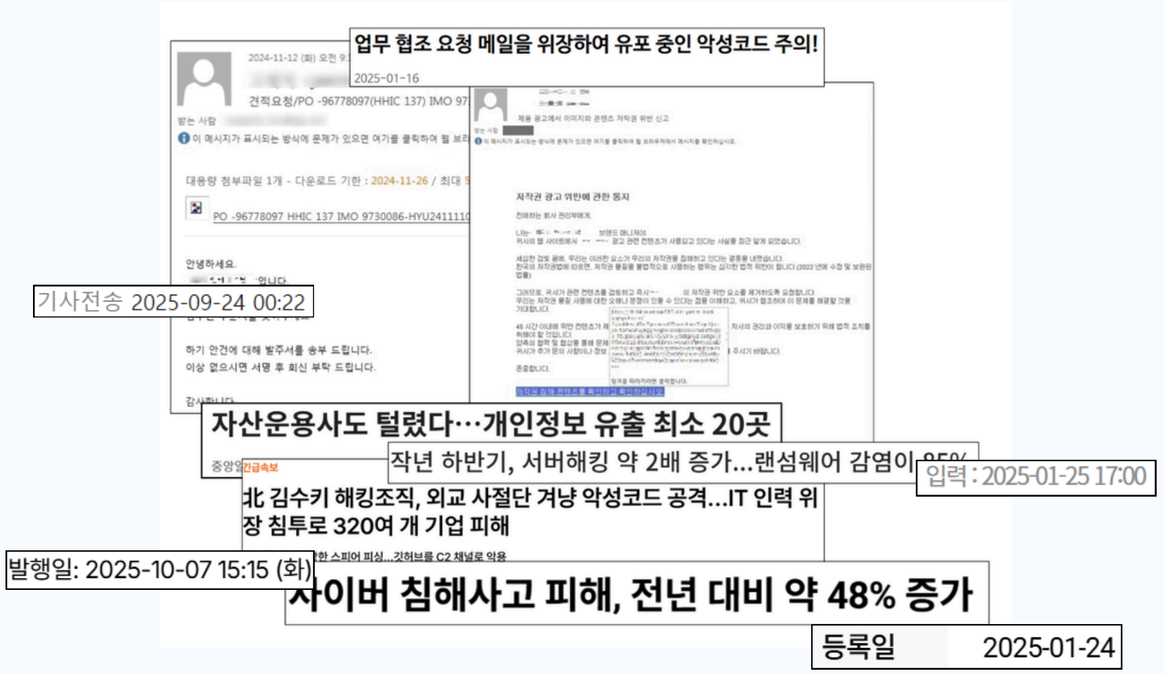
- 초고속 인터넷과 이메일 보급으로 웜·피싱·첨부파일(EXE/DLL)을 통한 대량 유포가 급증
- 현재까지도 유효한 방식

2010s — 웹·서비스·공급망 유통 시대

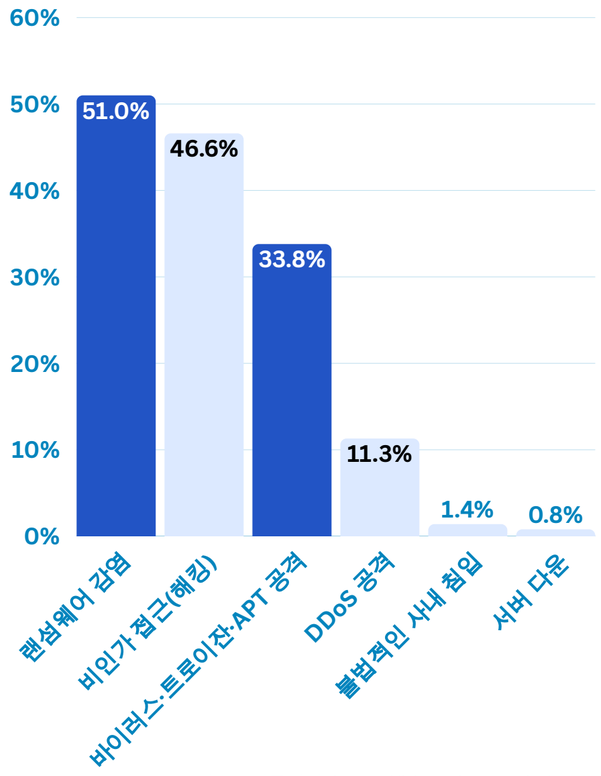
- 드라이브-by 다운로드, 악성 광고·공급망 공격, 클라우드·서비스 취약점 악용
- 자동화된 페이로드 배포와 랜섬웨어가 확산

2020s — 지능화·표적화 시대

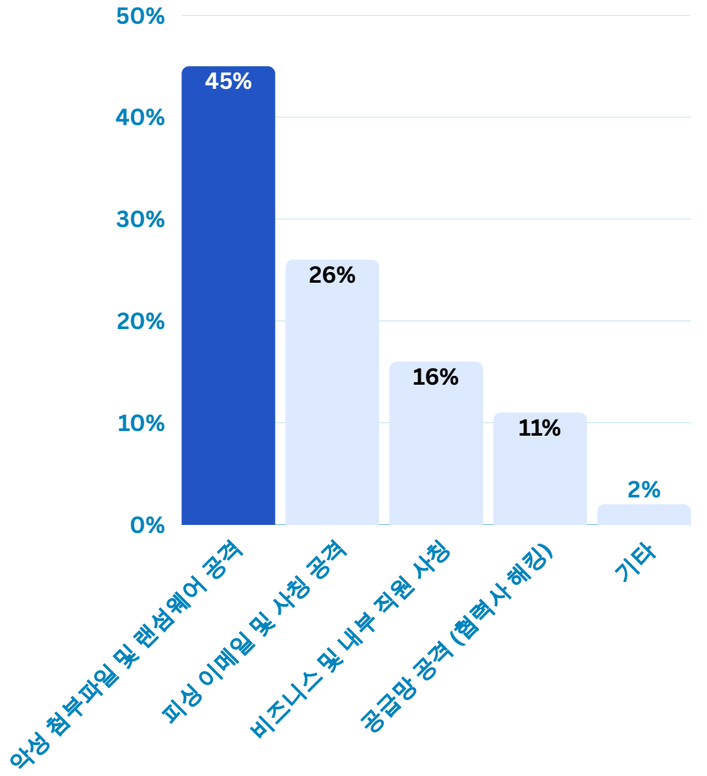
- APT·파일리스·Living-off-the-land, 크립터·폴리모픽 기법
- 자동화·AI 보조가 결합되어 표적형·고도화된 공격이 일반화



2025.08, “북 해킹 조직의 언론사 위장 스피어 피싱 공격 주의!”, 이스트시큐리티 공식 블로그, https://blog.alyac.co.kr/5620
2024.11, “업무 관련 메일로 위장해 유포 중인 악성코드 주의보“, https://m.boannews.com/html/detail.html?idx=134530,
2024.10, “저작권 위반 관련 내용의 피싱 메일을 통해 유포되는 악성코드 주의!”,
https://www.estsecurity.com/enterprise/security-center/notice/view/832456?category-id=5



2025.03, 이스트시큐리티, https://blog.alyac.co.kr/5575



2024.12, “2024 정보보호 실태조사“, 과학기술정보통신부

대규모 위험을 초래하는, 지속적인 악성코드 위협

경험한 침해사고 유형 중
악성코드와 관련된 피해 전체의 약 85%

- 과학기술정보통신부와 한국정보보호산업협회가 500개 기업을 대상으로 행한 '2024년 기업 정보보호실태조사' 결과
- 기업이 경험한 정보 침해사고 유형(중복 응답)은 '랜섬웨어 감염'이 51%, 외부 비인가 침투(해킹) 46.6%, 바이러스로 인한 시스템 마비 33.8% 순
- 기업 정보침해 공격의 주요 공통 경로가 사칭메일

기업이 최근 겪은 이메일 위협 사례 중
악성첨부 파일 및 랜섬웨어 전체의 약 45%

- 이스트시큐리티의 지난 3월 eGISEC 2025에 참가하여 만난 1,969명의 고객분들이 참여한 '국내 기업 보안 현황'에 관한 설문조사 결과
- 이메일은 여전히 대부분의 기업 커뮤니케이션의 핵심 채널이지만, 동시에 가장 많이 악용되는 공격 수단
- 특히 공격자의 사회공학적 기법, 실제 내부 발신처럼 보이는 메일, 공급망 위장 등은 점점 정교해지고 있으며, 단순한 스팸 필터나 URL 필터링으로는 대응에 한계가 있다는 점을 지적

결국, 대규모 금전적 피해와 개인정보유출 발생

01 YES24 랜섬웨어 사고

2025. 06~08, 두 차례 발생, 약48억의 손실 및 5일이 넘도록 서비스 마비

02 미래엔서해에너지 랜섬웨어 사고

2025.09, 한 차례 발생, 5일이 넘는 사이트 마비와 8TB에 달하는 데이터 탈취 주장

03 SK 텔레콤 악성코드 해킹 사고

2025.06 최초 공개, 3년간 여러 차례 발생, 유심정보 25종, 2,696만건 유출 및 1000억이 넘는 과징금

2025.08., “SK텔레콤 “개보위 1347억원 과징금 결정에 무거운 책임감 통감”, 조선일보, https://biz.chosun.com/it-science/ict/2025/08/28/RB6J7YL7RBEWBB47AEZ6CYXZSE/

단독] "백업 완료" 예스24 해명은 거짓…해커들에 비
예스24, 2분기 결
해킹 사고 여파에 잡손실 48억 반영
"잡손실 대부분 해킹 관련 비용으로 주
SK텔레콤 사흘새 10만명 이탈
피해자는 2324만4649명
SKT, 역대 최대 과징금 1347억원 '철퇴'..
미래엔서해에너지, 랜섬웨어 공격에 고객 개인정보 유출
[유출항목]
- 이름 / 생년월일 / 연락처 / 이메일 / 계좌번호 / 카드번호 / 주민등록번호(일부) 등
유출된 개인정보 항목에는 개인별로 차이가 있을 수 있으며, 결제 시 활용되는 비밀번호, CVC 값 등은 당사에서 수집하지 않으므로
이번 유출과는 무관함을 알려드립니다.

전통적인 탐지 기법

시그니처 기반 탐지

- 알려지지 않은 새로운 공격(0-Day),
  코드를 바꾼 변종 악성코드를 탐지할 수 없다.

행위 기반 탐지

- 악성코드가 샌드박스(분석 환경)를 감지해 동작을 숨긴다.
- 특정 조건(시간 지연·사용자 상호작용 등)에서만 악성행위를 발현하도록 설계
- 따라서 탐지 실패 및 추가 시간·자원 소모가 발생한다.

2023.03 , “신종 정보탈취 악성코드 LummaC2, 불법 크랙 위장해 유포”, 보안뉴스,
https://www.boannews.com/media/view.asp?idx=115357

Cuckoo Sandbox evasion in one Windows API function call

There are more than 400 Native API functions (or Nt-functions) in **ntdll.dll** that are usually hooked in sandboxes. In such a large list, there is enough space for different kinds of mistakes. We checked the hooked Nt-functions and found several issues.

One of them is a discrepancy in the number of arguments in the hooked and the original **NtLoadKeyEx** function. If a function is hooked incorrectly, in kernel mode this may lead an operating system to crash. Incorrect user-mode hooks are not as critical. However, they ma lead an analyzed application to crash or can be easily detected.

Let's look at the **NtLoadKeyEx** function. This function was first introduced in Windows Server 2003 and had only 4 arguments:

```
; Exported entry 235. NtLoadKeyEx
; Exported entry 1072. ZwLoadKeyEx
; __stdcall NtLoadKeyEx(x, x, x, x)
public _NtLoadKeyEx@16
```

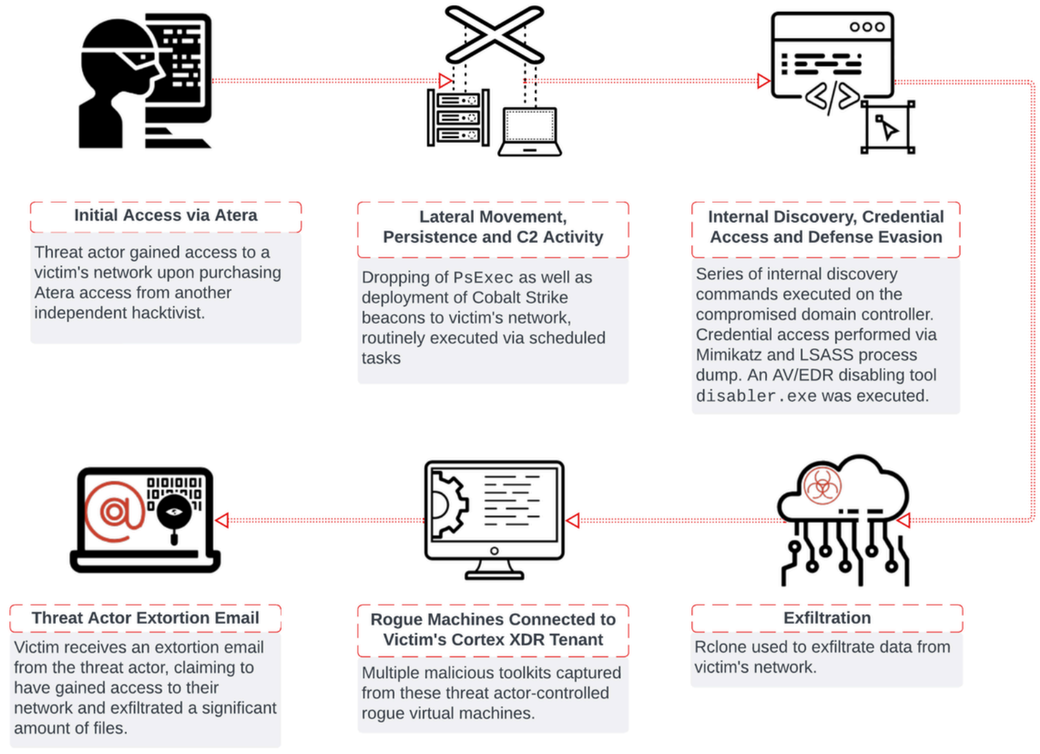
Later on, this function changed significantly. Starting from Windows Vista up to the latest version of Windows 10, it has 8 arguments:

```
; Exported entry 318. NtLoadKeyEx
; Exported entry 1450. ZwLoadKeyEx
; __stdcall NtLoadKeyEx(x, x, x, x, x, x, x, x)
public _NtLoadKeyEx@32
```

2022.02 , “Invisible Sandbox Evasion”, Check Point Research,
https://research.checkpoint.com/2022/invisible-cuckoo-cape-sandbox-evasion/

EDR 솔루션 탐지

- 수많은 로그와 이벤트가 발생하므로, 보안 전문가가 이를 분석하고 실제 위협을 가려내는 데 많은 시간과 노력이 필요하다. 전문 인력이 부족한 경우 운영에 어려움을 겪을 수 있다.
- 엔드포인트의 활동을 상시 모니터링하고 분석 에이전트가 동작하기 때문에 성능에 영향을 줄 수 있다.
- 암호화된 통신을 통해 이루어지는 공격이나, PowerShell 등 시스템에 기본적으로 내장된 정상도구를 악용하는 등의 탐지 회색지대가 존재한다.



paloalto NETWORKS | UNIT 42

2024.11, “TA Phone Home: EDR Evasion Testing Reveals Extortion Actor's Toolkit”, Unit 42 (Palo Alto Networks), https://unit42.paloaltonetworks.com/edr-bypass-extortion-attempt-thwarted/

단일 모델 접근의 한계

이미지 기반 악성코드 분류

- 변종에 강하나, 정규화 어렵고 모델 설명력 부족
- 패딩/압축률/섹션 배열에 따라 픽셀 패턴 급변

PE 구조기반 분석

- 안정성은 높으나 난독화 및 패킹에 취약
- 리소스/오버레이 내 페이로드 은닉 시 탐지 난항

Opcode 기반 분석

- 변종에 취약, 컴파일러에 따른 일반화 어려움
- 쓸모없는 코드삽입(NOP slide, junk code)·가젯 재배열에 민감

각 모델의 **사각지대를 상호 보완**하는 **다중 관점**이 필요

융합 모델 제안

01 기존 방식의 한계를 극복하는, 제로데이 등 변종에 강한 탐지 모델

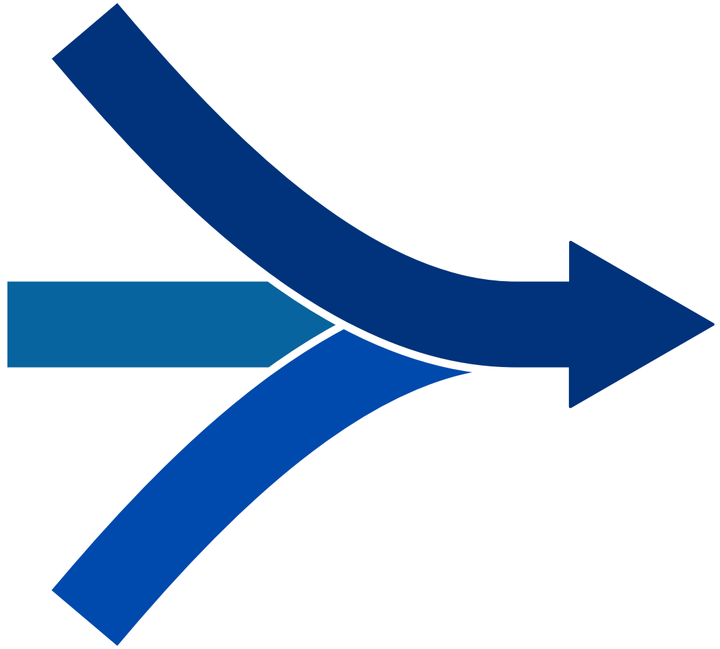
02 미탐을 최소화하고, 정확도를 최대로 높인 보수적인 모델

03 독립적인 모델을 융합하여, 균형잡힌 결과를 도출하는 안정적인 모델

이미지 분석
시각적 패턴을 통한 분류
난독화 탐지 가능

정적 분석
파일 자체를 분석
특정 함수, API 호출 등
구조적 특징 파악

OPCODE 분석
실제 실행 흐름 반영
코드 행위 패턴 학습



02 WHAT
서비스 소개

- 서비스 구조
- 세부 모델 소개
- Soft Voting
- 트러블 슈팅

시연 영상

YouTube https://youtu.be/zJJxkiXIPXY

서비스 주요 기능

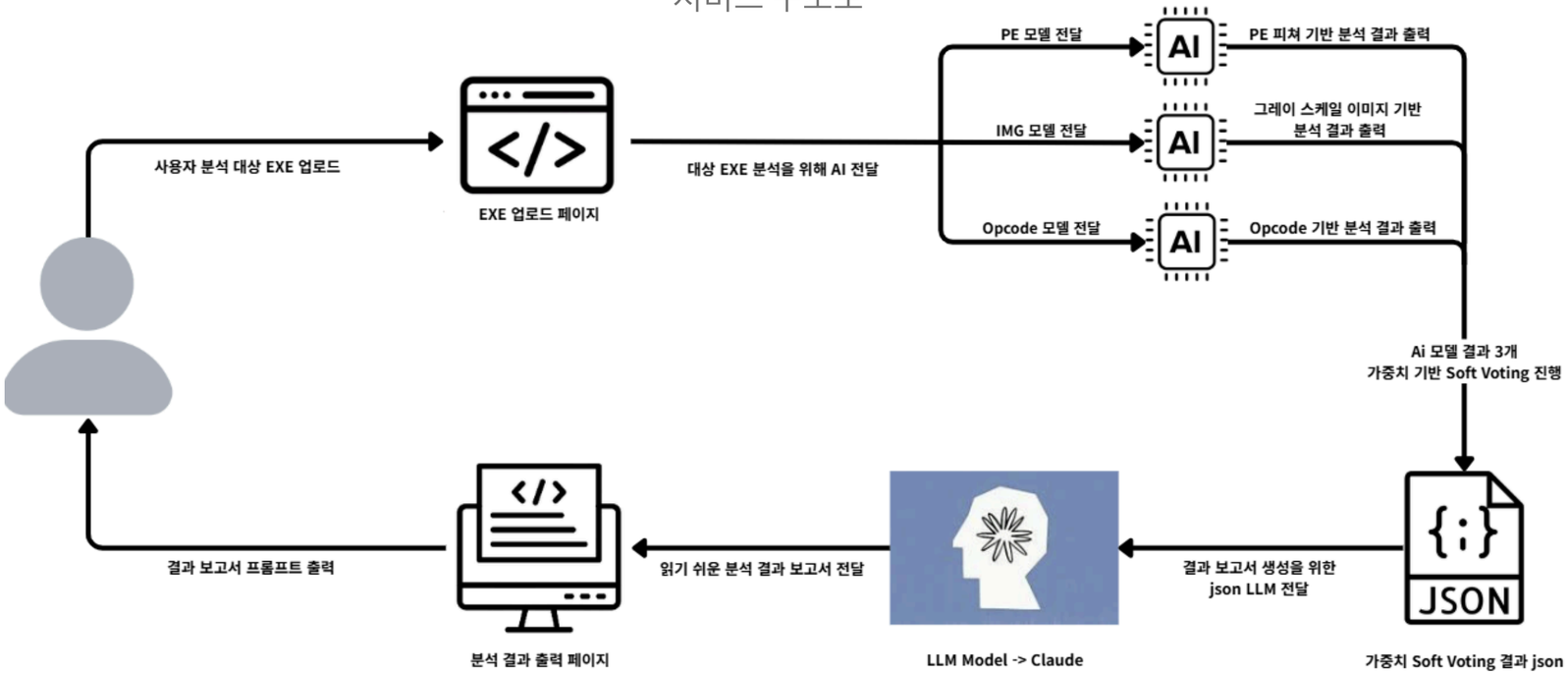
AI 기반 악성코드 분석 및 보고서 출력 서비스

사용자가 업로드 한 실행파일을
3가지의 세부 모델에 전달해 각각 분석

모델 앙상블을 통해 최종적인 결론을
AWS Bedrock과 같은 LLM 서비스를 이용해 출력

사용자는 피처 추출 결과, 악성 확률 등 보고서 출력 가능

서비스 구조도



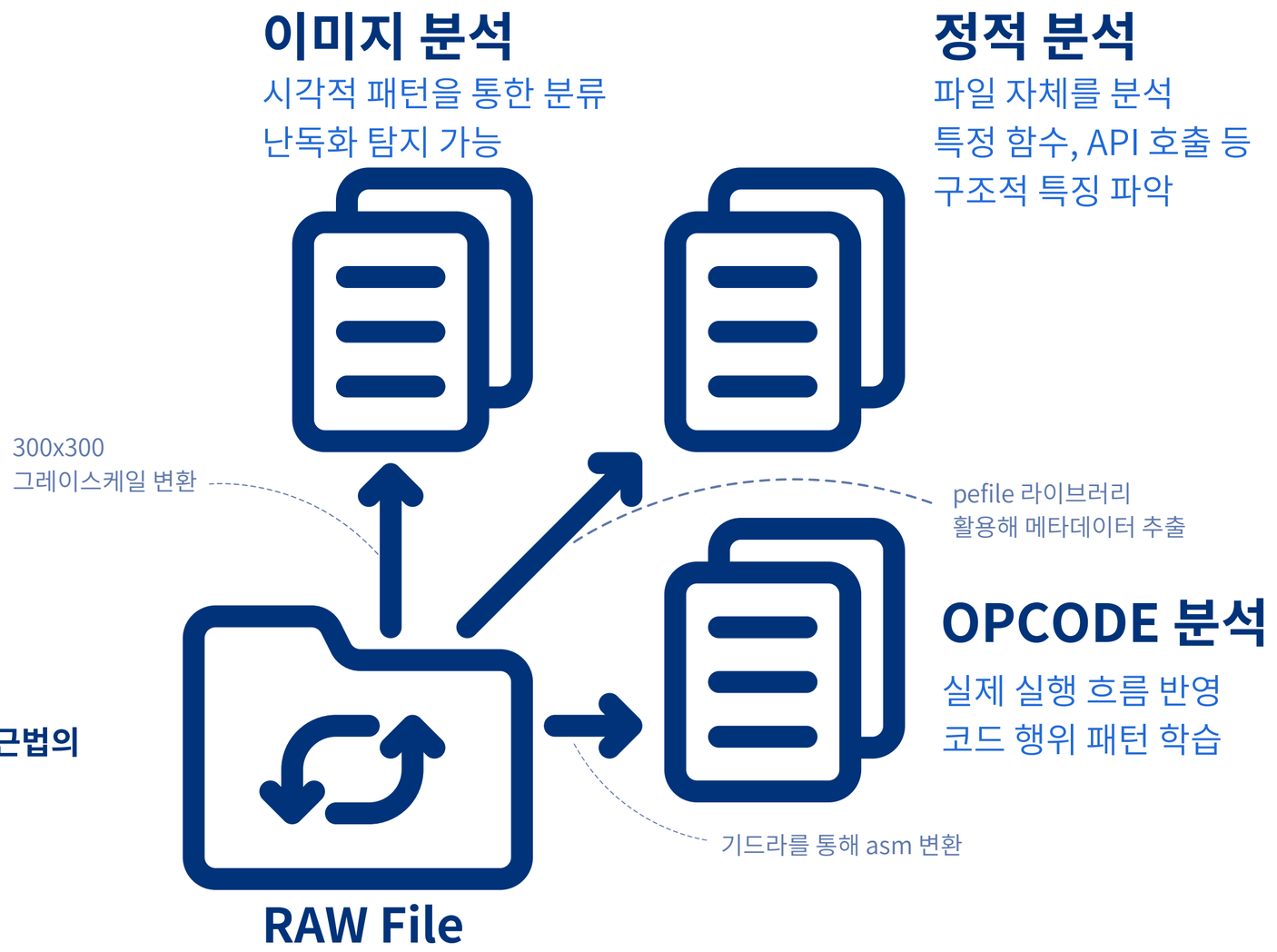
3가지 모델

PE 구조 분석, Opcode, Image
각각의 모델을 학습 후
Soft Voting으로 앙상블

이미지 변환은 '형태학적 패턴'을,
PE 헤더는 파일의 '정체성 신호'를,
OPCODE는 '행위 문법'을 포착해,
각각 다른 관점의 분석 결과를 제공한다.

