Pusan National University

Computer Science and Engineering

Technical Report 2023-10

[블록체인 보안]

Smart-Contract 취약점 탐지 툴

저자 1 김윤하

저자 2 윤지원

저자 3 최지원

지도교수 최윤호

목 차

1. 서론

1.1. 연구 배경

블록체인 기술은 투명성, 불변성, 분산화 등의 특징을 가지고 있어 금융, 의료, 로지스틱스 등 다양한 분야에서 활용되고 있다. 이러한 블록체인 기술의 중요한 구성 요소 중 하나가 스마트 컨트랙트이다. 스마트 컨트랙트는 자동으로 실행되는 코드로서, 복잡한 계약 조건을 프로그래밍 언어로 작성하여 블록체인 상에 배포할 수 있다.

그러나 스마트 컨트랙트는 코드의 복잡성과 함께 취약점이 발생할 가능성도 크게 증가한다. 한번 배포된 스마트 컨트랙트는 변경이 불가능하기 때문에, 이런 취약점들은 해커들에 의해 악용될 위험이 크다. 실제로 과거 몇몇 대형 사태들에서 보았듯이(예: The DAO), 이런 취약점들은 사용자들의 자산을 위협하며 전체 시스템의 신뢰도를 저하시킨다.

따라서 스마트 컨트랙트를 안전하게 개발하고 유지하기 위해서는 그것을 체계적으로 검사하고 평가하는 방법이 필요하다. 이 연구에서 우리는 자동화된 도구를 개발하여 스마트 컨트랙트의 취약점을 탐지하는 방법에 대해 탐구한다. 이 도구는 개발자들이 보다 안전한 스마트 컨트랙트를 작성하는데 도움을 줌으로써 전체적인 블록체인 생태계의 안전을 강화하는데 기여할 것으로 기대한다.

1.2. 기존 문제점

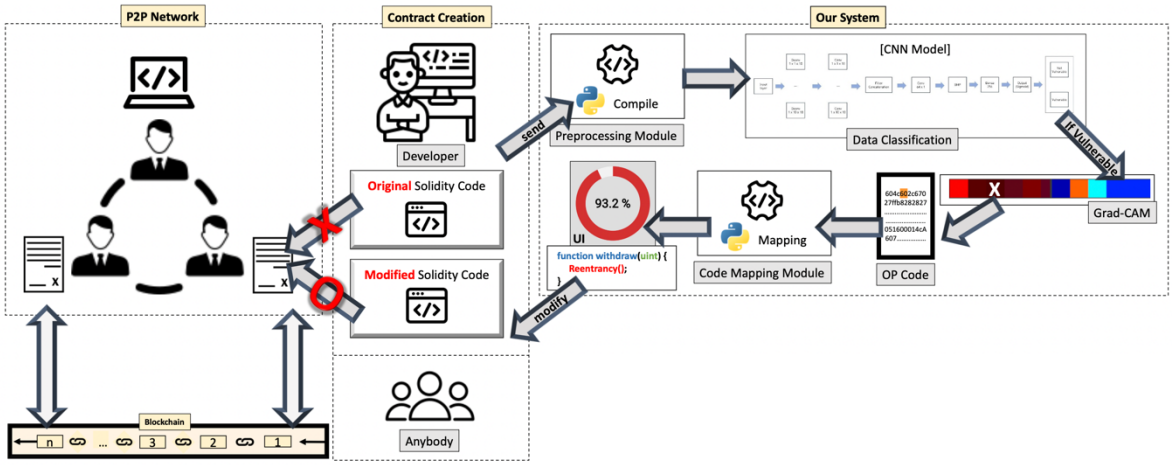
기존에 사용하던 취약점 탐지 도구[5]와 방법에는 여러 가지 문제점이 있었다. 이러한 문제점은 다음과 같다.

- 시간적 소모 : 기존 방법에는 취약점을 탐지하는 작업에 상당한 시간이 소요되었다. 특히 오픈소스 이더리움 가상머신 바이트코드의 보안 분석 도구인 Mythril Classic의 경우 솔리디티 파일 하나당 약 5분의 시간이 소요되었다.
- 유료화 버전의 한정된 기능 : 우리 졸업과제 팀은 초기 데이터 레이블링 작업을 위해 유료 버전의 도구를 사용하려고 했으나, 한정된 기능만 사용할 수 있었다. 또한 MythX의 경우 20,000 개의 데이터 셋을 탐지하는 데에 약 $500의 유료화 버전을 사용해야 했다.
- 취약점 위치 탐지가 부족한 점 : 기존의 취약점 탐지 도구들은 취약점 위치를 충분히 정확하게 식별하지 못하거나 식별하더라도 탐지율이 매우 낮은 경우가 많았

다.