서로 다른 편향·분산·표현력을 가진 모델들을 통해 기존 접근법의 취약점을 효과적으로 보완할 수 있다.[1].

[1]L. I. Kuncheva, C. J. Whitaker, "Measures of Diversity in Classifier Ensembles and Their Relationship with the Ensemble Accuracy," Machine Learning, 51(2), 181–207, 2003, DOI: 10.1023/A:1022859003006.



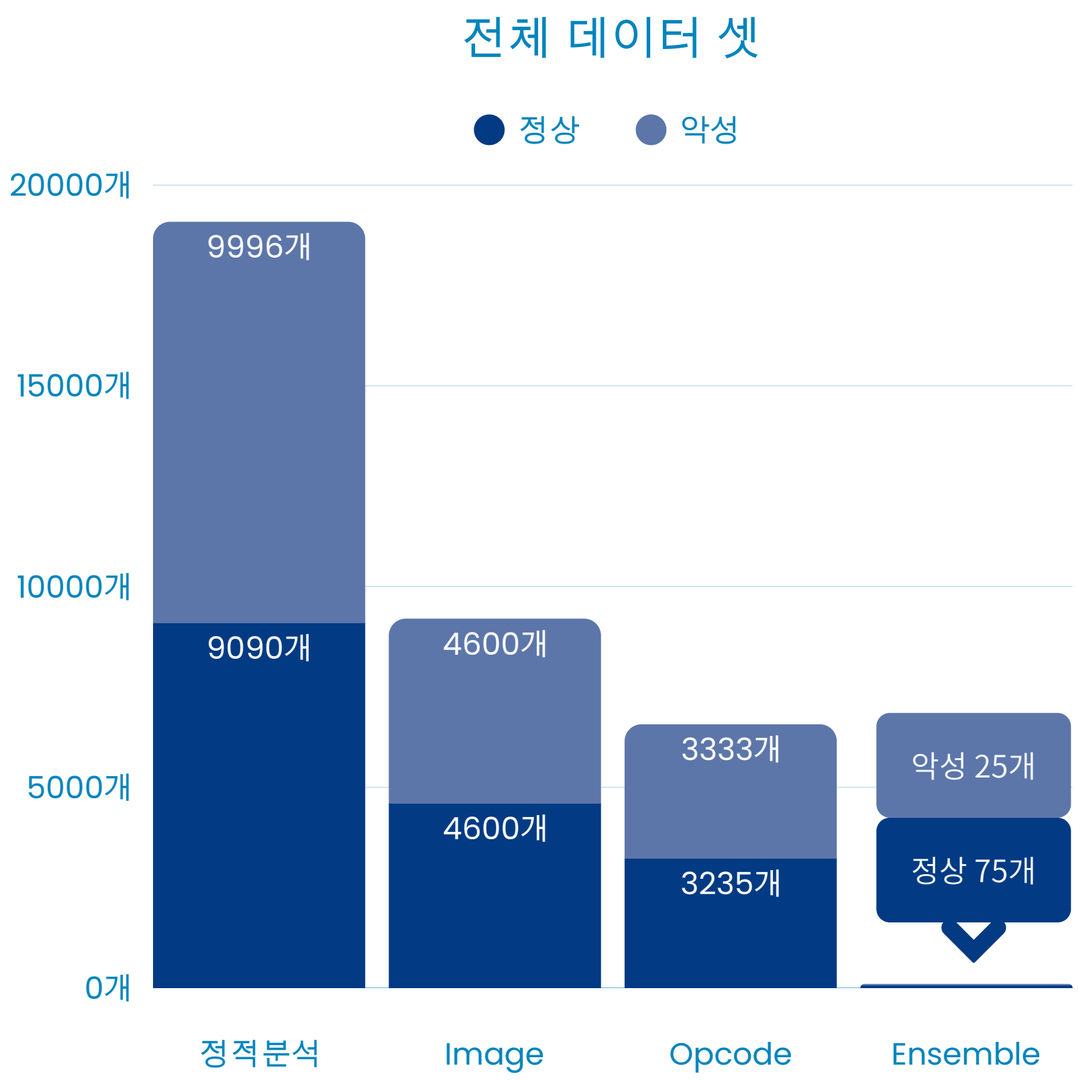
각 모델별 다양한 데이터 셋 활용

01 정적분석은 악성 데이터셋으로 KT 통신 빅데이터 플랫폼의 '정상 및 악성코드 데이터셋', 정상으로 동일 데이터셋의 정상 데이터 3,000개, 일반 사용자의 윈도우 실행파일 약 3,000개, 그리고 'Practical Security Analytics LLC'의 정상 데이터셋 약 3,000개 사용

02 Image는 KT 통신 빅데이터 플랫폼의 악성 데이터셋 'KT Malware Set' 4,600개와 정상 2,500개 그리고 자체 추출한 3,600개+'potableapps.com'에서 1,269개 중 정상 데이터셋 중복을 방지하기 위하여 정상 데이터셋 전처리 과정을 거쳐 최종적으로 악성 : 정상 = 4,600 : 4,600개 사용

03 Opcode는 'Microsoft Malware Classification Challenge'의 악성exe → ASM파일 변환 데이터셋 3,333개, 정상으로는 자체 추출한 정상 데이터셋과 portableapps.com 에서 받은 상용 프로그램을 ASM 파일로 변환하여 3,235개 사용

04 Ensemble는 악성데이터셋으로 KT 통신 빅데이터 플랫폼의 악성 데이터셋 '애드웨어 악성코드 데이터셋', '봇넷 악성코드 데이터셋', "웜바이러스 악성코드 데이터셋' 등 총 7가지 데이터셋으로부터 25개씩, 정상으로 일반 사용자의 윈도우 실행파일 75개 사용



악성 출처https://www.kaggle.com/competitions/malware-classification/data + 패밀리
정상 출처 : ldh's 추출 + portablesapps.com

01
PE 구조분석

- 사용한 피처
- 전처리 및 모델링
- 사용한 모델

사용한 피처

Header, Import, Strings, Yara 등 총 39개의 피처 사용

1. PE header 피처

※ 형광색 표시한 피처만 사용

Number	Feature name	Number	Feature name
1	DllCharacteristics	16	SubSystem
2	MajorImageVersion	17	MinorImageVersion
3	MajorOperatingSystemVersion	18	SizeOfStackCommit
4	SizeOfStackReserve	19	e_lfanew
5	AddressOfEntryPoint	20	e_minalloc
6	Characteristics	21	e_ovno
7	SizeOfHeaders	22	Machine
8	SizeOfInitializedData	23	PointerToSymbolTable
9	SizeOfUninitializedData	24	NumberOfSymbols
10	MajorSubsystemVersion	25	Magic
11	MinorSubsystemVersion	26	SizeOfCode
12	CheckSum	27	BaseOfCode
13	ImageBase	28	SectionAlignment
14	MajorLinkerVersion	29	FileAlignment
15	NumberOfSections		

- Hasan H. Al-Khshali and Muhammad Ilyas, “Impact of Portable Executable Header Features on Malware Detection Accuracy,” Computers, Materials & Continua (2023)
- 버전, 운영체제, 아키텍처 정보, 진입점, 섹션 수 등 메타데이터 정보

- 호출하는 API와 DLL 편중으로 로더·스티럴 행위를 추정
- 문자열 피처는 난독화·인코딩·C2·페이로드 토큰 존재를 드러냄
- 파일의 패킹 여부와 알려진 악성 패턴을 식별

2.Import & Strings & Yara rules

Feature name	Short description	Why it matters
imports_total	파일에 정의된 전체 임포트 함수 수	임포트량이 비정상적으로 많으면 로더·스틸러 등 악성 행위 의심
imports_unique	중복 제거한 고유 임포트 함수 수	고유 API 다양성으로 정상/비정상 행위 패턴 구분
import_dlls_unique	임포트를 제공하는 고유 DLL(라이브러리) 수	DLL 수가 적고 특정 DLL에 편중되면 드로퍼/로더 의심
imports_max_per_dll	한 DLL이 가진 임포트 함수 수의 최댓값	특정 DLL에 임포트가 몰림(편중도) 여부 확인
imports_entropy	DLL별 임포트 분포의 엔트로피	엔트로피 낮음 = 편중 → 의심 스코어 상승
strings_entropy	파일 내 문자열 전체의 엔트로피	난독화·인코딩(Base64 등) 여부 판단에 유용
strings_printable_ratio	파일 내 출력 가능한 ASCII 문자 비율	낮으면 바이너리 데이터·암호화 가능성
strings_avg_len	추출된 문자열의 평균 길이	짧은 문자열 다수면 난독화·토큰화 흔적 가능
strings_base64_blob_count	Base64 형태로 보이는 문자열 블롭 개수	숨겨진 페이로드·C2 토큰 존재 가능성 지표
yara_has_packer_generic	패커 관련 YARA 룰(일반) 탐지 여부	패킹 검출 시 정적 신호 약화 → 언패킹 우선 필요
yara_count_packer	매칭된 패커 관련 YARA 룰 개수	룰 다수 매칭 시 패킹 확률·유형 신뢰도 상승
yara_has_upx_like	UPX 계열 패커 탐지 여부	UPX 계열이면 표준 언패커 적용 우선
yara_has_mpress_like	MPRESS 계열 패커 탐지 여부	MPRESS 계열 특이 처리 필요
yara_has_aspack_like	ASPack 계열 패커 탐지 여부	ASPack 등 상용 패커 탐지는 언패킹/동적분석 필요

전처리 및 모델링

1. 결측치 처리

- 연속형 변수들은 중간값으로, 나머지 변수는 0으로 채움 (없다는 의미로 해석)

2. 스케일링

- 표준정규화: 값의 범위가 수천~수억 단위 이상으로 크거나, 연속형 특성이라 크기 자체가 영향을 줄 수 있는 경우
- 로그 스케일 변환 후 표준정규화: 값의 분포가 심하게 치우쳐 있고 일부 극단치가 큰 영향을 주는 경우
- 스케일링X: 값이 작거나(0~10 수준), 0/1 이진값, 이미 [0,1] 구간 비율, 혹은 범주형 성격을 띠는 경우

3. 파라미터 튜닝

- 랜덤 추출 기반의 RandomizedSearchCV를 사용.
- 각 모델별 40회 시도(n_iter=40), 5-Fold 교차검증 수행
  → 총 200 fit(40×5).
- 스코어링 기준: roc_auc
  → 이진 분류 불균형 문제에서 Accuracy보다 민감도/특이도를 반영

```
# 파라미터
xgb = XGBClassifier(
    tree_method="hist",random_state=42,
    scale_pos_weight=scale_pos_weight,
    reg_lambda=2.0, reg_alpha=0.0,
    n_estimators=600, min_child_weight=1,
    max_depth=8, learning_rate=0.07,
    gamma=0.0, colsample_bytree=0.8,
)
```

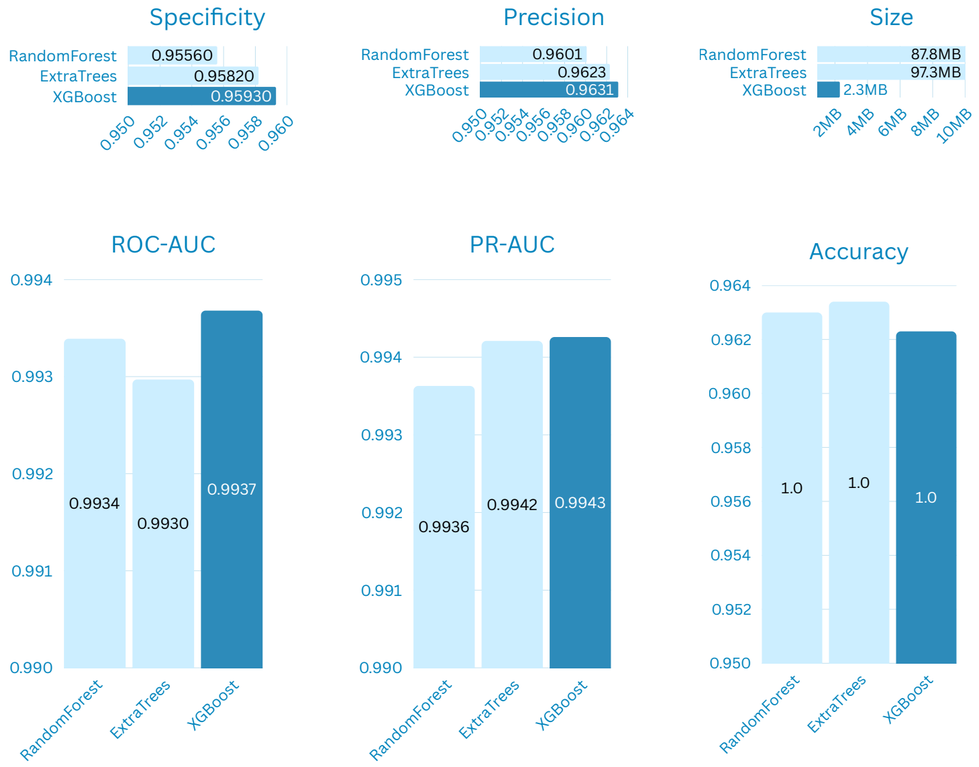
파라미터 선정 결과

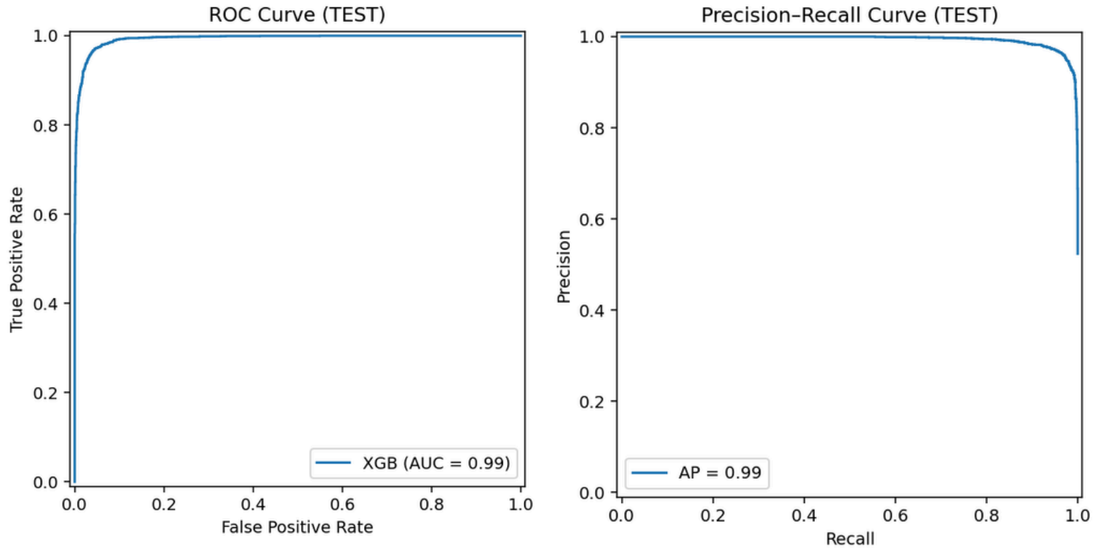
	count	mean	std	min	25%	50%	75%	max
DllCharacteristics	16605.0	3.765598e-17	1.000030	-1.195440	-1.195440	0.512282	0.565132	1.570937
SizeOfStackReserve	16605.0	-2.995362e-17	1.000030	-0.713796	-0.555842	-0.555842	-0.337203	4.340747
AddressOfEntryPoint	16605.0	-2.300010e-18	1.000030	-0.023137	-0.022673	-0.021935	-0.021835	64.170481
Characteristics	16605.0	3.765598e-17	1.000030	-0.253085	-0.247844	-0.210994	-0.125175	5.681735
SizeOfHeaders	16605.0	3.503504e-18	1.000030	-0.028208	-0.018164	-0.018164	-0.018164	128.772856
SizeOfInitializedData	16605.0	-8.023292e-18	1.000030	-0.138485	-0.131538	-0.111629	-0.047624	65.468732
SizeOfUninitializedData	16605.0	-2.112800e-18	1.000030	-0.020212	-0.020212	-0.020212	-0.017881	122.525569
SizeOfStackCommit	16605.0	-5.348861e-18	1.000030	-0.075898	-0.061606	-0.061606	-0.061606	58.461135
SizeOfCode	16605.0	-1.470937e-19	1.000030	-0.024744	-0.022434	-0.022267	-0.019273	127.461638

전처리 결과 일부

사용한 모델

XGBoost 모델 선택



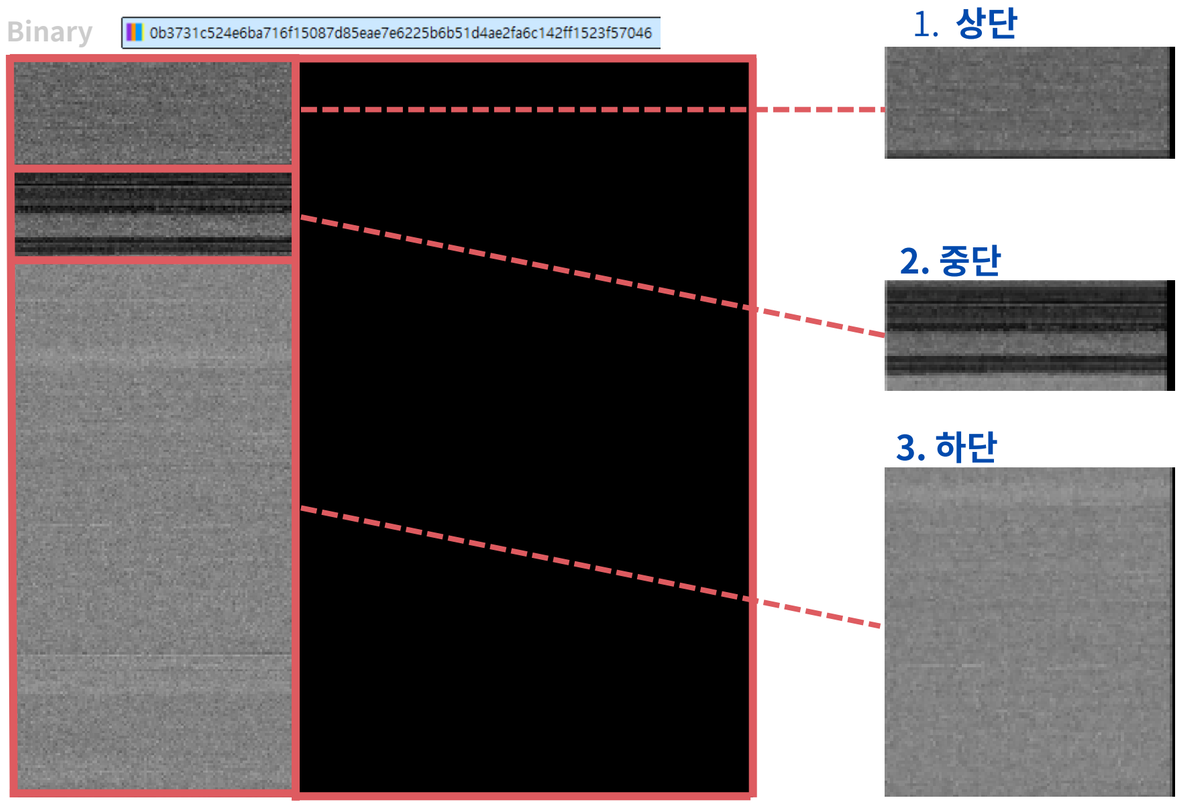


정확도는 타 모델에 비해 0.002 낮지만,
오탐 억제 지표(정밀도·특이도)와
전체 분리 성능(ROC/PR-AUC)을 확보하면서도
작은 용량으로 추론 지연·메모리·배포 비용을 낮춤

02 IMAGE

- 이미지 분석 및 해석
- 전처리 및 모델링
- 사용한 모델

3밝기 패턴(Gray Scale)의 해석



PE 파일의 크기에 따라 계산된 너비로2D 배열 생성
2D 배열 → 300 x 300 픽셀 크기의 정사각형 켄버스에 맞춰 리사이징

1. 상단

[상단 - 어두운 부분]

- PE 헤더와 섹션 테이블의 위치

(바이트 값 0x00 / 메타 데이터)

2. 중단

[중간 - 뚜렷한 가로줄]

- PE 파일의 섹션 경계 의미

(각 섹션의 시작은 데이터 분포가 급격하게 변화)

- 정상 파일과 다른 악성코드의 특징

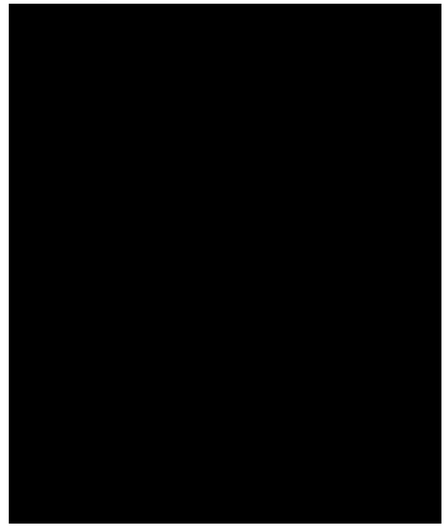
섹션의 개수, 순서, 크기 등을 조작하거나 비정상적인 섹션을 추가

3. 하단

[하단 - 전반적인 회색 질감]

- 데이터 밀도 높고, 바이트 값이 비교적 균일하게 분포 (코드 섹션 / 데이터 섹션의 특징)
- 밝은 회색(= 고 엔트로피)
- 어두운 회색(= 낮은 바이트 값)

4. 검정



[기타 - 검은색 영역]

- 데이터 채운 뒤 남은 영역

(검은색 = 픽셀 값 0)

- 파일 크기 분포가 특정 그룹

(Ex_랜섬웨어)에 치우쳐 있는지 파악

왜 EfficientV2-S 인가

1. 난독화 / 패킹에 대한 강력한 내성
- 구조적 패턴 학습
- 패킹 특징의 정량화

2. 고효율 / 고해상도 특징 추출
- 300 x 300에 최적화(고해상도)
- 효율적인 전이 학습

3. 균형 있는 확장
- compound scaling 방식을 활용하여
- 깊이, 너비, 해상도를 균형 있게 확장

4. 복합 스케일링
- 효율적인 모델의 크기
- 동적 해상도 및 스케일링
- 전이 학습 유리

EfficientV2-S : 전처리 및 모델학습

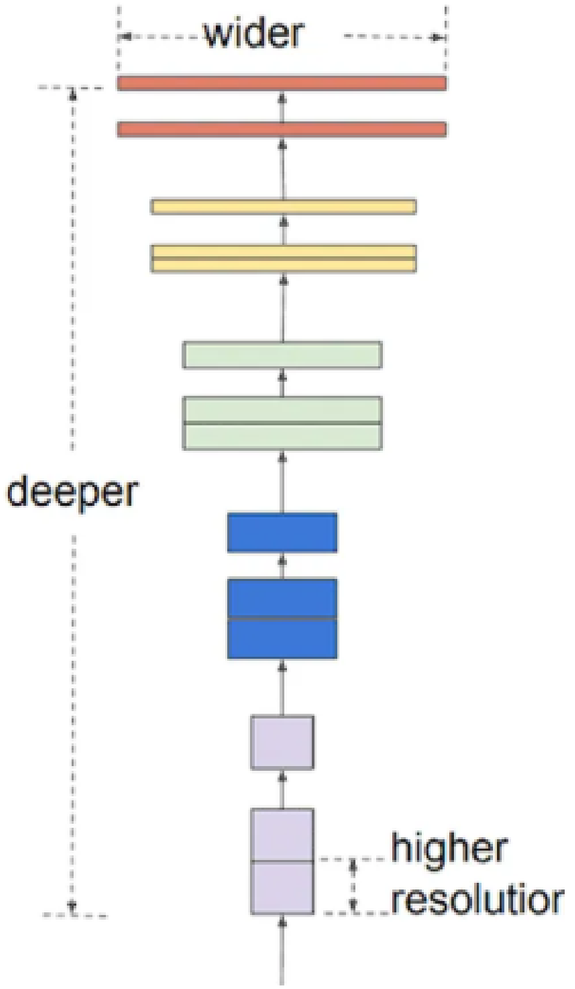
1. 전처리

- **EfficientV2-S 맞춤 최적화 전처리**
  - 300x300 리사이징
- **DATA 품질 필터링**
  - 과적합 방지
  - 유사 중복 제거
  - Split 누수 방지
  - 패킹 진단
  - PE 구조 이상 탐지

2. 모델 학습

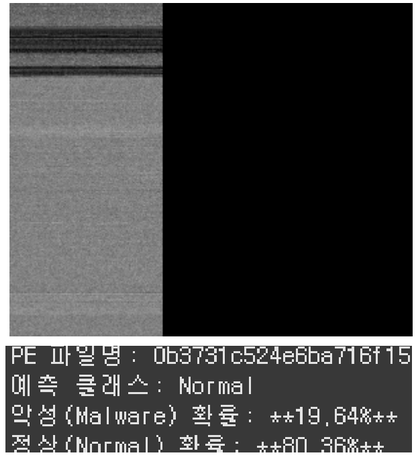
- **환경 / DATA 구축(EfficientV2-S)**
  - 1채널 → 3채널 확장
    (R : G : B = 150 : 150 : 150)
- **K-Fold(Fold = 5) 교차 검증학습**
- **최종 모델 성능 평가 & 시각화 리포트**
  - ROC - AUC/PR - AUC
  - F1 Score/Recall(민감도)
  - Specificity(특이도)
- **시각화**
  - ConfusionMartrix
  - ROC/PR Curve

복합 스케일링



악성

정상

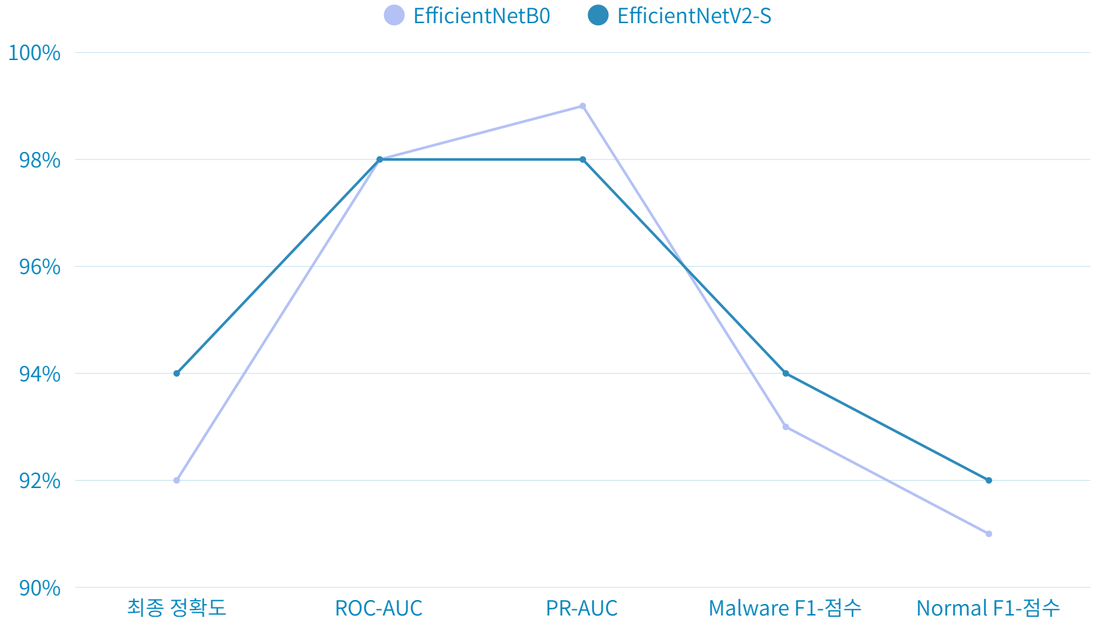


사용한 모델

EfficientV2-S vs EfficientB0

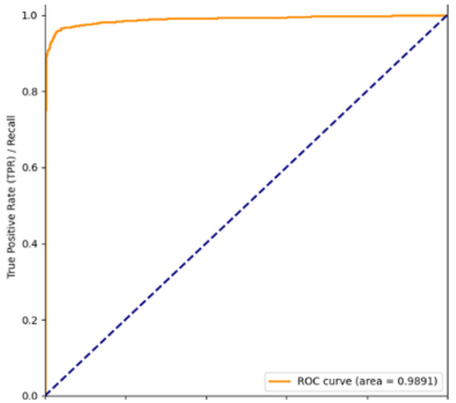
- **2000개 파일 POC 진행 결과**
  - 정확도 : **V2-S**
  - PR-AUC : **B0**
  - ROC-AUC : **V2-S**
  - F1-Score : **V2-S**

→ **EfficientV2-S 사용**
전체적으로 안정적인 성능을 확인

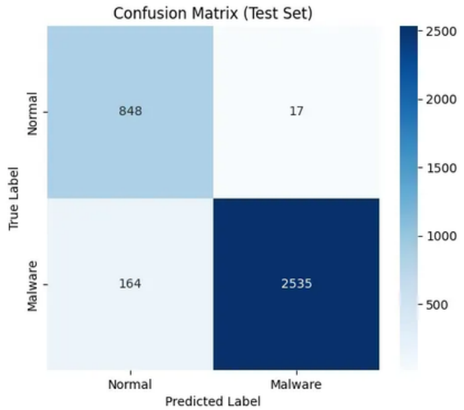


EfficientV2-S 모델 학습 결과

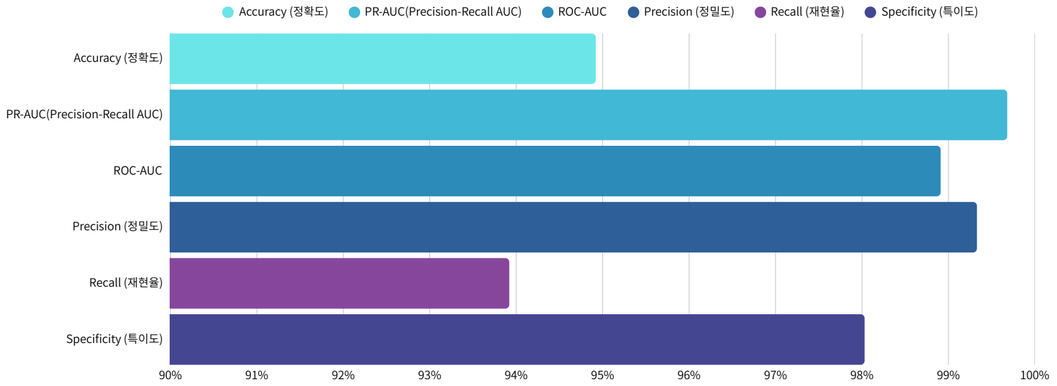
ROC-Curve



Confusion Matrix



최종 모델 성능지표



03 OPCODE

- 데이터 전처리
- 피처 추출&TF-IDF
- 사용한 모델

데이터 전처리

1. 데이터셋 무결성 검증

- 1차 제거 : ASM 파일의 내용이 없거나 JMP 한 줄만 있는 경우
- 2차 제거 : 파일 명은 달라도 해시값이 같은 파일

```
; Program : chktrust.exe
; Lang    : x86:LE:32:default
; AddrSz  : 4
; --- auto-generated by export_asm.py ---


; ===== Function entry @ 0040525a =====
0040525a: FF 25 00 20 40 00         JMP dword ptr [0x00402000]
```

```
; Program : Com.Samsung.Scpm.Client.Win32.exe
; Lang    : x86:LE:64:default
; AddrSz  : 8
; --- auto-generated by export_asm.py ---
```

1차 제거 예시

2. 데이터셋 정제 및 정규화

- Opcode 화이트 리스트 필터링
- Opcode 내용이 같은 파일 제거

Opcode 화이트 리스트

```
OPCODE_WHITELIST = {
    # 1. 데이터 이동 (Data Transfer)
    'mov', 'lea', 'push', 'pop', 'pusha', 'popa', 'xchg', 'movzx', 'movsx',

    # 2. 산술 연산 (Arithmetic)
    'add', 'sub', 'inc', 'dec', 'mul', 'imul', 'div', 'idiv', 'neg', 'sbb', 'adc',

    # 3. 논리 및 비트 연산 (Logic & Bitwise)
    'and', 'or', 'xor', 'not', 'shl', 'shr', 'sar', 'rol', 'ror',

    # 4. 흐름 제어 (Control Flow)
    'jmp', 'call', 'retn', 'ret', 'leave', 'enter',
    'je', 'jz', 'jne', 'jnz', 'jg', 'jnle', 'jge', 'jnl', 'jl', 'jnge', 'jle', 'jng',
    'ja', 'jae', 'jb', 'jbe', 'jc', 'jnc', 'jo', 'jno', 'jp', 'jpe', 'jnp', 'jpo',
    'js', 'jns',

    # 5. 비교 및 테스트 (Comparison & Test)
    'cmp', 'test', 'setz', 'setnz', 'seta', 'setae', 'setb', 'setbe',

    # 6. 문자열 및 데이터 블록 처리 (String & Block)
    'rep', 'repe', 'repne', 'movsb', 'movsd', 'stosb', 'stosd', 'cmpsb', 'scasb',
```

```
; Program : validate-docstrings.exe
; AddrSz  : 8
; --- auto-generated by export_asm_final.py ---

140001000: 85 C9                    TEST ECX, ECX
140001002: 75 6D                    JNZ 140001071
140001004: 4C 8B DC                 MOV R11, RSP
140001007: 49 89 53 10              MOV R110x10, RDX
14000100b: 4D 89 43 18              MOV R110x18, R8
14000100f: 4D 89 4B 20              MOV R110x20, R9
140001013: 48 81 EC 48 08 00 00     SUB RSP, 0x848
14000101a: 48 8B 05 A7 33 01 00     MOV RAX, 1400143c8
140001021: 48 33 C4                 XOR RAX, RSP
140001024: 48 89 84 24 30 08 00 00  MOV RSP0x830, RAX
14000102c: 4C 8B CA                 MOV R9, RDX
14000102f: BA 00 04 00 00           MOV EDX, 0x400
140001034: 49 8D 43 18              LEA RAX, R110x18
140001038: 44 8D 42 FF              LEA R8D, RDX-0x1
14000103c: 48 8D 4C 24 30           LEA RCX, RSP0x30
140001041: 48 89 44 24 20           MOV RSP0x20, RAX
140001046: E8 D9 2A 00 00           CALL 140003b24
```

```
; Program : toco_from_protos.exe
; AddrSz  : 8
; --- auto-generated by export_asm_final.py ---

140001000: 85 C9                    TEST ECX, ECX
140001002: 75 6D                    JNZ 140001071
140001004: 4C 8B DC                 MOV R11, RSP
140001007: 49 89 53 10              MOV R110x10, RDX
14000100b: 4D 89 43 18              MOV R110x18, R8
14000100f: 4D 89 4B 20              MOV R110x20, R9
140001013: 48 81 EC 48 08 00 00     SUB RSP, 0x848
14000101a: 48 8B 05 A7 33 01 00     MOV RAX, 1400143c8
140001021: 48 33 C4                 XOR RAX, RSP
140001024: 48 89 84 24 30 08 00 00  MOV RSP0x830, RAX
14000102c: 4C 8B CA                 MOV R9, RDX
14000102f: BA 00 04 00 00           MOV EDX, 0x400
140001034: 49 8D 43 18              LEA RAX, R110x18
140001038: 44 8D 42 FF              LEA R8D, RDX-0x1
14000103c: 48 8D 4C 24 30           LEA RCX, RSP0x30
140001041: 48 89 44 24 20           MOV RSP0x20, RAX
140001046: E8 D9 2A 00 00           CALL 140003b24
```

쌍둥이 파일 예시

Opcode Feature Extraction & TF-IDF 기반 벡터화

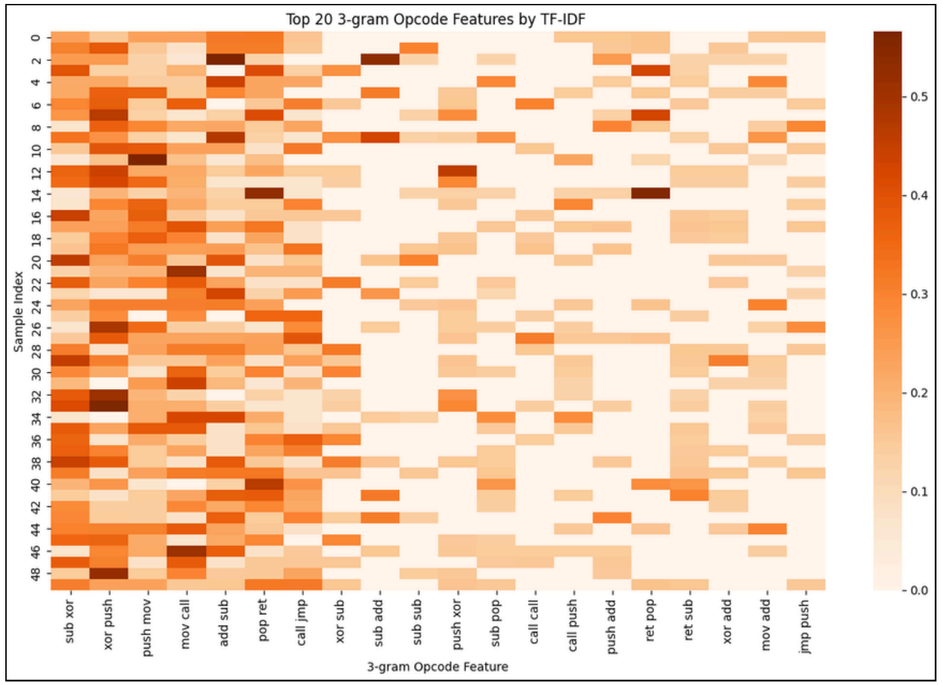
1. Feature Extraction 과정

- 각 악성코드 샘플을 디스어셈블하여 opcode 시퀀스 추출
- 이 시퀀스를 문장(token sequence)처럼 처리
- Opcode 빈도, 3-gram 패턴을 특징(feature)으로 사용

```
TARGET_TRIGRAMS_TO_TRACK = [
    # --- 공통 피처 (20개) ---
    'mov mov cmp', 'mov lea mov', 'mov mov call', 'mov jmp mov',
    'mov mov add', 'mov mov mov', 'test jz mov', 'mov mov lea',
    'jz mov mov', 'mov mov test', 'mov call mov', 'cmp jnz mov',
    'jmp mov mov', 'call mov mov', 'push push push', 'pop pop pop',
    'mov test jz', 'lea mov call', 'lea call mov', 'lea mov mov',

    # --- 악성 특화 피처 (15개, "retn sub sub" 제외) ---
    'mov push call', 'mov imul mov', 'mov push mov', 'push call add',
    'push push call', 'mov mov push', 'push push mov', 'imul mov mov',
    'mov push push', 'mov mov imul', 'mov mul mov', 'push mov push',
    'push mov call', 'push call mov','push mov mov',

    # --- 정상 특화 피처 (14개, "pop ret mov"와 "pop pop ret" 제외) ---
    'test jnz mov', 'mov cmp jz', 'mov call test', 'mov xor mov',
    'jmp mov mov', 'call lea mov', 'push sub mov',
    'lea mov lea', 'sub mov mov', 'mov add pop', 'jnz mov mov',
    'xor mov mov', 'call test jz', 'cmp jz mov'
]
```

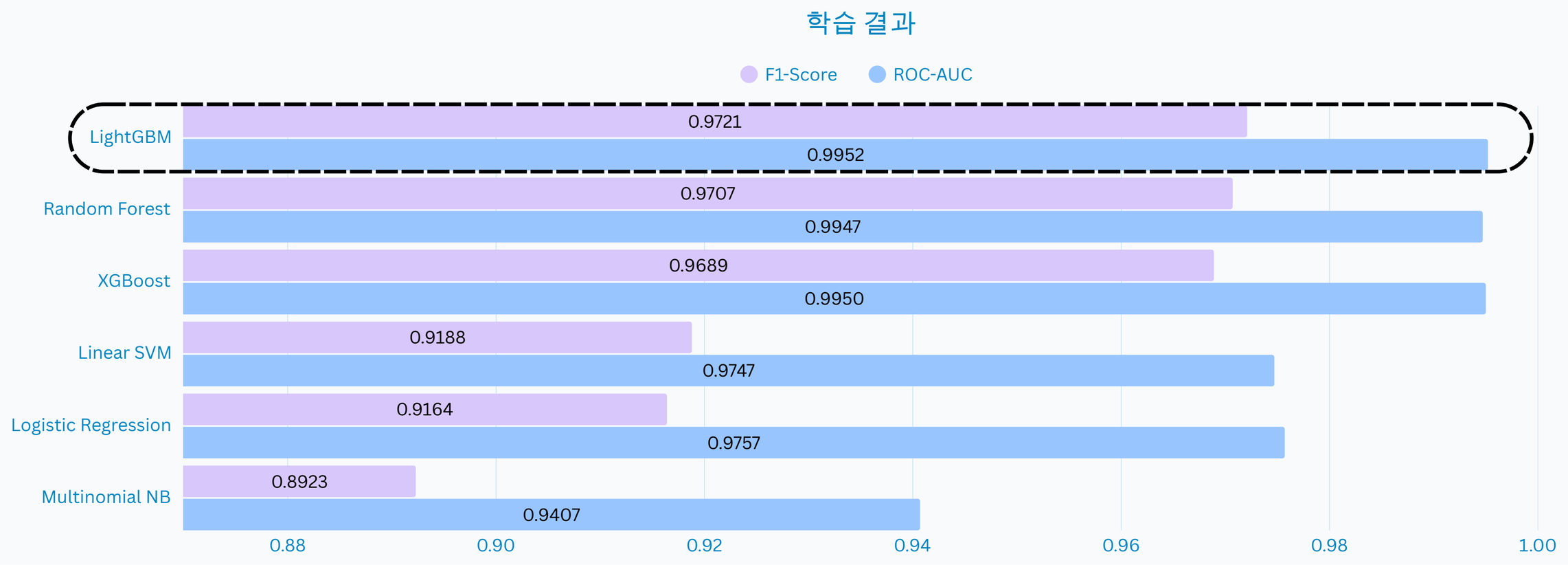
2. TF-IDF 기반 벡터화

- 특정 opcode가 얼마나 “특정 클래스(악성/정상)에 특이적인가”를 반영
- 단순 빈도보다 정보량 기반 가중치 부여 → 모델이 중요 opcode에 집중



사용한 모델

가장 높은 F1-SCORE와 ROC-AUC를 기록한 LIGHTGBM 모델 선택



04
ENSENBLE

- Soft Voting
- 앙상블 결과

Optuna 및 Baysian Log-odds

: 세 모델의 예측이 일치할수록 결합 신뢰도가 높아지는 학습적 확률 통합 방식

- **Baysian Log-odds:** $\text{Combined log-odds} = w_1 \log \frac{p_1}{1-p_1} + w_2 \log \frac{p_2}{1-p_2} + w_3 \log \frac{p_3}{1-p_3}$, $p_{\text{final}} = \frac{1}{1+e^{-\text{Combined log-odds}}}$

⇨ 모든 모델이 같은 방향으로 확신할수록, log-odds 합이 증가하여 최종확률이 높아진다.