1.3. 연구 목표

본 과제는 스마트 컨트랙트에서 발생하는 취약점을 탐지하기 위해 Static Analysis 기법을 사용하는 취약점 탐지 소프트웨어 개발을 목표로 한다. CNN 모델을 활용하여 스마트 컨트랙트의 Reentrancy(재진입) 취약점을 탐지하고, 이 취약점의 위치를 XAI 모델인 Grad-CAM을 통해 시각적으로 표시하며, 취약점 여부(확률)와 위치를 기존의 Solidity 코드로 표시하는 UI를 제공한다.

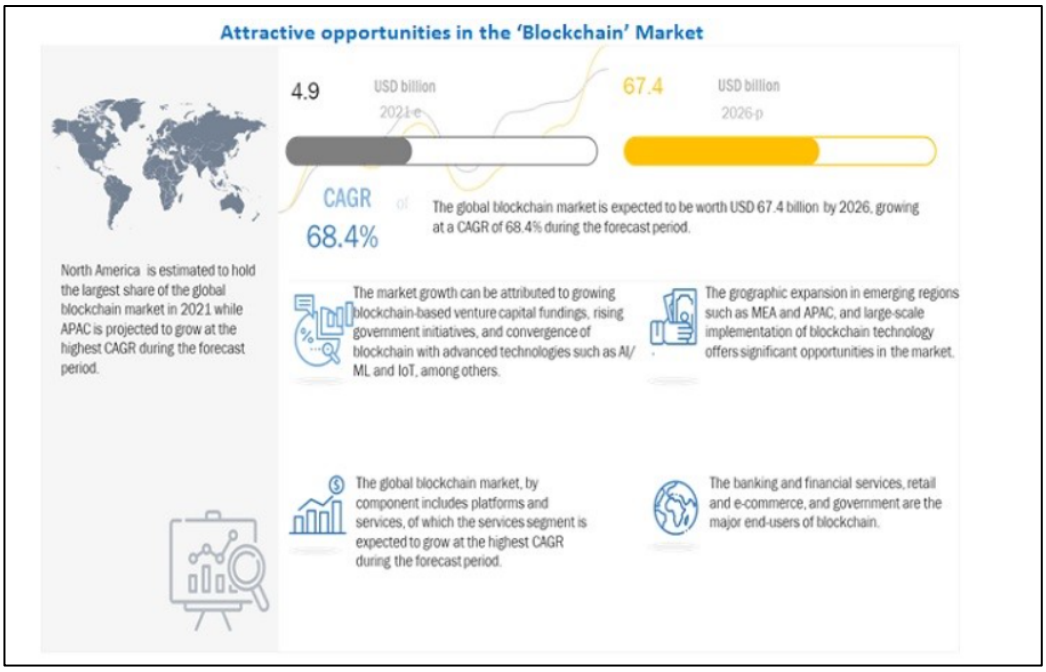


[그림 1] 시스템 전체 구성도

2. 연구 배경

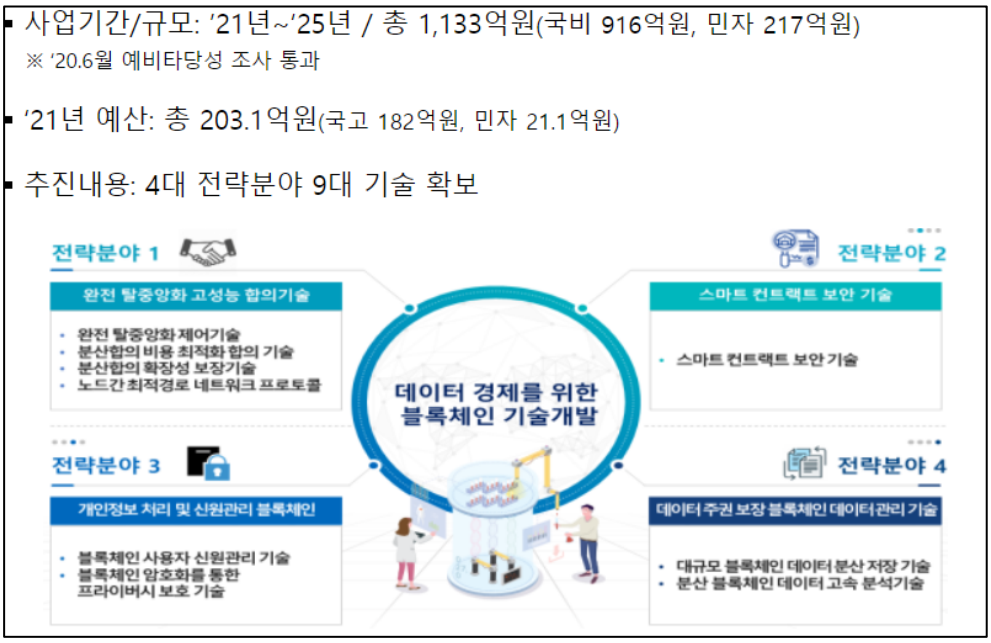
2.1. 연구 배경 및 필요성

- 4차 산업혁명의 핵심 기술 중 하나인 블록체인은 분산 원장 기술로, 중앙화된 서버나 데이터베이스를 사용하지 않아 데이터의 변조나 위조를 방지할 수 있다. 블록체인은 보안성과 불변성 등의 특성으로 인해 금융, 물류 등 다양한 산업 분야에서 주목받고 있다.



[그림 2] 블록체인 시장 규모 예측

- 시카고의 시장 조사 분석 기업인 마켓앤마켓(MarketsandMarkets)이 발표한 보고서인 '2026년까지 블록체인 시장 예측(Blobkchain Market – Global Forecast to 2026)'에 따르면, 블록체인 시장이 2026년까지 연간 68.4% 성장한다는 전망이다. 2021년 49억 달러로 예상되는 시장 규모는 2026년에 674억 달러로 크게 증가할 것으로 예상되었다.



[그림 3] 데이터 경제를 위한 블록체인 기술개발 사업 개요

- 블록체인 시장이 성장함에 따라, 과학기술정보통신부는 국내 블록체인 기술 역량을 강화하기 위한 '데이터 경제를 위한 블록체인 기술 개발 사업'의 수행기관 선정을 완료하여 기술 개발에 착수한다고 밝혔다. 그의 전략 분야 중 하나인 스마트 컨트랙트의 보안 기술은 계약 조건이 충족되면 자동으로 거래를 수행하는 디지털 계약인 스마트 컨트랙트의 보안을 의미한다. 한 번 블록체인에 배포되면 다시 수정할 수 없는 스마트 컨트랙트의 특성으로 인해, 2016년 발생한 더 다오(The DAO) 사건과 같이 스마트 컨트랙트 코드상의 보안 취약점을 통한 악의적인 공격이 발생하고 있다.

- 한 번 배포되면 수정이 불가한 특성을 고려했을 때, 스마트 컨트랙트 보안을 위해 배포되기 전에 스마트 컨트랙트 코드상의 취약점을 탐지할 수 있는 도구가 필요하다. 이를 위해 본 과제에서는 인공지능을 기반으로 Static Analysis 기법을 사용한 취약점 탐지 소프트웨어를 개발하고자 한다.