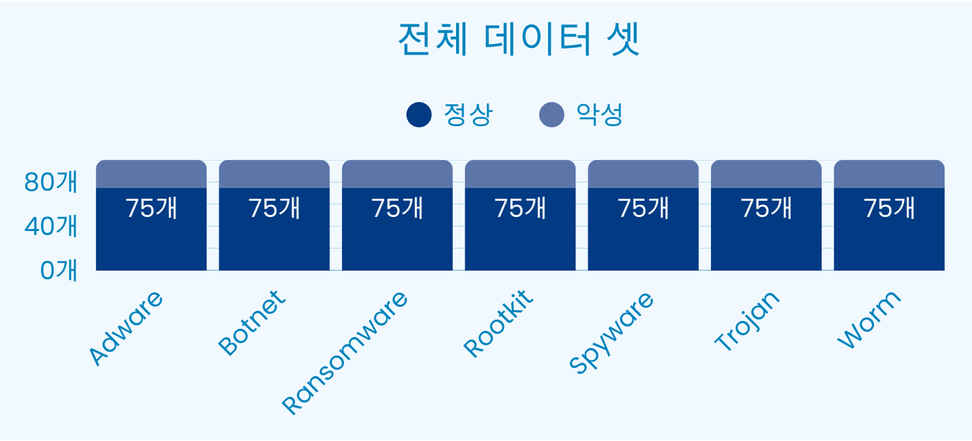
- **Optuna**로 하이퍼파라미터 최적화

  1) 패밀리간 **표준편차를 최소화**

  2) 패밀리별 **정확도 평균을 최대화**

Object () $\alpha \cdot std\_fnr + \beta \cdot std\_acc + \gamma \cdot mean\_fnr - \delta \cdot mean\_acc$

↳이 값을 최소로 만드는 것



하이퍼 파라미터	탐색 범위	기본값/의미	비고
w_pe	[0.0, 1.0]	PE 모델의 가중치	
w_img	[0.0, 1.0]	IMG 모델의 가중치	
w_opc	[0.0, 1.0]	OPC 모델의 가중치	
k	[0.4, 2.0]	결합 민감도	1보다 크면 민감도↑, 작으면 완만
bias_logit	[prior_bias - 3.0, prior_bias + 3.0]	사전 확률 기반 편향 보정값	클래스 불균형 보정용
threshold	[0.2, 0.8]	최종 이진 분류 기준	낮을수록 탐지↑(재현율↑), 오탐↑, 미탐↓

최적 하이퍼 파라미터 탐색 과정

0 Input Data(예시)

	PE구조 분석	Image	OPCode	실제 라벨
A.exe	0.90	0.70	0.60	악성
B.exe	0.40	0.30	0.20	정상
C.exe	0.80	0.50	0.40	악성

1 확률 → 로그오즈(logit)

	PE구조 분석	Image	OPCode
A.exe	2.20	0.85	0.41
B.exe	-0.40	-0.85	-1.39
C.exe	1.39	0.00	-0.41

$$\text{logit}(p) = \ln \frac{p}{1-p}$$

2 Optuna 하이퍼파라미터 탐색

최소	범위	최대
[0.0]	w_pe	[1.0]
[0.0]	w_img	[1.0]
[0.0]	w_opc	[1.0]
[0.4]	k	[2.0]
[prior_bias - 3.0]	bias_logit	[prior_bias + 3.0]
[0.2]	threshold	[0.8]

✓ 탐색 결과

w_pe	0.5
w_img	0.3
w_opc	0.2
k	1
k	0
threshold	0.5

3 Combine

A.exe	0.5×2.20 + 0.3×0.85 + 0.2×0.41 = **1.33**
B.exe	0.5×(-0.4) + 0.3×(-0.85) + 0.2×(-1.39) = **-0.76**
C.exe	0.5×1.39 + 0.3×0.00 + 0.2×(-0.41) = **0.55**

$$combined\_logit = bias\_logit + k \times \Sigma(w_i \times logit(p_i))$$

4 Sigmoid

A.exe	sigmoid(1.33) = 0.79
B.exe	sigmoid(-0.76) = 0.32
C.exe	sigmoid(0.55) = 0.63

$$\text{prob\_final} = sigmoid(combined\_logit)$$

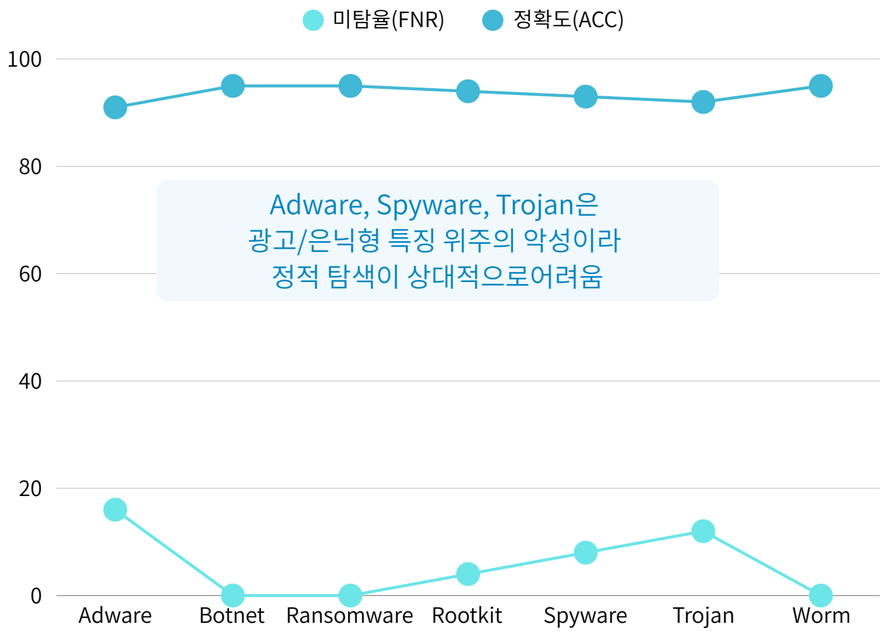
5 Threshold

A.exe	sigmoid(1.33) = 0.79 > 0.5	악성
B.exe	sigmoid(-0.76) = 0.32 > 0.5	정상
C.exe	sigmoid(0.55) = 0.63 > 0.5	악성

$$\text{pred} = 1[\text{prob\_final} \geq threshold]$$

앙상블 결과

PE 중심의 다중 특징 결합으로 높은 정확도와 민감 탐지를 달성한 앙상블 모델



미탐율 표준편차	정확도 표준편차	미탐율 평균	정확도 평균
0.0599	0.0158	5.7143	0.9323

하이퍼 파라미터	값
W_PE	0.418998112937013
W_IMG	0.201554043026697
W_OPC	0.101200998008704
K	1.066
bias_logit	0.544
Threshold	0.325

PE (약 60%): 파일 헤더·임포트 테이블·섹션 정보 등의 특징이 주 탐지 근거
IMG (약 30%): 이미지 변환 기반 CNN 특징이 PE가 잡지 못한 형태적 특이성 보완을 수행
OPC (약 10%): 보통 실행 시점의 코드 패턴 관련, 특정 악성 행위 패턴에 보정 효과
K: 모델의 확신정도가 거의 1에 근접. 과도한 확신을 피하면서, 분류간 명확한 경계를 약간 강화
bias_logit: 모델을 오른쪽(양수)로 0.544만큼 밀어냄. 악성 판정 쪽으로 확률이 약간 더 높게 보정
Threshold: 예측확률이 0.325 이상이면 악성으로 예측

⇨ **bias와 threshold로 FN(미탐) 최소화, 높은 민감도를 확보**

05
트러블 슈팅

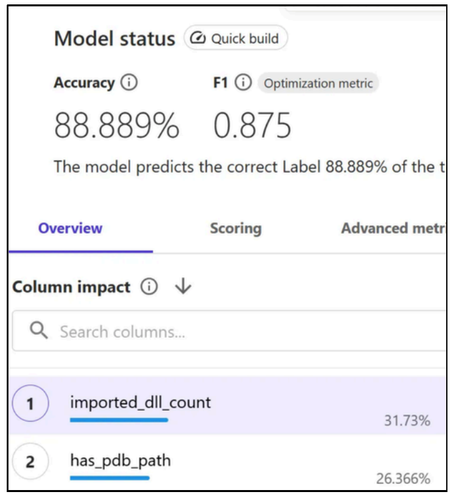
- 데이터 셋 관련
- 전처리 및 모델링 관련
- 기타

악성 바이너리 파일확보의 어려움

문제 발생 개요


```
"file_name": [
    "C:\\Windows\\System32\\Kpaemlmb.exe",
    "Kpaemlmb.exe"
],
"file_type": "Win32 EXE",
"mime_type": "application/octet-stream",
"file_size": 6758400,
"av_detection": {
    "a": "Trojan.Agent.DQQO",
    "b": "Backdoor.Berbew",
    "c": "Trojan.Agent.DQQO",
    "d": "Backdoor.Win32.Padodor.gen"
},
"pe_header_fileinfo_item_number": 0,
"pe_header_timestamp": "2020/07/11 03:39:59",
"pe_header_api_import_number": 142,
"pe_header_sectionsize_text": 47056,
"entropy_section_text": "7.20",
"pe_header_sectionsize_bss": 0,
"pe_header_sectionsize_data": 13076,
"entropy_section_data": "6.09",
```


※ 초기에 제공받은 Json파일의 일부
→ 빌드 환경 차이, 재현불가



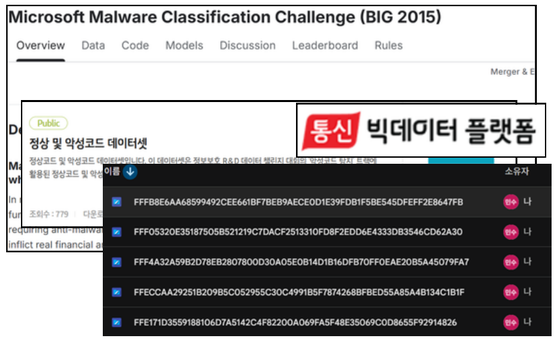
※ 너무 높은 특정 피처 의존도

- **제공받은 json파일의 문제**
- 피처 확장/수정 불가 (새로운 특징 추가 불가)
- **asm 변환 및 이미지 추출 불가**
- 데이터셋 자체의 한계로 인한 데이터 누수
- 툴 버전에 따라 추출 방식이 달라 재현성 보장 불가

- 악성코드 바이너리 원본은 그 특성상 전파 위험 등으로 **구하기가 쉽지 않음**
- **기존 데이터 플랫폼**
  - **C-TAS 플랫폼**
  - **Malwarebazaar**
  - **vx-underground**

→ json으로만 제공하거나, 출처 불확실

개선 및 해결



Kaggle - MS 2015 Challenge

- 대량 asm 파일 원본 확보
- 출처 신뢰 가능

KT 통신 빅데이터 플랫폼

- 대량 바이너리 원본 확보
- 출처 신뢰 가능

7월 3주차~ 9월 1주차까지, 5주가 넘는 시간 소요

문서형 악성코드 → 주제 변경 → json 파일 활용 실패 → 2주간의 데이터셋 탐색 → 데이터셋 선정

치트 피처 가능성 항상 경계

검증된 데이터셋의 확보 중요성

앙상블 검증 데이터 셋 부족

문제 발생 개요

현실 환경을 재현하기 위해 **불균형 데이터셋** 구성 필요

그러나, 학습에 사용하고 남은 정상 데이터셋은 단 75개

```
destination_dir = "/content/drive/MyDrive/FitBool/FamilyTest2/exe/newBen2/normal"

# Check if the directory exists
if os.path.exists(destination_dir):
    # Count the number of files in the destination directory
    file_count = len(os.listdir(destination_dir))
    print(f"There are {file_count} files in {destination_dir}")
else:
    print(f"Error: Directory not found at {destination_dir}")
```

```
There are 75 files in /content/drive/MyDrive/FitBool/FamilyTest2/exe/newBen2/normal
```

현실 비율을 반영하면서, 클래스 불균형을 완화하는 합의점 3:1 (75 : 25)

- 현실 비율: 실전 적합성과 신뢰도를 확보하기 위함
- 클래스 불균형 문제 완화: 각 모델의 성능을 더 객관적으로 평가하기 위함

⇨ **“총 100개의 데이터셋만으로, 모델별 가중치를 산출해야한다"**

개선 및 해결

```
Weights=[0.41899811293701344, 0.201554043026697, 0.10120099800870452], k=1.066, bias_logit=0.544, Threshold=0.325
std_fnr=0.0599 | std_acc=0.0158 | mean_fnr=0.0571 | mean_acc=0.9323

==== Ben vs 패밀리 성능 요약 (pe+img+opc, mode=evidence) ====
              family  accuracy   fnr
1  ben vs ransomware  0.947368  0.00
2         ben vs worm  0.947368  0.00
3       ben vs botnet  0.947368  0.00
6      ben vs rootkit  0.936842  0.04
4      ben vs spyware  0.926316  0.08
5       ben vs trojan  0.915789  0.12
0       ben vs adware  0.905263  0.16
```

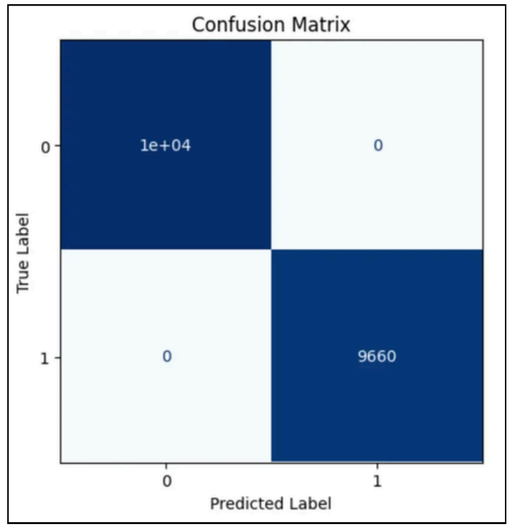
7개 악성 패밀리(adware, botnet, ransomware, rootkit, spyware, trojan, worm)을
각각 조합해 (75개 정상 + 25개 악성) × **7회 반복** 실험 수행

평균 성능과 변동성(표준편차)을 고려하여 가장 안정적인 Soft-Voting 가중치 도출

⇨ **“한정된 데이터에서도 7배의 반복성으로 실험 복잡성과 실험 실행의 신뢰성 확보”**

PE 정적분석 모델 과적합

문제 발생 개요



※ 오탐, 미탐 0인 정확도 100% 모델

	model	pr_auc_mean
0	ExtraTrees	0.999874
1	RandomForest	0.999865
2	HistGB	0.999865
3	GradientBoosting	0.999543

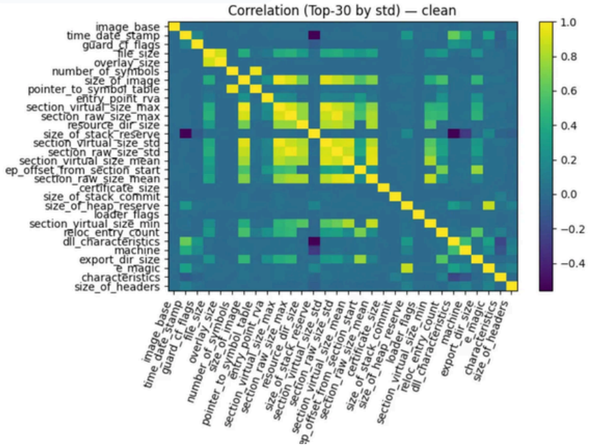
※ 너무 높은 특정 train auc

※ 중복되는 파일 발견

- 초기 실험에서 110개 이상의 피처를
- 모두 사용하여 학습
- 학습 데이터에서는 AUC 0.99 이상
  → 매우 높은 성능
- 혼동행렬 상 미탐 오탐 → 0%

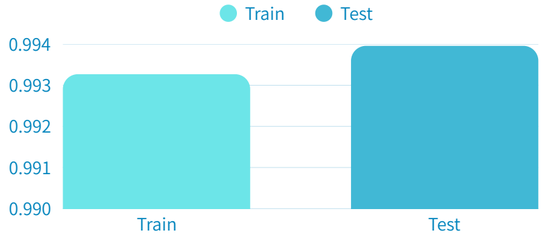
- 고차원 피처 공간
  - 일부는 서로 강하게 상관관계
  - 모델이 중복된 정보 학습 반복
- 검증 프로세스 부족
  - 정상데이터셋에서 중복되는 파일들 다수 발견

개선 및 해결




```
=== MODEL: xgb ===
CV ROC AUC (mean±std): 0.992411 ± 0.001333
CV AvgPrecision (mean±std): 0.995021 ± 0.000
```

※ 안정적으로 수렴한 CV ROC AUC



※ Train 및 Test의 AUC 비교

피처 차원 축소
피어슨 상관계수 기반으로
$|\rho| \geq 0.98$ 피처 제거
특정 데이터셋의 일반화가 아닌
모델의 일반화에 집중

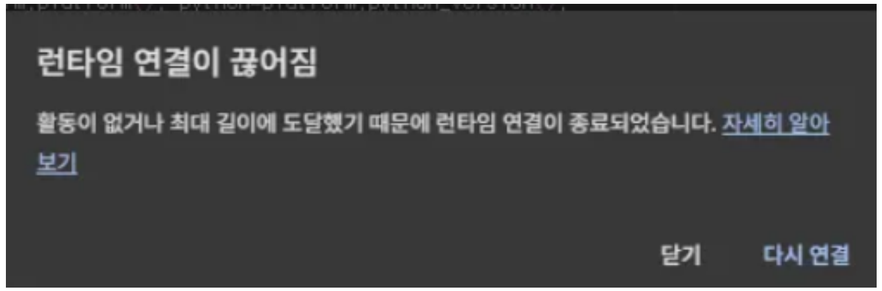
교차검증 도입
최종 CV 점수와 테스트 점수가
안정적으로 수렴하도록 모델링

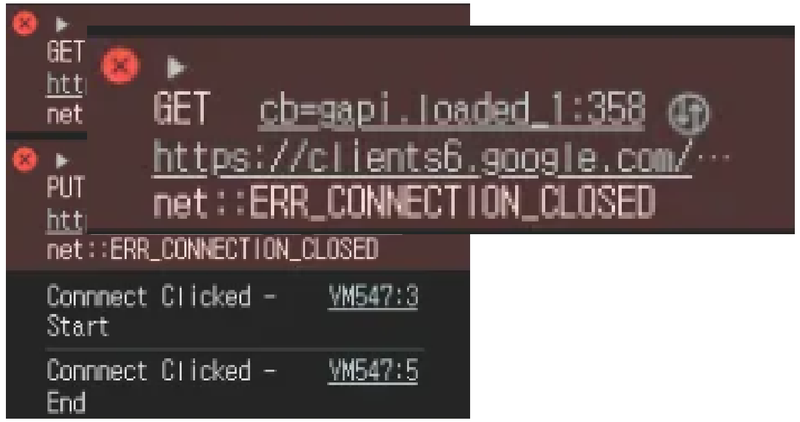
많은 피처 ≠ 좋은 성능
통제 가능하지 않은 무작정
많은 피처의 사용은 지양해야 함

I/O 병목 현상과 런타임 종료

전처리, 모델학습 과정에서의 ISSUE

- I/O 병목 현상
- 쓰로틀링 현상
- 런터임 연결 중단 현상





주요 원인 = 많은 양의 IMG Data 변환 / 저장

I/O 병목 현상 (I/O Bottleneck)

컴퓨터 시스템의 성능 또는 속도가 **입출력 장치(Disk/SSD)의 Data 전송 속도에 의해 제한**을 받는 현상입니다.
병목 현상으로 인하여 **CPU 사용률은 낮고, Disk - 대기열 - Delay 시간**은 높아지지게 되어 **학습이 오래 걸리고 중단되는 ISSUE**가 발생했었습니다.

1. 로컬에 저장

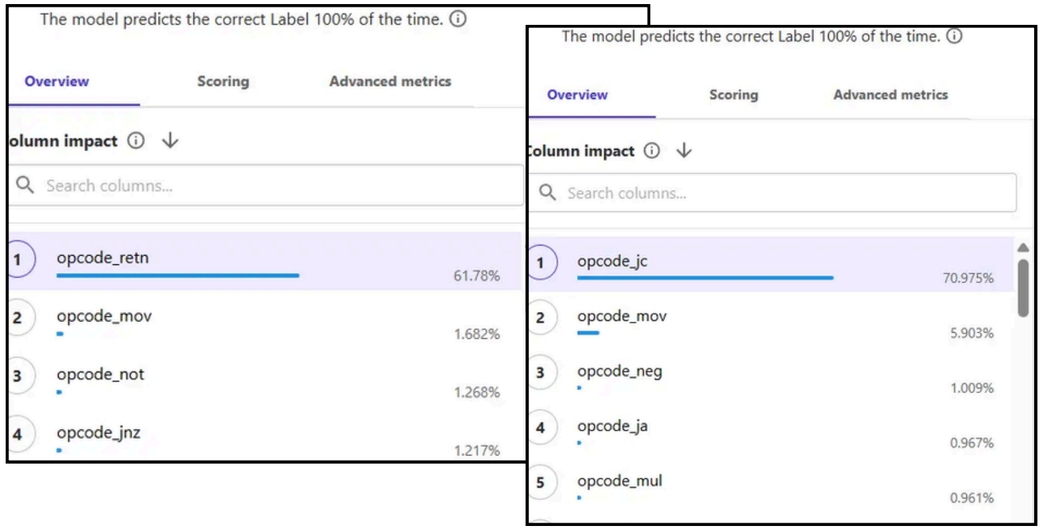
로컬에 변환한 IMG 파일들을 Drive에 Zip 파일 형태로 저장하여 병목현상을 효과적으로 줄였습니다.

```
54 # 300x300 PNG 저장 위치 (로컬에 생성)
55 IMG_ROOT = os.path.join(OUT_ROOT, "Images_288")
56 os.makedirs(IMG_ROOT, exist_ok=True)
```

```
# 2. 이미지 폴더 압축 및 이동 (I/O 쓰로틀링 방지)
# 이미지 폴더 압축
try:
    shutil.make_archive(os.path.join(OUT_ROOT, "Images_288"), 'zip', root_dir=OUT_ROOT, base_dir="Images_288")
    zip_path = os.path.join(OUT_ROOT, "Images_288.zip")
    shutil.move(zip_path, os.path.join(FINAL_DRIVE_PATH, "Images_288.zip"))
    print(f"Images_288.zip 파일을 Drive로 이동 완료.")
    shutil.rmtree(IMG_ROOT) # 로컬의 원본 이미지 폴더 삭제
except Exception as e:
    print(f"[ERROR] 이미지 압축 및 이동 실패: {e}")

print(f"[SUCCESS] 모든 전처리 산출물 (CSV, Images_288.zip)이 Drive: {FINAL_DRIVE_PATH}에 저장되었습니다.")
```

Opcode 단일 명령어 기반 피처 노이즈

문제 발생 개요

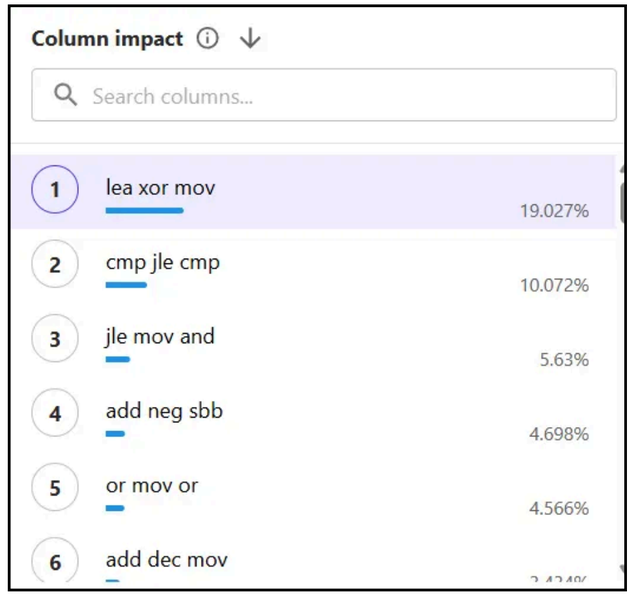


- 1-gram 즉 단일 명령어 빈도 기반 피처 사용 시 문제 발생
- retn, ret 등의 함수 호출 후 반환 주소, jc와 같은 분기 관련 명령어 관련 피처가 치트피처로 작용하여 모델 학습을 왜곡함

개선 및 해결

상위 15개 피처:

	Feature	Importance
7	add push push	0.282537
40	jc mov jmp	0.225234
15	add mov sub	0.111941
5	jmp lea push	0.100818
17	shr and and	0.060003
19	jle xor mov	0.037661
30	call fxch add	0.030859
20	jnb test jz	0.019519
22	add mov pusha	0.018851
39	mul movzx test	0.016781
9	xor shl add	0.016575
18	imul cmp jnz	0.014292
6	or xor or	0.013236
4	mov xor stosb	0.005644
21	mov std cld	0.005301



1. 단일 명령어 단위의 피처로는 명령어 간의 문맥적 연관성을 반영하기 어렵다고 판단하여, n-gram 크기를 조정하며 반복 실험을 수행
2. 2-gram에서는 연속성은 잡히지만 잡음이 많았고, 3-gram에서 가장 뚜렷한 특징 분포와 안정적인 성능을 확인함.
3. 최종적으로 3-gram 기반 TF-IDF를 적용하여 opcode 간의 의미 있는 패턴을 반영하도록 개선함.

하이퍼파라미터 최적화

문제 발생 개요

Image 또는 OPCode 결과가 시간 내에 예측하지 못하는 경우,
PE+IMG, PE+OPC 조합에 대한 베이즈 기반 소프트보팅 최적화 필요

문제 **PE+IMG+OPC 조합은 높은 성능을 보이지만,
PE+IMG 또는 PE+OPC 조합은 성능이 급격히 저하된다**

1) 결합 수식의 구조적 특성

```
combined_logit = bias_logit + k * sum(wi * li for wi, li in zip(weights, logits))
```

세 모델(PE, IMG, OPC) 모두 있을 때 → logit 합이 커져 “증거 강화 효과” 발생
두 모델만 있을 때 → logit 합의 스케일이 작아져 분별력이 약화
=> 구조적으로 모델 수가 많을수록 확신도가 높아지도록 설계되어 있었으며,
이로 인해 PE+IMG+OPC 조합만 유리하게 작동했다.

2) 스케일 불균형 문제 — 특히 OPC

OPC 모델의 확률 분포는 [0.0, 1.0]에 가까운 극단적인 형태를 보였고,
다른 모델(PE, IMG)보다 logit 스케일이 3~5배 높았다.
=> PE+OPC 결합 시 OPC의 과신이 전체 결합을 지배했고,PE 신호가 묻힘

개선 및 해결

1) 스케일 조정 및 개별 모델 스케일 탐색

```
# 모델별 scale 독립 (특히 opc 낮게)
scale_pe = trial.suggest_float("scale_pe", 0.6, 2.5)
scale_img = trial.suggest_float("scale_img", 0.8, 2.0)
scale_opc = trial.suggest_float("scale_opc", 0.3, 1.2)
```

모델별 스케일 파라미터를 분리하여 탐색하도록 변경
특히 OPC 모델은 좁은 범위(0.3~1.2)로 제한해 과신을 억제

2) Logit Clipping 도입

```
# logit 변환 + clipping (OPC 과신 방지용)
def logit(p: float, clip=3.0) -> float:
    p = min(max(p, 1e-8), 1 - 1e-8)
    l = math.log(p / (1 - p))

    return max(min(l, clip), -clip)
```

OPC 모델의 극단적인 logit 값이 결합을 왜곡하지 않도록 clipping 적용
⇨ 모델 간 확률 분포의 과도한 차이를 완화하고 안정성 향상

3) Bias 및 k 탐색 범위 확장

```
k = trial.suggest_float("k", 0.4, 2.0)
bias_logit = trial.suggest_float("bias_logit", prior_bias - 2.0, prior_bias + 2.0)
```

logit 합 범위가 좁은 2모델 조합에서도 보정이 충분히 작동하도록
bias_logit과 k 탐색 범위를 확장
⇨ 결합 스케일 보정이 유효하게 작동하며, FNR 개선 효과 관찰

4) 탐색 안정화를 위한 정규화 강화

```
reg = (0.01 * sum(wi * wi for wi in w) +
       0.02 * (k - 1.0) ** 2 +
       0.02 * (bias_logit - prior_bias) ** 2)
return obj + reg
```

PE 가중치가 과도하게 커지는 현상을 방지하기 위해
정규화 항을 강화
특정 모델에 대한 가중치 집중 현상 완화, FNR 균형 유지.

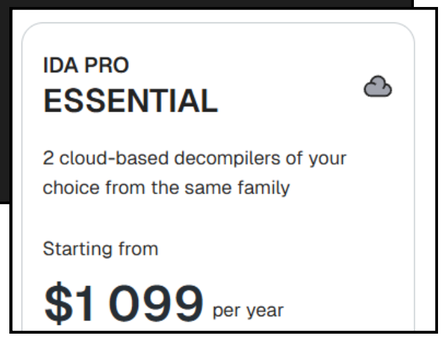
해결

- **조합 간 성능 격차 해소**
- **False Negative 감소 및 평균 정확도 안정화**
- **PE 중심 편향 완화로 균형 잡힌 앙상블 구조 확립**

IDA Pro 사용 제약

문제 발생 개요

```
;
; +-------------------------------------------------------------------------+
; |   This file has been generated by The Interactive Disassembler (IDA)    |
; |             Copyright   (c) 2013 Hex-Rays, <support@hex-rays.com>       |
; |                          License info:                                  |
; |                                        Microsoft                        |
; +-------------------------------------------------------------------------+
;


        .686p
        .mmx
        .model flat
```



문제점

- 악성ASM 파일들은 IDA Pro로 변환되어 있는 데이터셋
- IDA Pro는 유료 라이선스 필요
- 따라서 분석 환경을 구축하는데 제약 발생

개선 및 해결

Ghidra • 무료 오픈소스 디스어셈블러

```
; Program : CompatTelRunner.exe
; Lang    : x86:LE:64:default
; AddrSz  : 8
; --- auto-generated by export_asm.py ---

140001000: CC                      INT3

; ===== Function FUN_140001008 @ 140001008 =====
140001008: 40 55                   PUSH RBP
14000100a: 48 8D 6C 24 F9          LEA RBP, [RSP + -0x7]
14000100f: 48 81 EC 00 01 00 00    SUB RSP, 0x100
140001016: 48 8B 05 23 65 02 00    MOV RAX, qword ptr [0x140027540]
14000101d: 48 33 C4                XOR RAX, RSP
140001020: 48 89 45 F7             MOV qword ptr [RBP + -0x9], RAX
140001024: 48 8B 45 7F             MOV RAX, qword ptr [RBP + 0x7f]
140001028: 4C 8D 1D CD CE 01 00    LEA R11, [0x14001defc]
14000102f: 4C 8B D2                MOV R10, RDX
140001032: 45 33 C9                XOR R9D, R9D
140001035: 48 83 CA FF             OR RDX, -0x1
140001039: 4C 8B 00                MOV R8, qword ptr [RAX]
14000103c: 4D 85 C0                TEST R8, R8
14000103f: 74 15                   JZ 0x140001056
140001041: 48 8B C2                MOV RAX, RDX
140001044: 48 FF C0                INC RAX
140001047: 45 38 0C 00             CMP byte ptr [R8 + RAX*0x1], R9B
14000104b: 75 F7                   JNZ 0x140001044
14000104d: B9 01 00 00 00          MOV ECX, 0x1
140001052: 03 C1                   ADD EAX, ECX
140001054: EB 0A                   JMP 0x140001060
```

자동화

- 윈도우 배치 파일
- 파이썬 스크립트 제작

```
Decompiler Switch Analysis                    4.827 secs
Decompiler Switch Analysis - One Time         3.241 secs
Demangler Microsoft                           0.160 secs
Disassemble Entry Points                      2.564 secs
Embedded Media                                0.024 secs
External Entry References                     0.000 secs
Function ID                                   1.451 secs
Function Start Search                         0.024 secs
Non-Returning Functions - Discovered          0.163 secs
Non-Returning Functions - Known               0.012 secs
PDB Universal                                 0.004 secs
Reference                                     0.304 secs
Scalar Operand References                     0.949 secs
Shared Return Calls                           0.226 secs
Stack                                         5.192 secs
Subroutine References                         0.162 secs
Subroutine References - One Time              0.013 secs
Windows x86 PE Exception Handling             3.739 secs
Windows x86 PE RTTI Analyzer                  0.076 secs
Windows x86 Thread Environment Block (TEB) Analyzer     0.017 secs
WindowsResourceReference                      0.594 secs
x86 Constant Reference Analyzer               4.685 secs
-----------------------------------------------------
Total Time   63 secs
-----------------------------------------------------
```

해결책

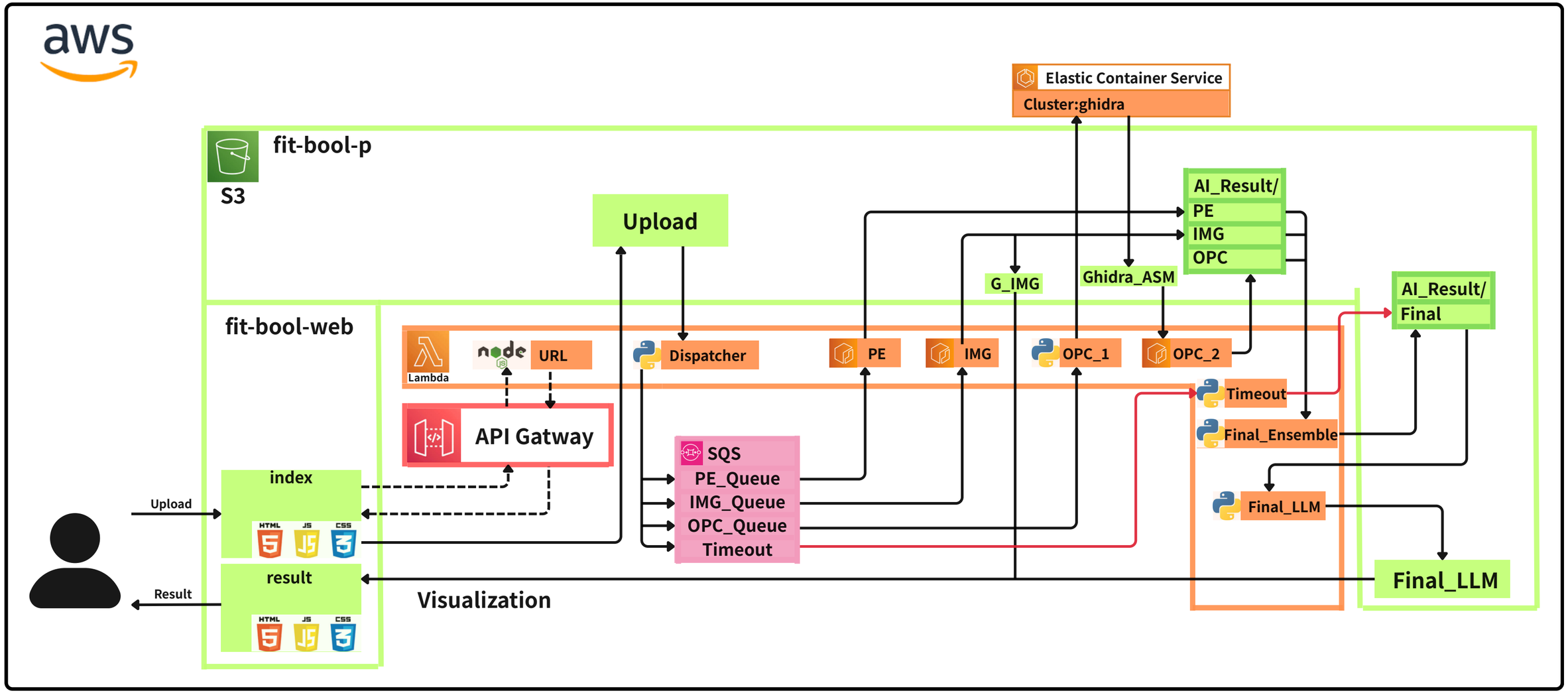
- Ghidra를 사용하여 비용 부담 없이 유사한 분석 환경을 구축
- 대규모 악성 코드의 자동 디스어셈블 및 Opcode 추출을 효율적으로 수행할 수 있었다. (약 4000개 + 변환)

06 AWS

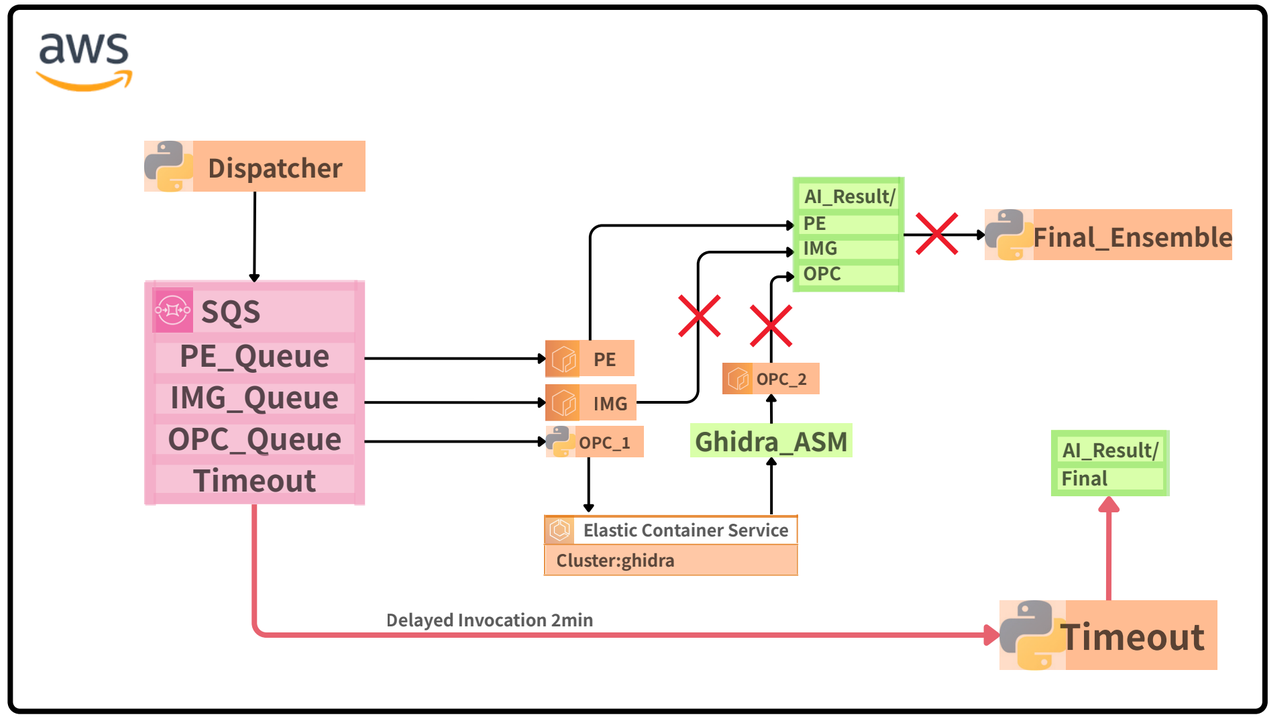
- AWS 아키텍쳐 구조도
- AWS Bedrock

어떻게 활용했는가,

- **S3** → 범용 버킷을 이용하여 FE, 결과 data 저장 및 이벤트 발생을 이용한 람다 호출 담당
- **Lambda** → S3 이벤트 기반 자동 실행, 분석 서비스 결과 처리 담당
- **ECR** → AI 모델, 전처리, 결과 출력, 오픈 소스 도구 Docker 이미지를 저장
- **ECS** → Java 기반 오픈소스 도구를 담은 Docker 이미지를 컨테이너로 실행 담당
- **Bedrock** → LLM 모델 호출 및 요약 결과 생성 지원
- **SQS** → 메시지 큐 기반 병렬 처리로 PE, IMG, OPC Lambda 비동기적 호출
- **Gateway** → FE 파일 업로드 시 S3 업로드 URL 경로 전달 담당

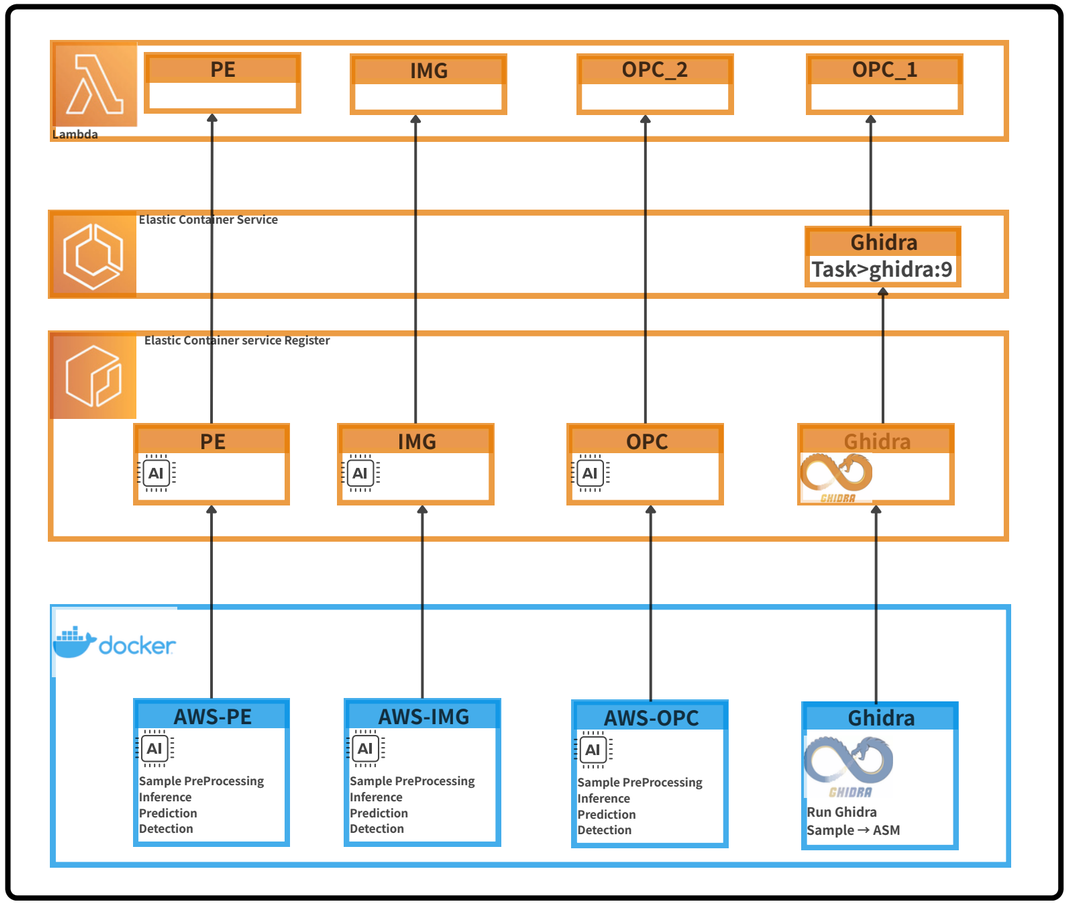

aws
Elastic Container Service
Cluster:ghidra
fit-bool-p
S3
Upload
AI_Result/
PE
IMG
OPC
G_IMG
Ghidra_ASM
AI_Result/
Final
fit-bool-web
Lambda
URL
Dispatcher
PE
IMG
OPC_1
OPC_2
Timeout
API Gatway
SQS
PE_Queue
IMG_Queue
OPC_Queue
Timeout
Final_Ensemble
index
Final_LLM
Upload
result
Result
Visualization
Final_LLM

실행흐름 및 컨테이너 구조



예측 실패, 지연 시 타임아웃 람다로 흐름을 재조정하는 파이프라인

- SQS Delayed Invocation 2분 설정
- 2분뒤 최종 앙상블 결과가 존재하지 않으면 저장된 모델 결과만 이용하여 앙상블 진행



Docker & AWS를 이용한 컨테이너 기반 자동 분석 아키텍쳐

- 대용량 라이브러리 및 AI 모델을 단일 컨테이너 워크 플로우로 처리
- Java기반 오픈 소스 리버스 엔지니어링 도구인 Ghidra를 클러스터로 실행

AWS Bedrock 활용 결과 해석 자동화

하나의 JSON과 고정된 MARKDOWN 템플릿만 사용해 재현성과 감사성(AUDITABILITY)을 확보

확률 임곗값과 신중한 표현을 강제해 단정을 피하며, 일관된 리스크 커뮤니케이션 유지

PE·OPCODE·이미지 간 상충을 명시하고 원인 후보를 제시

가중치·서브모델 확률·핵심 피처 공개 및 체크리스트 제공

Amazon Bedrock

※ Claude Haiku 3 모델 사용

```
## 주요 PE 피처 근거

- **문자열/인코딩/암호화 관련**: strings_entropy↑(6.57) → 내장 데이터/인코딩/암호화 아티팩트 가능성
- **DLL/Imports 관련**: imports_total, import_dlls_unique, imports_max_per_dll=0 → 소수 DLL에 호출 집중, 특정 라이브러리 의존/래핑 가능성
- **헤더/섹션/이미지 관련**: SizeOfCode↑(29696), SizeOfInitializedData↑(489984), NumberOfSections=6 → PE 구조 이상 가능성, 패킹/난독화 징후
- **YARA/패킹 플래그**: yara_has_*=0 → 난독화/패킹 징후 미탐지

## 주요 Opcode 근거

- `mov mov mov`(0.32) → 반복적인 레지스터 이동, 초기화/루프 패턴 가능
- `push call mov`(0.21) → 함수 호출 직전 스택/레지스터 조작, 복잡한 제어 흐름 시사
- `push push push`(0.19) → 스택 기반 데이터 준비, 함수 호출 전 단계 가능
- `call mov mov`(0.15) → 연속 호출 및 레지스터 이동, 제어 흐름 복잡성 반영
- `push mov call`(0.12) → 스택+레지스터 조합, API 호출 패턴 가능

## 이미지 근거(바이너리 시각화)

- 이미지 기반 확률: 77.12%, 보조 지표(entropy=1.04)
- 요약: 보조 근거임을 명시. 대표 패밀리 이미지는 유사성 비교용(확률 제공 없음)

## 전체 요약 / 권고

- 종합 판단:
  PE, Opcode, 이미지 모두 악성으로 수렴하여, 해당 파일은 86.03% 확률로 악성으로 판단됩니다.
  문자열/DLL/Opcode 분석 결과 등 주요 근거를 종합하면 악성코드 의심 정황이 강합니다.

- 운영 권고:
  - [ ] 격리 및 백업
  - [ ] 추가 정적 분석
  - [ ] 샌드박스 동적 분석
  - [ ] 네트워크 차단/IoC 조회
  - [ ] VirusTotal 업로드(https://www.virustotal.com/gui/home/upload)
```

※ 베드락 프롬프트 결과

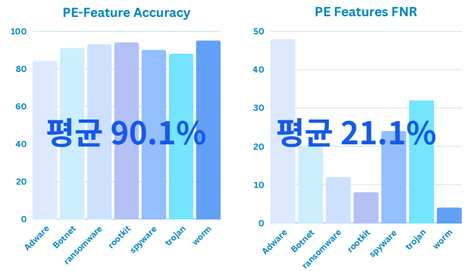
03 IMPACT

제품 성과측정 및 기대효과

- 융합 모델의 안정성
- 기대 효과
- 향후 개선 방안

앙상블 모델의 효과

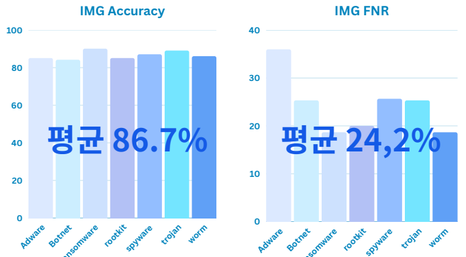
PE



PE

- **Accuracy**
  - **90% 이상**의 매우 안정적인 기본 탐지력
- **FNR 탐지**
  - **Worm / Rookit / Ransomware** 특화
- **“안정적”** 기본 Model
- 실제 배포 환경에서의 **Backbone** 활용에 적합

IMG

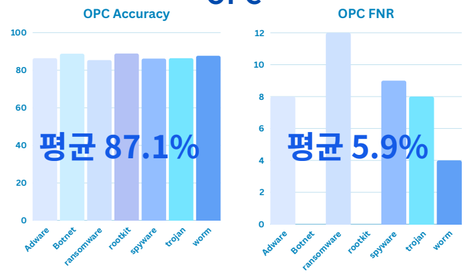


IMG의 정확도와 미탐율 성능 그래프

IMG

- **Accuracy**
  - 86.7% 성능
  - **“변형, 패킹, 난독화“**에 특화된 Model
- **FNR 탐지**
  - PE, OPC Model에서 놓친 **비정형 탐지**에 우수

OPC

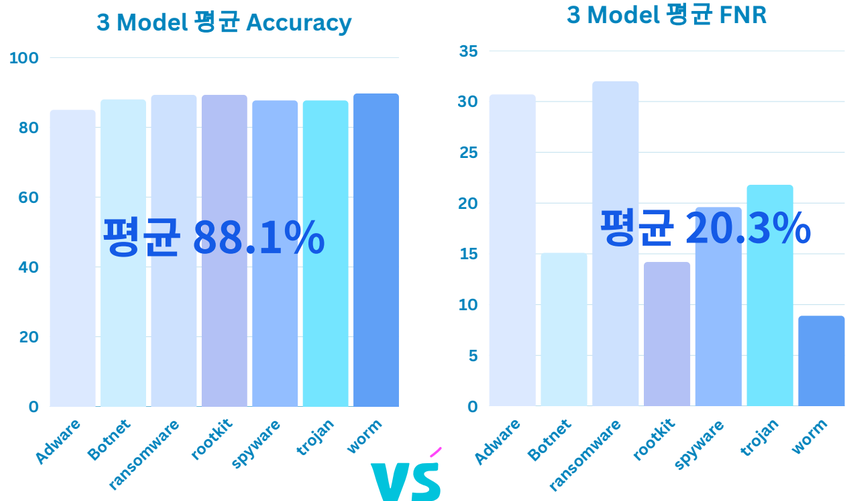


OPC의 정확도와 미탐율 성능 그래프

OPC

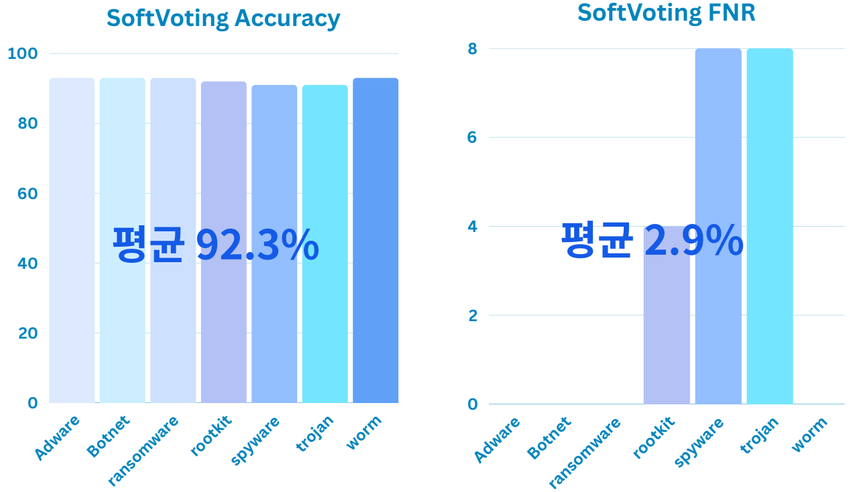
- **Accuracy**
  - 87.1% 성능
  - **작은 편차 + 일관성** 있는 Model
- **FNR 탐지**
  - **매우 낮은 미탐율**을 가진 안정형 Model
  - **탐지 안정성**이 **우수한 “놓치지 않는“** Model

3가지 모델 단순 평균

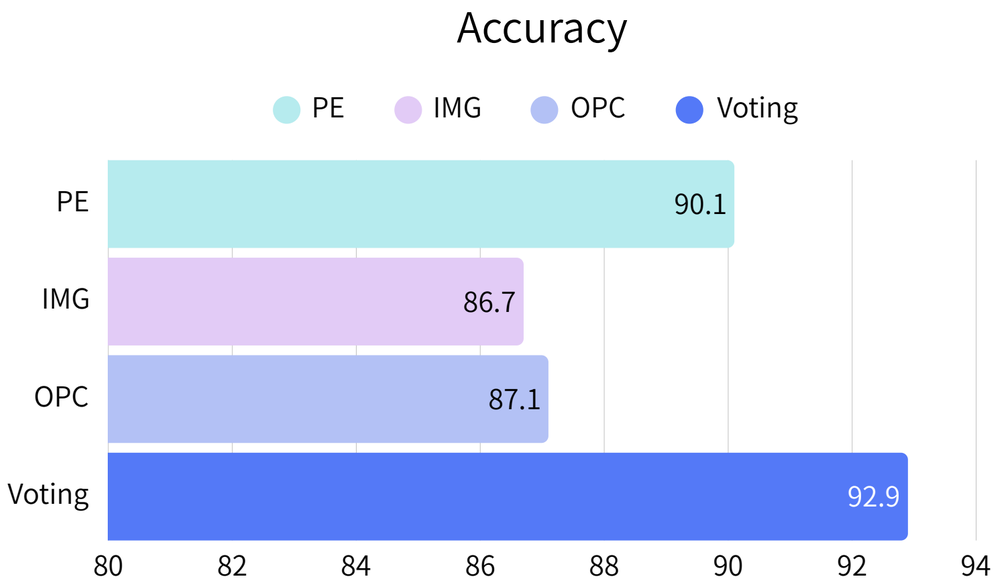


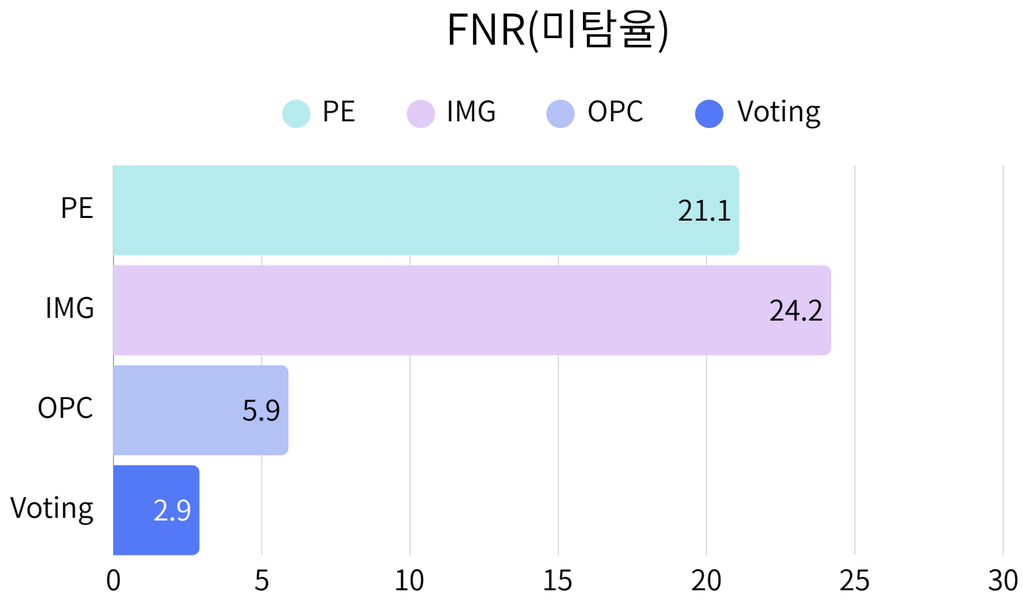
VS

Baysian Log-Odds Soft Voting지표



모델 설명력





‘Voting’ 핵심 성과

- **안정적 성능 확보**
  - 단일 Model의 **단점 극복**
  - **성능편차 최소화**
  - **안정적 탐지** 성능
- **최고의 신뢰도**
  - **Voting** > PE, IMG, OPC
  - **가장 높은 Accuracy**
  - **가장 낮은 FNR**

‘앙상블’의 효과

- **Accuracy**
  Voting > PE > OPC > IMG
  - **비교적 낮은 OPC, IMG의 Accuracy를 Voting을 통해 극복**
- **Voting을 통한 단일 Model들의 극대화**
  - **PE의 안정적인 ‘기본 탐지력’**
  - **IMG의 ‘변형, 패킹, 난독화 탐지’**
  - **OPC의 놓치지 않는 ‘철저한 탐지’**

상세 정보 제공

파일 정보

File Information

ID	9ba7edff-4d3d-4d84-b62c-e1169a8e2799
MD5	9b95ceaf0383581afb69fa92a4e97f35
SHA1	b7c20de0defa22bdb03630ff9429781c63b0646e
SHA256	855363e7ca77c49e9fd83f614d7b9054b2a536a0b2998c59e873c9
SHA512	25c1cd40b40d8bea75434c4c0b67253b9a8c639b93dbad77cfe4e2 da96df392d7d8ce2ab99e7e4ca60136dba4e10
Filename	jupyter.exe
Size	108413
FileType	application/x-msdownload

분석 대상 파일의 기본 정보
(해시값, 파일 크기, 유형 등)
정보의 투명성을 확보합니다.

PE 분석 결과

PE Extraction Result

file	s3://fit-bool-p/Upload/jupyter.exe
prediction.label	0
prediction.prob	0.0007676669047214091
prediction.prob_percent	0.08%
DllCharacteristics	33088
SizeOfStackReserve	1048576
AddressOfEntryPoint	17020

PE 모델이 도출한 결과 Table
PE 모델에서 추출한
Feature 값들을 확인할 수 있습니다.

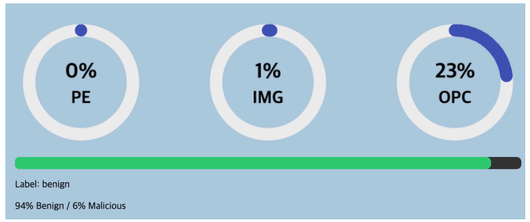
OPC 분석 결과

OPC Extraction Result

file	jupyter.asm
prediction.label	0
prediction.prob	0.2330322249626429
prediction.prob_percent	23.30%
features.opcode_count	15391
features.trigram_count	15389

OPC 모델이 도출한 결과 Table
각 명령어의 3-gram TF-IDF 점수
Feature 값들을 확인할 수 있습니다.

분석 결과



최종 분석 결과 : 시각화
Model 별 판별 %
종합 악성/정상 %
시각화를 통해 한눈에 분석 결과를 파악
할 수 있습니다.

근거 설명

악성 확률: 6.50% / 최종 판정: benign

- 소프트 보팅 가중치: PE=0.37, Opcode=0.267, Image=0.363
- 서브모델 예측: PE=0.08% / Opcode=23.30% / Image=0.69%
- 모델 간 일치도: 일부 불일치 → 불확실성 주석
- 회색구간 여부: 아니오

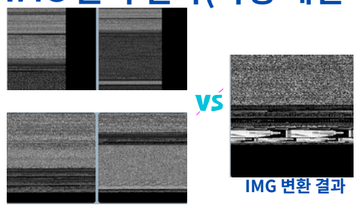
주요 PE 피처 근거

- 사용 피처: DllCharacteristics, SizeOfStackReserve, AddressOfEntry

더보기

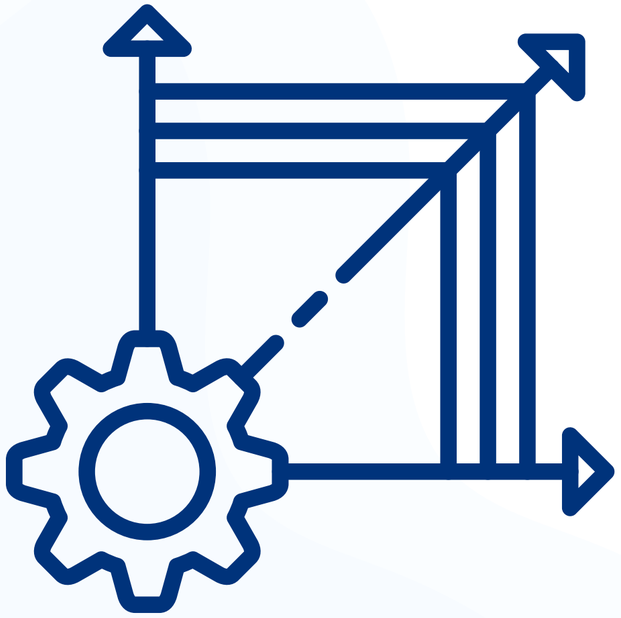
근거 설명 - BedRock prompt
모델 결과에 대한 상세 결과 출력과
각 모델에서 왜 악성(정상)으로 판단했
는지 상세 이유를 확인할 수 있습니다.

IMG 분석 결과(악성 패밀리 뷰어)



악성 패밀리 뷰어
업로드한 파일의 이미지와 패밀리 간의시각적 비교를 통해
어떤 유형의 패밀리 코드인지
보조탐색할 수 있습니다.

확장성 & 유연성



01 **교체 & 추가**
Model 의 교체
New Model & Convert 용이성 확보

02 **발전 가능성**
Model & Docker 기반 구성
향후 발전 가능성 확보

03 **Docker**
Docker 기반 Model / Logic 분리
Model & Logic의 모듈화 + 컨테이너화

04 **AWS**
Cloud 기반 인프라
다양한 기술과 높은 확장성을 지원

실 서비스 운영 방안

기존 아키텍쳐

- 서버리스 & 자동화 구조 구현
- 독립된 이벤트 단위의 분산 설계
- 프로젝트 핵심 프로토타입

실제 사용자 제공 서비스 구현

- **Cloud Front** → FE 데이터가 저장된 S3 실제 접속 가능한 사이트로 적용
- **VPC** → 웹페이지, 샘플 업로드 S3는 외부망, 나머지 모든 서비스는 내부망으로 분리
- **DynamoDB** → 기존 분석 결과 저장 공간은 S3 → DynamoDB로 옮겨 내부망에 저장
- **ECS** → Applicatin Auto Scaling을 이용한 클러스터 테스크 병렬 처리
- **SQS** → DynamoDB 이벤트 발생 → 람다 출력 사이에 추가하여 안정적인 병렬 처리 지원
- **Gateway** → 외부망에 존재하는 결과 출력 페이지에 내부망에 존재하는 결과 데이터 출력
- **Cloud Watch, Etc** → 관리자 로그 분석 / 모니터링 체계 설정
- **Dual Analysis Options** → 다른 정밀 분석용 모델 개발로 빠른 분석 & 정밀 분석 지원

향후 개선 방안

PE	pefile, lief 라이브러리 특징 추출의 한계	패커 식별 라이브러리(PEiD, Detect It Easy 등)와 자동 언패커 도구(upx, unipacker등) 활용
IMG	Grey scale로 변환한 png	더 다양한 정보를 담을 수 있도록 IMG 변환 방법 개선 → 특성 기반 3채널 RGB 인코딩
OPC	OPCode 3-gram TF-IDF	API호출 내역을 통한 정밀분석 쿠쿠 샌드박스를 연결하여, 동적 분석까지
융합	7가지 패밀리, 각 100개의 데이터셋으로 최적의 가중치 산출	정상 데이터셋 추가 수집 후, 가중치 재탐색

04 HOW

팀 활동 요약

- 우리가 집중한 부분
- 3개월 간의 일정
- 활동한 방식
- 팀 소개

Operate as One Team

01 **다양한 데이터셋 학습과 변환**
각각 5,000개 이상의 데이터 및 다양한 데이터 셋 사용
pe: exe → json | img: exe → png | opcode: exe → asm

02 **효과적인 역할분배**
3가지 단일모델: 메인/서브
메인 개발자가 모델 개발에 집중하면서 서브개발자와 피드백 교류

03 **미탐을 줄이자**
실 상황에서는 정상이 악성보다 압도적 다수
악성 데이터를 정상으로 예측하지 않도록

04 **자주, 많이**
매주 온오프라인을 병행한 회의와 멘토링까지
팀원들간의 소통 격차를 줄이기 위해 노력

5 Keyword Our Team

재미있는

성장하는

문제해결적인

끈기있는

소통하는

on going

Your Canva profile name won’t be shared

3개월 간의 일정

Weekly Tracker

WHAT WE DID EVERY WEEK

- → 주제 탐색, 선정 주제 심화 분석, 데이터셋 확인, 주제 재선정, 관련 논문 분석
- → 총 7주간 수집 (악성 json, 악성 img, 악성 asm , 악성 exe, 정상 exe)
- → 5~7주차환경구상, 시나리오 작성 | 8 ~10주차 환경구축 | 11주차 BedRock
- → exe to json 변환, poc, 과적합 처리, 모델 개발, 성능 향상, 최종 학습
- → exe to png 변환, poc, 모델개발(cnn), 모델개발(efficientnet_v2), 최종학습
- → 쿠쿠 샌드박스 시도, IDA 시도, Ghidra 성공, 정상exe to asm변환, asm 3-gram 추출, 모델 개발, 최종학습
- → 업로드 화면, 분석화면, 모델별 결과 , BedRock, 이미지 출력

활동 방식

지역을 넘나드는 미팅

멤버들의 거주지를 반영한
7번의 오프라인 미팅

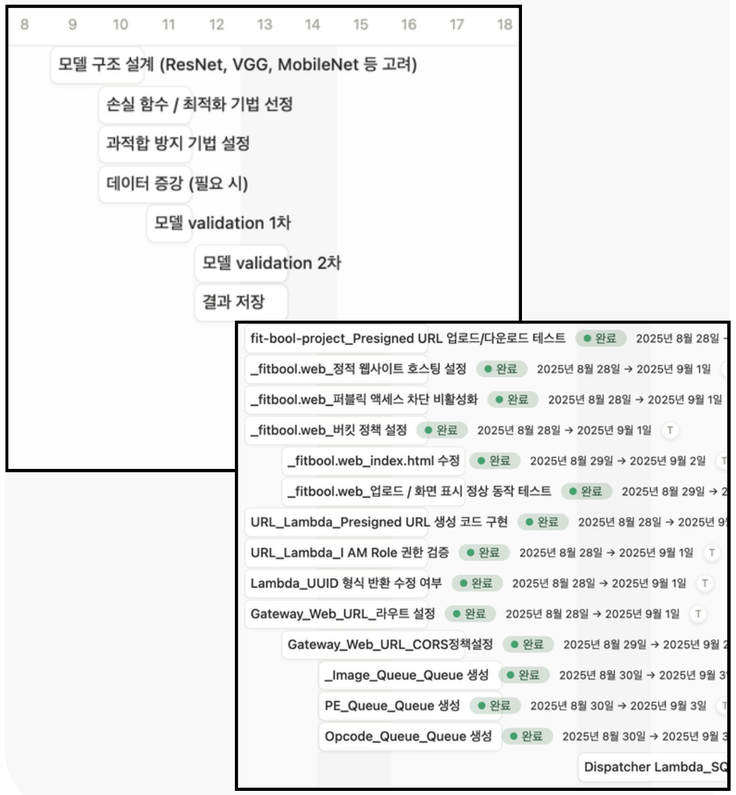
- 2025/07/21 송파
- 2025/07/25 송파
- 2025/08/05 가산
- 2025/08/12 가산
- 2025/09/09 용산
- 2025/09/23 천안
- 2025/09/29 신사

Daily Report

교육 기간 이후, 8월 간 매일 아침 보고
31일 중 30일 완료

WBS

학기 중, WBS 체크
100가지의 세부작업 중 99가지 완료



팀 소개

홍태경
OpCode
BedRock

이도협
AWS
OpCode

강민성
IMG

김민수
PE 정적분석
BedRock

김서현
SoftVoting
Front End
IMG
PE

We are Fit Bool!

악성코드엔 핏불처럼.

우린 한 번 문 악성코드는 절대 놓지 않는다!

Fit-Bool Notion

https://www.notion.so/AI-Fit-bool-Fit-Bool-Page-Guider-23122f1948db809f959ae9609b1d211a

Fit-Bool Github

https://github.com/AISecurityForMalware-FitBool/Multi-Model-softVoting

APPENDIX

- **PE 구조분석 모델**
- **IMAGE 분석 모델**
- **Opcode 분석 모델**
- **Ensemble**
- **AWS**
- **FRONT END**

피처 추출 함수 일부

```
def _extract_strings(file_bytes: bytes) -> Dict[str, float]:
 strings = re.findall(rb"[\x20-\x7e]{4,}", file_bytes)
 if strings:
 printable_len = sum(len(s) for s in strings)
 return {
 "strings_entropy": _entropy(b"".join(strings)),
 "strings_printable_ratio": (printable_len / len(file_bytes)) if file_bytes
else 0.0,
 "strings_avg_len": (printable_len / len(strings)) if strings else 0.0,
    "strings_base64_blob_count":    len(re.findall(rb'[A-Za-z0-9+/=]{20,}',
file_bytes)),
 }
 else:
 return {
 "strings_entropy": 0.0,
 "strings_printable_ratio": 0.0,
 "strings_avg_len": 0.0,
 "strings_base64_blob_count": 0,
 }
```

```
def _extract_imports_pefile(file_bytes: bytes) -> Dict[str, float]:
 all_imports, dlls, imports_per_dll = [], set(), []
 try:
 pe2 = pefile.PE(data=file_bytes, fast_load=True)
 # 임포트 디렉터리 명시 파싱(환경/버전 차이 방지)
 try:
 pe2.parse_data_directories(directories=[
 pefile.DIRECTORY_ENTRY["IMAGE_DIRECTORY_ENTRY_IMPORT"]
 ])
 except Exception:
 pass
 if hasattr(pe2, "DIRECTORY_ENTRY_IMPORT"):
 for entry in pe2.DIRECTORY_ENTRY_IMPORT:
 name = (entry.dll or b"").decode("utf-8","ignore").lower()
 if name: dlls.add(name)
 cnt = 0
 for imp in entry.imports:
 if imp.name:
 all_imports.append(imp.name.decode("utf-8","ignore")); cnt += 1
 if cnt > 0: imports_per_dll.append(cnt)
 try: pe2.close()
 except Exception: pass
 except Exception:
 pass
 return {
 "imports_total": len(all_imports),
 "imports_unique": len(set(all_imports)),
 "import_dlls_unique": len(dlls),
 "imports_max_per_dll": max(imports_per_dll) if imports_per_dll else 0,
 "imports_entropy": _entropy(''.join(all_imports)) if all_imports else 0.0,
 }
```

피처 추출 함수 일부

```
def _extract_headers_pefile(file_bytes: bytes) -> Dict[str, float]:
    feats: Dict[str, float] = {}
    pe = None
    try:
        # 풀 파싱 고정
        pe = pefile.PE(data=file_bytes, fast_load=False)
        DOS  = getattr(pe, "DOS_HEADER", None)
        FILE = getattr(pe, "FILE_HEADER", None)
        OPT  = getattr(pe, "OPTIONAL_HEADER", None)
        g = lambda obj, attr, d=-1: getattr(obj, attr, d) if obj else d
        feats.update({
            "DllCharacteristics": g(OPT,"DllCharacteristics"),
            "MajorImageVersion": g(OPT,"MajorImageVersion"),
                                            "MajorOperatingSystemVersion":
g(OPT,"MajorOperatingSystemVersion"),
            "SizeOfStackReserve": g(OPT,"SizeOfStackReserve"),
            "AddressOfEntryPoint": g(OPT,"AddressOfEntryPoint"),
            "Characteristics": g(FILE,"Characteristics"),
            "SizeOfHeaders": g(OPT,"SizeOfHeaders"),
            "SizeOfInitializedData": g(OPT,"SizeOfInitializedData"),
            "SizeOfUninitializedData": g(OPT,"SizeOfUninitializedData"),
            "MinorSubsystemVersion": g(OPT,"MinorSubsystemVersion"),
            "ImageBase": g(OPT,"ImageBase"),
            "MajorLinkerVersion": g(OPT,"MajorLinkerVersion"),
            "NumberOfSections": g(FILE,"NumberOfSections"),
            "MinorImageVersion": g(OPT,"MinorImageVersion"),
            "SizeOfStackCommit": g(OPT,"SizeOfStackCommit"),
```

```
"e_lfanew": g(DOS,"e_lfanew"),
     "e_minalloc": g(DOS,"e_minalloc"),
     "e_ovno": g(DOS,"e_ovno"),
     "Machine": g(FILE,"Machine"),
     "PointerToSymbolTable": g(FILE,"PointerToSymbolTable"),
     "NumberOfSymbols": g(FILE,"NumberOfSymbols"),
     "SizeOfCode": g(OPT,"SizeOfCode"),
     "BaseOfCode": g(OPT,"BaseOfCode"),
     "SectionAlignment": g(OPT,"SectionAlignment"),
     "FileAlignment": g(OPT,"FileAlignment"),
    })
except Exception:
for k in _HEADER_KEYS: feats[k] = -1
finally:
if pe:
try: pe.close()
except Exception: pass
return feats
```

학습 결과

Model	ROC-AUC	PR-AUC	Accuracy	Precision	Recall	F1-score	Specificity	Size
RandomForest	0.993391	0.993631	0.963	0.9601	**0.9717**	0.9659	0.9556	87.8MB
ExtraTrees	0.992967	0.994209	**0.9634**	0.9623	0.97	**0.9661**	0.9582	97.3MB
XGBoost	**0.993681**	**0.994259**	0.9623	**0.9631**	0.965	0.964	**0.9593**	**2.3MB**

1. PE 바이트 → 300×300 이미지 변환 (전처리)

```
# EfficientNetV2-S 최종 입력 크기
IMG_SIZE = 300
def width_by_size(n_bytes: int) -> int:
    # 파일 크기에 기반하여 이미지 너비(W)를 결정하는 휴리스틱.
  if n_bytes < 1024*1024:
    # 파일 크기에 따라 32, 64, 128, 256, 384, 512, 768 중 하나를 반환
     return next(w for s, w in [(10, 32), (30, 64), (60, 128), (100, 256), (200, 384), (500,
512), (1024, 768)] if n_bytes < s * 1024)
  return 1024

def exe_bytes_to_gray_square(bytes_data: bytes, target=IMG_SIZE) -> np.ndarray:

# 바이트를 W x H 그레이스케일 배열로 변환 (패딩 포함)
  N = len(bytes_data)
  W = width_by_size(N)
  H = math.ceil(N / W)

  arr = np.frombuffer(bytes_data, dtype=np.uint8)
  if len(arr) < H * W:
    arr = np.pad(arr, (0, H * W - len(arr)), constant_values=0)

# 정사각형으로 패딩.
  img = arr.reshape(H, W)
  side = max(H, W)
  sq = np.zeros((side, side), dtype=np.uint8)
  sq[:H, :W] = img  # 정사각형 중앙 패딩 대신 좌상단에 배치

  # target x target 크기로 리사이징.
  return cv2.resize(sq, (target, target), interpolation=cv2.INTER_AREA)
```

2. PNG 파일 저장 (전처리 결과 저장)

```
def convert_to_png(df_eligible, out_img_root, img_size):
  # eligible 파일 목록을 순회하며 바이트를 PNG로 변환 후 저장
  for _, r in df_eligible.iterrows():
    out_path = Path(out_img_root) / f"{r['sha256']}.png"

    # 파일 읽기 및 변환
    with open(r["path"], "rb") as f: data = f.read()
    gray = exe_bytes_to_gray_square(data, target=img_size)

    # PNG 저장
    cv2.imwrite(str(out_path), gray)
```

3.데이터 증강(선택적) 및 EfficientNetV2 전처리 적용)

```
def pipeline(ds, training=False):
  # 데이터 증강(선택적) 및 EfficientNetV2 전처리 적용

  # 1. 파일 로드, 디코딩, 리사이징 후 캐싱 (핵심 최적화)
  ds = ds.map(load_and_preprocess, num_parallel_calls=AUTOTUNE).cache()

  # 2. 훈련 시 증강 적용 (data_augment 레이어 사용)
  if training:
    # data_augment(x, training=True) 적용 (코드 생략)
    pass

  # 3. EfficientNetV2 모델 전용 전처리
  ds = ds.map(lambda x,y: (preprocess_input(x), y), num_parallel_calls=AUTOTUNE)

  return ds.prefetch(AUTOTUNE) # 프리페칭 적용
```

모델학습

```
def make_ds_from_manifest(fold_idx, subset, local_img_root):

  # Manifest CSV를 읽어 tf.data.Dataset을 생성
  csv_path = os.path.join(OUT_ROOT, f"manifest_split_fold_{fold_idx+1}.csv")
  if not os.path.exists(csv_path):
    raise FileNotFoundError(f"Manifest file not found: {csv_path}")

  df = pd.read_csv(csv_path)
  # eligible하고 해당 split에 속하는 파일만 필터링
  df_subset = df[(df["eligible"] == 1) & (df["split"] == subset)].reset_index(drop=True)

  # Local Unzip 경로 + sha256.png I/O 병목 현상 감소 핵심
  paths = [os.path.join(local_img_root, f"{sha256}.png") for sha256 in df_subset["sha256"]]
  labels = df_subset["label"].values.astype(np.float32)

  if not paths:
    print(f"[WARN] No {subset} data found in Fold {fold_idx+1}.")
    return None, 0

  ds = tf.data.Dataset.from_tensor_slices((paths, labels))
```

```
  # 이미지 변환 (디코딩 및 리사이징) 핵심 로직
  # 3채널(RGB)로 디코딩
  # IMG_SIZE x IMG_SIZE로 리사이징
  # 모델 입력 타입으로 변환
  def load_and_preprocess(file_path, label):
    image = tf.io.read_file(file_path)
    image = tf.io.decode_png(image, channels=3)
    image = tf.image.resize(image, [IMG_SIZE, IMG_SIZE])
    image = tf.cast(image, tf.float32)
    return image, label

  ds = ds.map(load_and_preprocess, num_parallel_calls=AUTOTUNE)

  # 핵심 최적화: 캐싱 추가 (디코딩 및 리사이징 결과를 메모리에 저장)
  ds = ds.cache()

  if subset == "train":
    ds = ds.shuffle(buffer_size=len(paths), seed=SEED)

  ds = ds.batch(BATCH_SIZE)
  return ds, len(df_subset)
```

2가지 모델 비교

Model	ROC-AUC	PR-AUC	Accuracy	Malware F1-Score	Normal F1-Score
EfficientNetB0	0.984	0.9865	0.9195	0.9278	0.9091
EfficientNetV2-S	**0.9809**	**0.9788**	**0.9354**	**0.94**	**0.92**

학습 결과

Model	**ROC-AUC**	**PR-AUC**	**Accuracy**	**F1-Score**	**Precision**	**Recall**	**Specificity**
EfficientNetV2-S	**0.9891**	**0.9968**	**0.9492**	**0.9657**	**0.9933**	**0.9392**	**0.9803**

Opcode 추출 코드 일부

```
import re
from tqdm.auto import tqdm

# --- Opcode 추출 함수 정의 ---
def extract_opcodes_from_asm(file_path, whitelist):
    """
    IDA 또는 Ghidra 형식의 .asm 파일에서 Whitelist에 포함된 Opcode를 추출합니다.
    """
    ida_regex = re.compile(r'^\s*\.[a-zA-Z0-9]+:[0-9a-fA-F]+\s+(?:[0-9a-fA-F]{2}\s+)*\s*([a-zA-Z]+)\s*.*')
    ghidra_regex = re.compile(r'^[0-9a-fA-F]+:\s+(?:[0-9a-fA-F]{2}\s+)*\s*([a-zA-Z]+).*')

    opcodes = []
    with open(file_path, 'r', encoding='utf-8', errors='ignore') as f:
        for line in f:
            match = ida_regex.match(line) or ghidra_regex.match(line)
            if match:
                opcode = match.group(1).lower()
                if opcode in whitelist:   # 지정된 Opcode만 추출
                    opcodes.append(opcode)
    return ' '.join(opcodes)


# --- 모든 파일에서 Opcode 시퀀스 추출 ---
print("Extracting opcodes from all files...")

opcodes_corpus = [
    extract_opcodes_from_asm(f, OPCODE_WHITELIST)
    for f in tqdm(all_files, desc="Parsing files")
]

print("Extraction complete.")
```

피처 추출 코드 일부

```
from sklearn.feature_extraction.text import CountVectorizer

# --- 1. 학습 데이터에서 Trigram 피처 추출 ---
vectorizer = CountVectorizer(ngram_range=(3, 3))
X_train_counts = vectorizer.fit_transform(X_train)
feature_names = vectorizer.get_feature_names_out()

# --- 2. 악성 / 정상 인덱스 분리 ---
mal_idx = [i for i, y in enumerate(y_train) if y == 1]
ben_idx = [i for i, y in enumerate(y_train) if y == 0]

# --- 3. 공통 상위 20개 Trigram ---
total_counts = X_train_counts.sum(axis=0).A1
top_common = {f for f, _ in sorted(
    zip(feature_names, total_counts), key=lambda x: x[1], reverse=True
)[:20]}

# --- 4. 악성 전용 15개 / 정상 전용 15개 피처 선택 ---
def get_top_unique_features(indices, exclude_set, top_n=15):
    counts = X_train_counts[indices].sum(axis=0).A1
    ranked = sorted(zip(feature_names, counts), key=lambda x: x[1], reverse=True)
    feats = []
    for f, _ in ranked:
        if f not in exclude_set:
            feats.append(f)
        if len(feats) >= top_n:
            break
    return set(feats)

mal_features = get_top_unique_features(mal_idx, top_common, 15)
ben_features = get_top_unique_features(ben_idx, top_common | mal_features, 15)
```

```
# --- 5. 피처 통합 및 치트키 제거 ---
cheat_keys = {"retn sub sub", "pop ret mov", "pop pop ret"}
final_features = list((top_common | mal_features | ben_features) - cheat_keys)
print(f"최종 선택된 피처 수: {len(final_features)}")
```

학습 결과

Model	Accuracy	Precision	Recall	F1-Score	ROC-AUC	PR-AUC
LightGBM	**0.9712**	**0.9621**	**0.9823**	**0.9721**	**0.9952**	**0.9958**
Random Forest	0.968	0.9575	0.9807	0.969	0.9947	0.995
XGBoost	0.968	0.9604	0.9775	0.9689	0.9942	0.995
Linear SVM	0.9121	0.8693	0.9742	0.9188	0.9747	0.9635
Logistic Regression	0.9096	0.8676	0.971	0.9164	0.9757	0.9745
Multinomial NB	0.8841	0.8488	0.9404	0.8923	0.9407	0.9353

모든 지표에서 가장 높은 성능을 보인 LightGBM 사용

Baysian Log-odds Soft Voting 가중치 탐색 추출 코드 일부

1) 유틸리티 함수

```
def load_results(folder):
    data = {}
    for fp in glob.glob(os.path.join(folder, "*", "*.json")):
        js = json.load(open(fp)); data[js["hashes"]["sha256"]] = js
    return data

def logit(p, clip=3.0):
    p = min(max(p, 1e-8), 1-1e-8)
    return max(min(math.log(p/(1-p)), clip), -clip)

def sigmoid(x): return 1/(1+math.exp(-x))
```

2) 결합 및 평가

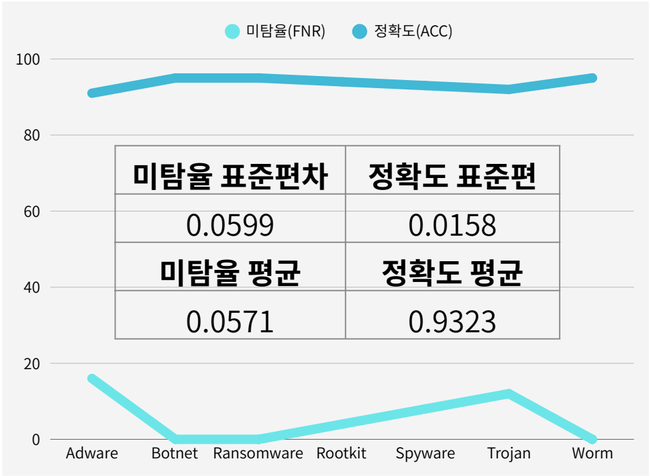
```
def evaluate(pe,img,opc,t=0.5,w=(1,1,1),mode="evidence",k=1.0,bias=0.0):
    rows=[]
    for sha,v in pe.items():
        fam=v["family"].split("_")[0].lower(); probs=[]; ws=[]
        for src,wi in zip([pe,img,opc],w):
            if sha in src and wi>0: probs.append(src[sha]["prediction"]["prob"]); ws.append(wi)
        if not probs: continue
        p=combine_probs(probs,ws,mode,k,bias)
        rows.append({"family":fam,"pred":int(p>=t),"prob":p})
    df=pd.DataFrame(rows)
    res=[]
    for f in [f for f in df.family.unique() if f!="ben"]:
        sub=df[df.family.isin([f,"ben"])].copy(); sub["label"]=(sub.family==f).astype(int)
        acc=accuracy_score(sub.label,sub.pred)
        tn,fp,fn,tp=confusion_matrix(sub.label,sub.pred).ravel()
        fnr=fn/(tp+fn) if tp+fn>0 else 0
        res.append({"acc":acc,"fnr":fnr})
    s=pd.DataFrame(res)
    return {"std_acc":s.acc.std(),"std_fnr":s.fnr.std(),
            "mean_acc":s.acc.mean(),"mean_fnr":s.fnr.mean()}
```

3) Optuna 최적화

```
def compute_prior_bias(pe):
    p=sum(v["family"].split("_")[0].lower()!="ben" for v in pe.values())/len(pe)
    return logit(p)

def obj_fn(m,a=1,b=0.3,g=0.7,d=0.2):
    return a*m["std_fnr"]+b*m["std_acc"]+g*m["mean_fnr"]-d*m["mean_acc"]

def optuna_objective_factory(pe,img,opc,mode="evidence"):
    bias0=compute_prior_bias(pe)
    def obj(trial):
        w=[trial.suggest_float(f"w_{x}",0,1) for x in ["pe","img","opc"]]
        s=[trial.suggest_float(f"scale_{x}",lo,hi)
           for (x,lo,hi) in [("pe",0.6,2.5),("img",0.8,2.0),("opc",0.3,1.2)]]
        w=[wi/ sum(w)*si for wi,si in zip(w,s)]
        k=trial.suggest_float("k",0.4,2.0)
        bias=trial.suggest_float("bias",bias0-3,bias0+3)
        t=trial.suggest_float("t",0.2,0.8)
        m=evaluate(pe,img,opc,t,w,mode,k,bias)
        return obj_fn(m)+0.01*sum(wi**2 for wi in w)
    return obj

def run_optuna(pe,img,opc,name="pe+img+opc"):
    study=optuna.create_study(direction="minimize")
    study.optimize(optuna_objective_factory(pe,img,opc),n_trials=200)
    print(f"Best({name})",study.best_params)
```

활성 모델 별 학습 결과

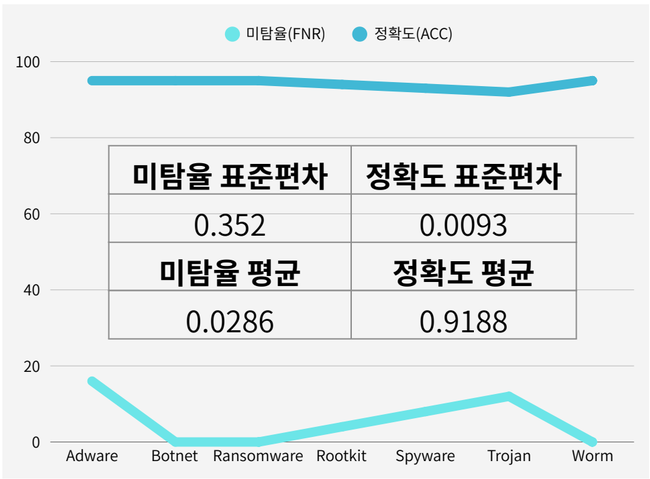
PE+IMG+OPC



미탐율 표준편차	정확도 표준편
0.0599	0.0158
미탐율 평균	**정확도 평균**
0.0571	0.9323

하이퍼 파라미터	값
W_PE	0.418998112937013
W_IMG	0.201554043026697
W_OPC	0.101200998008704
K	1.066
bias_logit	0.544
Threshold	0.325

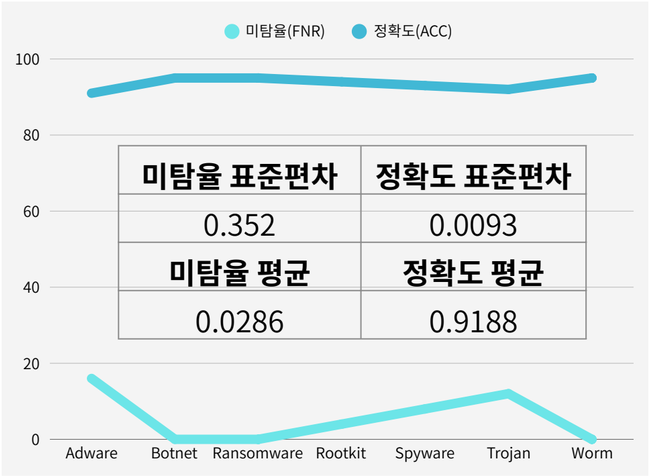
PE+IMG



미탐율 표준편차	정확도 표준편차
0.352	0.0093
미탐율 평균	**정확도 평균**
0.0286	0.9188

하이퍼 파라미터	값
W_PE	0.374908367758468
W_IMG	0.345771217902336
W_OPC	0
K	1.07
bias_logit	0.679
Threshold	0.38

PE+OPC



미탐율 표준편차	정확도 표준편차
0.352	0.0093
미탐율 평균	**정확도 평균**
0.0286	0.9188

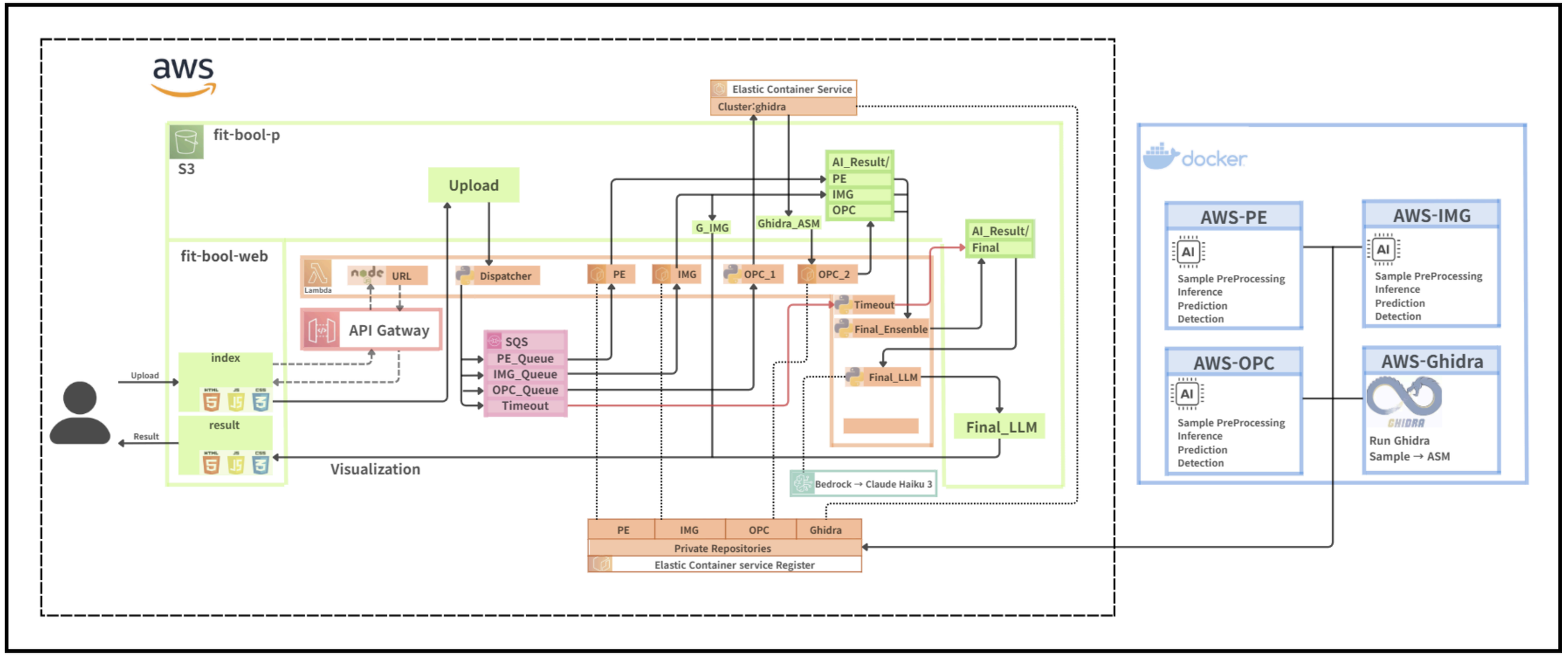
하이퍼 파라미터	값
W_PE	0.31276200066339
W_IMG	0
W_OPC	0.263543863317283
K	1.048
bias_logit	0.34
Threshold	0.357

베드락 전체 프롬프트

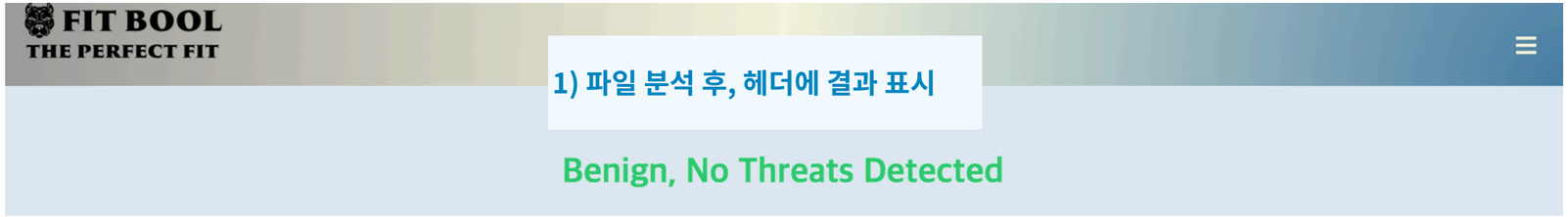
```
당신은 전문 보안 분석가입니다. 아래 JSON은 동일 파일에 대해 3개 모델(PE 정적 피처, Opcode, 바이너리 이미지) 결과와
소프트 보팅(가중치 합산) 최종 결과를 담고 있습니다.
이 JSON만을 근거로, **모델 결과의 근거와 불확실성**을 명확히 설명하는 xAI 스타일의 요약 리포트를 작성하세요.

# 작성 원칙
- 과도한 확정 표현 금지: "가능성이 있다/의심된다/정황상" 표현을 사용.
- 허위 정보/추정 금지: JSON에 없는 사실을 지어내지 말고, 과해석하지 말 것.
- 수치 표기: 확률은 소수 2자리까지 %로 표기.
- 일관된 Markdown 섹션을 반드시 사용할 것(아래 출력 포맷 준수).
- 모델 간 **불일치(disagreement)**, **회색구간(gray zone)**을 반드시 포함.
- 악성 혹은 회색 구간으로 판단 시 **운영 권고 체크리스트**를 반드시 포함.(정상일 경우에는 포함X)
- 사용자가 쉽게 이해할 수 있도록 최대한 깔끔하고 조리있게 작성할 것

# 불확실성 규칙
- 최종 악성 확률(final_prob) 기준:
- 고위험(> 70%): 적극 조치 권고.
- 회색구간(40% ~ 60%): 보수적 판단, 추가 분석 권장. (※ 반드시 "회색구간" 문구 포함)
- 저위험(< 30%): 정상 가능성이 높음. 단, 서브모델 불일치·의심 피처 있으면 보수적 권고 유지.
- 서브모델 간 불일치:
- 두 개 이상 모델이 서로 상충할 경우: "모달리티 간 신호 불일치"로 명시하고 원인 후보(피처 민감도/모달리티 차이)를 간단 기술.
- 이 경우 최종 판단 문단에서 **불확실성↑**을 강조하고, 격리·추가 분석을 명확히 권고. 바이러스 토탈 등 교차 검증 사이트 제공할 것

# 보안적 해석(가이드라인)
- PE 피처:
- strings_entropy↑ + base64_blob_count↑: 내장 데이터/인코딩/암호화 아티팩트 가능성(단정 금지).
- imports_total vs import_dlls_unique: 소수 DLL에 호출 편중 → 특정 라이브러리 의존/래핑 가능성.
- imports_max_per_dll↑: 특정 DLL 호출 집중(프록시/래핑) 가능성.
- YARA 패커 플래그(UPX/mpress/aspack 등) 1: 패킹/난독화 가능성. 0: 패킹 징후 미검출(비악성 보장 아님).
- 피처 분석을 토대로 해당 실행파일이 할 가능성이 있는 행위 등일 반드시 제시할 것
- Opcode:
- n-gram은 통계적 특징일 뿐 의미 과해석 금지. 분기/호출/검사 패턴 증가는 “상대적 경향”으로만 기술.
- 피처 분석을 토대로 해당 실행파일이 할 가능성이 있는 행위 등일 반드시 제시할 것
- 이미지(바이너리 시각화):
- 보조 신호임을 인지하고 있으나, 어떠한 근거로 악성/정상을 판단했는지 간략히 서술할 것(패밀리 유사성 비교 용도)
- 대표 패밀리 이미지는 참고용이며 확률 제공 없음.
- 종합:
- 세 가지의 분석결과를 토대로 해당 실행파일이 어떤 행위를 할 지 가능성을 제시하고 그에 따른 대응 방안을 간략하게 제시할 것

# 출력 포맷 (Markdown)
# 악성 확률: {{final_prob%}} / 최종 판정: {{malicious|benign}}
- 소프트 보팅 가중치: PE={{w_pe}}, Opcode={{w_opcode}}, Image={{w_image}}
- 서브모델 예측: PE={{pe_prob%}} / Opcode={{op_prob%}} / Image={{img_prob%}}
- 모델 간 일치도: (예: 모두 정상으로 수렴 / 일부 불일치 → 불확실성 주석)
- 회색구간 여부: (해당 시 명시)
```

```
## 주요 PE 피처 근거
- 사용 피처(값이 존재하는 것만 표시, 수치도 같이 표시) :
DllCharacteristics, SizeOfStackReserve, AddressOfEntryPoint, Characteristics,
SizeOfHeaders, SizeOfInitializedData, SizeOfUninitializedData, SizeOfStackCommit,
SizeOfCode, BaseOfCode, SectionAlignment, FileAlignment, ImageBase,
PointerToSymbolTable, NumberOfSymbols, imports_total, imports_unique,
import_dlls_unique, imports_max_per_dll, strings_avg_len, strings_base64_blob_count,
e_minalloc, e_ovno, MinorImageVersion, MajorImageVersion,
MajorOperatingSystemVersion, MinorSubsystemVersion, MajorLinkerVersion,
NumberOfSections, Machine, imports_entropy, strings_entropy,
strings_printable_ratio, yara_has_packer_generic, yara_count_packer,
yara_has_upx_like, yara_has_mpress_like, yara_has_aspack_like, e_lfanew

- 요약 작성 규칙:
  1. **문자열/인코딩/암호화 관련**
  - strings_entropy ↑, strings_printable_ratio ↓, strings_base64_blob_count ↑ → 내장 데이터/인코딩/암호화 아티팩트 가능성
  2. **DLL/Imports 관련**
  - imports_total, import_dlls_unique, imports_max_per_dll → 소수 DLL에 호출 집중 시 특정 라이브러리 의존/래핑 가능성
  - imports_entropy ↑ → 호출 패턴 다양성/난독화 가능성
  3. **헤더/섹션/이미지 관련**
  - SizeOfCode, SizeOfInitializedData, SizeOfUninitializedData, NumberOfSections, SectionAlignment → PE 구조 이상 여부, 패킹/난독화 징후 가능성
  - ImageBase, FileAlignment, BaseOfCode → 메모리 레이아웃 특이점 가능성
  4. **YARA/패킹 플래그**
  - yara_has_* 플래그 1 → 난독화/패킹 가능성
  - yara_count_packer → 패킹 탐지 핫수
  5. **기타 PE 헤더**
  - DllCharacteristics, AddressOfEntryPoint, e_lfanew 등 → 일반적 구조와 차이 발생 시 의심

- 작성 지침:
  - 피처 수치를 기반으로 3~5문장 정도로 “가능성 중심” 기술
  - 단정적 판단 금지, 예: “~ 가능성”, “~ 의심”, “정황상 ~”
  - TF-IDF처럼 수치가 높은 피처는 중요성을 강조 가능
  - 필요 시 “이 피처 단독으로 악성/정상 판단 불가” 명시
  - 판단한 근거들로 일어날 수 있는 실행파일의 행위를 반드시 제공할 것

## 주요 Opcode 근거
- 대표 n-그램 상위 5개 (TF-IDF 점수 기준):
예시)- `mov mov mov` (0.32) → 반복적인 레지스터 이동, 초기화/루프 패턴 가능
    - `push call mov` (0.21) → 함수 호출 직전 스택/레지스터 조작, 복잡한 제어 흐름 시사
    - `push push push` (0.19) → 스택 기반 데이터 준비, 함수 호출 전 단계 가능
    - `call mov mov` (0.15) → 연속 호출 및 레지스터 이동, 제어 흐름 복잡성 반영
    - `push mov call` (0.12) → 스택+레지스터 조합, API 호출 패턴 가능
- 요약: 각 n-그램은 TF-IDF 기반 통계적 중요도를 나타냅니다.
      반복 이동, 스택/레지스터 조작 등은 악성코드에서 흔히 관찰되는 패턴이지만, 통계적 특징이므로 단정적 판단은 어렵습니다.
- opcode 분석 결과가 없다면 : "Opcode 분석 결과가 존재하지 않아, 본 항목은 판단에 반영되지 않았습니다." 라고 출력
```

```
## 이미지 근거(바이너리 시각화)
- 이미지 기반 확률: {{img_prob%}}, 보조 지표(있으면 간략히)
- 요약: 보조 근거임을 명시하나 간략하게 판단 근거를 제시할 것. 대표 패밀리 이미지는 유사성 비교용(확률 제공 없음)
- 이미지 분석 결과가 없다면 : "이미지 분석 결과가 존재하지 않아, 본 항목은 판단에 반영되지 않았습니다." 라고 출력

## 전체 요약 / 권고
- 종합 판단(한 문단): 최종 확률, 서브모델 일치/불일치, 핵심 근거 요약.
- 운영 권고(체크리스트 형태로 그대로 나열):
- [ ] 격리 및 백업
- [ ] 추가 정적 분석
- [ ] 샌드박스 동적 분석
- [ ] 네트워크 차단/IoC 조회
- [ ] VirusTotal 업로드(https://www.virustotal.com/gui/home/upload)
```


aws
Elastic Container Service
Cluster:ghidra
fit-bool-p
S3
Upload
AI_Result/
PE
IMG
OPC
G_IMG
Ghidra_ASM
AI_Result/
Final
fit-bool-web
Lambda
node
URL
Dispatcher
PE
IMG
OPC_1
OPC_2
Timeout
Final_Ensenble
API Gatway
SQS
PE_Queue
IMG_Queue
OPC_Queue
Timeout
Final_LLM
index
result
Upload
Result
Visualization
Final_LLM
Bedrock → Claude Haiku 3
PE
IMG
OPC
Ghidra
Private Repositories
Elastic Container service Register
docker
AWS-PE
AI
Sample PreProcessing
Inference
Prediction
Detection
AWS-IMG
AI
Sample PreProcessing
Inference
Prediction
Detection
AWS-OPC
AI
Sample PreProcessing
Inference
Prediction
Detection
AWS-Ghidra
GHIDRA
Run Ghidra
Sample → ASM

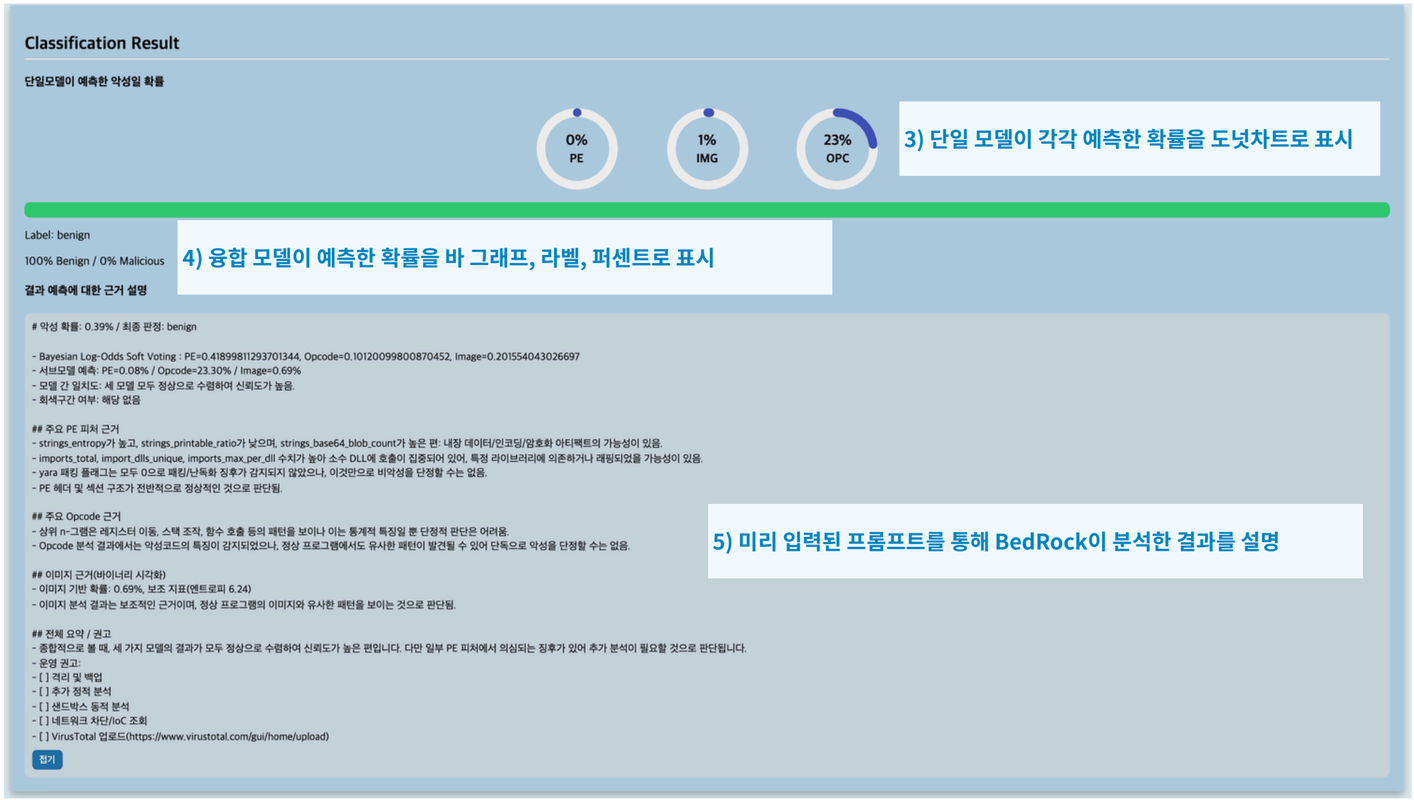

FIT BOOL
THE PERFECT FIT
3가지 융합 모델을 통한 견고한 악성 탐지
exe 파일을 업로드 하면,
pe 특징 추출, 이미지 변환, opcode 추출을 통해 악성 유무를 확인하고,
3가지 모델을 근거로 한 설명과 결과를 알려줍니다.
여러 악성 패밀리의 이미지 사진을 통해,
업로드한 exe는 어느 패밀리에 속하는지도 확인할 수 있습니다.
1) 파일 업로드
드래그 앤 드랍 or fitbool 아이콘 클릭 2가지 방식 구현
jupyter.exe
여기에 exe 파일을 업로드하세요
2) 디자인적 재미 요소
마우스 포인터를 추적하여 header, upload box, 눈동자의 색/위치가 변함



File Information

ID	8b183ed3-1e74-46bf-8c58-1eb24509b82f
MD5	9b95ceaf0383581afb69fa92a4e97f35
SHA1	b7c20de0defa22bdb03630ff9429781c63b0646e
SHA256	855363e7ca77c49e9fd83f614d7b9054b2a536a0b2998c59e873c947f5b894ba
SHA512	25c1cd40b40d8bea75434c4c0b67253b9a8c639b93dbad77cfe4e265fe3279160dc3a8a3b0b10e7b6a182938b1da96df392d7d8ce2ab99e7e4ca60136dba4e10
Filename	jupyter.exe
Size	108413
FileType	application/x-msdownload

2) 파일에 대한 기본 정보, 해시값을 안내



PE Extraction Result

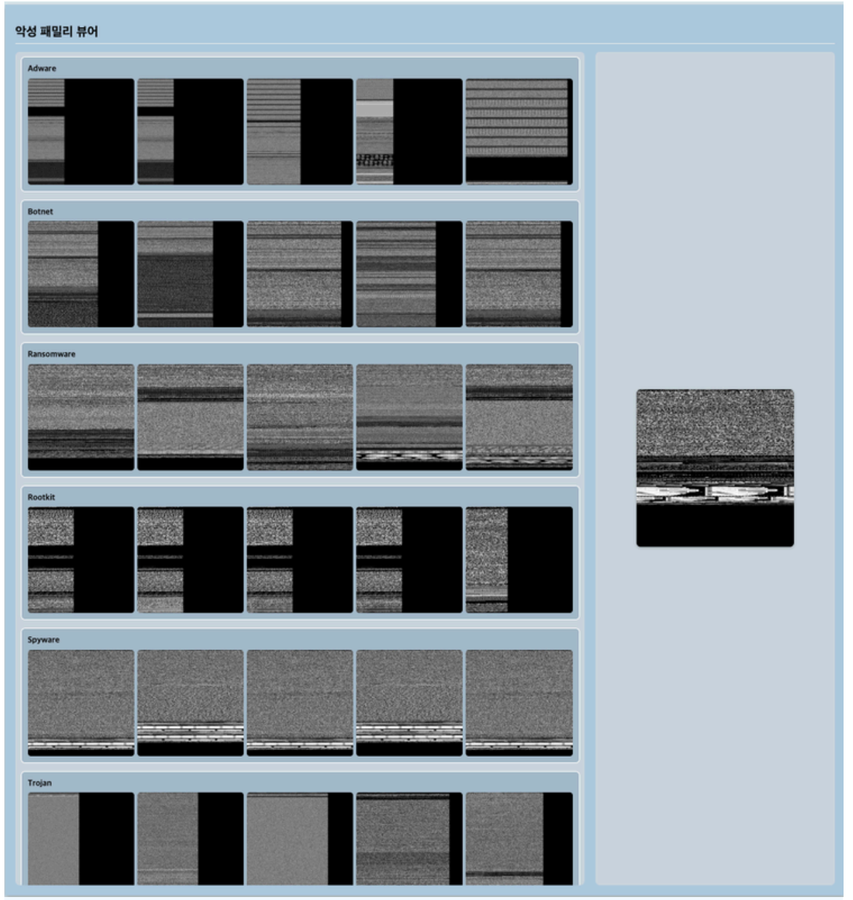
file	s3://fit-bool-p/Upload/jupyter.exe	분석된 파일 경로
prediction.label	0	예측 결과 라벨 (benign=정상, malicious=악성)
prediction.prob	0.0007676669047214091	악성일 확률 (0~1 실수값)
prediction.prob_percent	0.08%	악성일 확률 (%)
DllCharacteristics	33088	PE 헤더의 DLL 특성 플래그 (보안/호환성)
SizeOfStackReserve	1048576	예약된 스택 메모리 크기 (바이트)
AddressOfEntryPoint	17020	실행 시작 지점의 RVA
Characteristics	34	파일 특성 비트 플래그
SizeOfHeaders	1024	헤더 크기 (바이트)
SizeOfInitializedData	45568	초기화된 데이터 크기
SizeOfUninitializedData	0	초기화되지 않은 데이터 크기
SizeOfStackCommit	4096	실제 커밋된 스택 크기
SizeOfCode	61440	코드 섹션 크기
BaseOfCode	4096	코드 섹션 시작 주소
SectionAlignment	4096	메모리 상 섹션 정렬 단위
FileAlignment	512	파일 상 섹션 정렬 단위
ImageBase	5368709120	이미지가 메모리에 로드될 기본 주소
PointerToSymbolTable	0	심볼 테이블 포인터 (디버깅 정보)
NumberOfSymbols	0	심볼 개수
imports_total	86	임포트된 API 총 개수
imports_unique	86	고유 임포트 개수
import_dlls_unique	2	임포트된 DLL 개수
imports_max_per_dll	83	하나의 DLL에서 최대 임포트 수
strings_avg_len	7.289433384379786	문자열 평균 길이
strings_base64_blob_count	34	Base64로 인코딩된 문자열 개수
e_minalloc	0	MS-DOS 헤더 최소 추가 할당
e_ovno	0	MS-DOS 헤더 오버레이 번호
MinorImageVersion	0	PE Minor Image Version
MajorImageVersion	0	PE Major Image Version
MajorOperatingSystemVersion	5	지원 OS Major Version
MinorSubsystemVersion	2	지원 Subsystem Minor Version
MajorLinkerVersion	10	링커 Major Version
NumberOfSections	6	섹션 개수
Machine	34404	대상 아키텍처 (예: 332=Intel 386)
imports_entropy	4.700062784015653	임포트 섹션 엔트로피 (난독화 지표)
strings_entropy	5.78949684077731 1	문자열 엔트로피 (난독화 지표)
strings_printable_ratio	0.08781234722773099	출력 가능한 문자열 비율
yara_has_packer_generic	0	YARA: 제너릭 패커 탐지 여부
yara_count_packer	0	YARA: 패커 탐지 개수
yara_has_upx_like	0	YARA: UPX 패커 탐지 여부
yara_has_mpress_like	0	YARA: MPRESS 패커 탐지 여부
yara_has_aspack_like	0	YARA: ASPACK 패커 탐지 여부
e_lfanew	248	PE 헤더 시작 오프셋

6) PE 정적 분석 시, 추출한 feature 정보 제공

OPC Extraction Result

file	jupyter.asm	분석된 파일 경로
prediction.label	0	예측 결과 라벨 (benign=정상, malicious=악성, indeterminate=판단 불가)
prediction.prob	0.2330322249626429	악성일 확률 (0~1 실수값)
prediction.prob_percent	23.30%	악성일 확률 (%)
features.opcode_count	15391	분석된 전체 Opcode 명령어 수
features.trigram_count	15389	3-그램(Opcode 3개 연속) 추출 개수
features.evidence_count	41	탐지에 사용된 특징 개수
mov mov cmp	0.011085	-
mov lea mov	0.135308	-
mov mov call	0.062619	-
mov jmp mov	0.156347	-
mov mov add	0.002516	-
mov mov mov	0.84552	-
test jz mov	0.045985	-
mov mov lea	0.018726	-
jz mov mov	0.019906	-
mov mov test	0.022882	-
mov call mov	0.43093	-
cmp jnz mov	0.018987	-
jmp mov mov	0.022297	-
call mov mov	0.030664	-
push push push	0.077292	-
pop pop pop	0.088195	-
mov test jz	0.045963	-
lea mov call	0.017494	-
lea call mov	0.009132	-
lea mov mov	0.027871	-
mov push call	0	-
mov imul mov	0.012937	-
mov push mov	0.025	-
push call add	0	-
push push call	0	-
mov mov push	0.01732	-
push push mov	0.00117	-
imul mov mov	0.002039	-
mov push push	0.008562	-
mov mov imul	0.000249	-
mov mul mov	0	-
push mov push	0.031685	-
push mov call	0	-
push call mov	0	-
push mov mov	0	-
test jnz mov	0.017403	-
mov cmp jz	0.021022	-
mov call test	0.026544	-
mov xor mov	0.087084	-
call lea mov	0.01073	-
push sub mov	0.02365	-
lea mov lea	0.12865	-
sub mov mov	0.016552	-
mov add pop	0	-
jnz mov mov	0.00878	-
xor mov mov	0.015582	-
call test jz	0.020711	-
cmp jz mov	0.029804	-

7) Opcode 분석 시, 추출한 feature 정보 제공



8) 왼쪽: 미리 패밀리별로 군집화시킨 이미지를 정렬
오른쪽: 업로드한 파일의 바이너리값을 이미지로 변환

9) 보고서 출력 버튼을 통해, 분석 내용 저장

- 서울경제, 「자산운용사 19곳 '핀번호'까지 털렸다」, 2025-09-22, https://m.sedaily.com/NewsView/2GY038F01T, 열람일: 2025-10-15. 서울경제
- 보안뉴스, 「SGI서울보증보험 서비스 장애, 비상…(랜섬웨어 의심)」, 2025-07-14, https://m.boannews.com/html/detail.html?idx=138136, 열람일: 2025-10-15. 보안뉴스
- DailyDarkWeb, “Killsec Ransomware Attack Hits Cadorim, Top4Fans, UwayApply, …” https://dailydarkweb.net/killsec-ransomware-attack-hits-cadorim-top4fans-uwayapply-scanbo-mortdash-fdb-collections-ax-capital-vtk-legal-behca-fractalite-rainwalk/, 열람일: 2025-10-15.
- 보안뉴스, 「미래엔서해에너지, 랜섬웨어 공격으로 일부 전산 시스템 중단…」, 2025-10-03, https://m.boannews.com/html/detail.html?idx=139651, 열람일: 2025-10-15. 보안뉴스
- YTN, 「충남 도시가스 공급업체, 해킹으로 개인정보 유출 사고」, 2025-10-04, https://www.ytn.co.kr/_ln/0115_202510041511278825, 열람일: 2025-10-15. YTN
- SBS, 「'미래엔서해에너지' 고객 카드·계좌 등 개인정보 유출」, 2025-10-04, https://news.sbs.co.kr/amp/news.amp?news_id=N1008283502, 열람일: 2025-10-15. SBS 뉴스
- 보안뉴스, 「랜섬웨어 조직 건라, 이번엔 코스피 상장사 화천기계 해킹…265GB 데이터 탈취」, 2025-09-10, https://m.boannews.com/html/detail.html?idx=139188, 열람일: 2025-10-15.

- Hasan H. Al-Khshali, Muhammad Ilyas, “Impact of Portable Executable Header Features on Malware Detection Accuracy,” Computers, Materials & Continua, 74(1), 154–169, 2023. https://doi.org/10.32604/cmc.2023.032182.
- P. V. Shijo, A. Salim, “Integrated static and dynamic analysis for malware detection,” Procedia Computer Science, 46, 804–811, 2015. https://doi.org/10.1016/j.procs.2015.02.149.
- Sadia Nazim et al., “Multimodal malware classification using proposed ensemble deep neural network framework,” Scientific Reports, 15, Article No. 18006, 2025. https://doi.org/10.1038/s41598-025-96203-3
- V. Saini, R. Gupta, and N. Soni, “OpCode-Based Malware Classification Using Machine Learning and Deep Learning Techniques,” *arXiv preprint arXiv:2504.13408*, Apr. 2025.

FIT BOOL

THANK YOU

kisia 한국정보보호산업협회
Korea Information Security Industry Association

강민성 김민수 김서현 이도협 홍태경