2.2. 국내외 기술 및 시장 현황

- 현재 제안된 인공지능 기반 스마트 컨트랙트 취약점 탐지 모델들은 다음과 같다.[3][4]

ContractWard	스마트 컨트랙트 소스코드를 바이트코드로 컴파일하여 전처리 과정을 거친 후, eXtreme Gradient Boosting, Adaptive Boosting, Random Forest, Support Vector Machine, k-

	Nearest Neighbor 다섯 가지 머신러닝 모델을 학습한 탐지 도구이다.
Combining Graph Neural Networks with Expert knowledge	전문 지식과 인공지능 모델을 결합한 모델로, Combining graph feature and expert patterns를 통해 취약점을 탐지한다. 특정 취약점에 대한 전문 지식을 security pattern으로 만들고, 스마트 컨트랙트로부터 해당 패턴을 추출한다. 추출 후, 스마트 컨트랙트 소스코드를 범주화된 node와 edge로 표현한 contract graph를 생성하고 normalization한다. security pattern과 normalization이 수행된 contract graph는 neural network를 통해 각각 벡터로 변환되며, 변환된 두 벡터는 결합되어 fully connected layer와 sigmoid layer를 통해 최종 취약점 탐지 결과로 계산된다. 전문 지식과 인공지능의 결합이 보다 정확한 스마트 컨트랙트 취약점 탐지를 가능하게 함을 보였다.
SMARTEMBED	기존 취약점 존재 코드와 코드 유사도를 측정하는 취약점 탐지 모델이다. 스마트 컨트랙트 소스코드를 Abstract syntax tree(AST)로 변환하고 이를 통해 소스코드를 contract-lever, function-level, statement-level의 세 가지 code snippet 형태로 재구성한 후, 취약점 탐지와 관련 없는 요소를 제거하는 normalization을 수행한다. Normalization 처리된 데이터에 대해 code embedding learning을 수행한다. 이러한 embedding 과정을 통해 모든 code snippet은 동일한 크기의 벡터 표현으로 변환된다. 벡터 변환 과정이 수행된 이후 유사도 측정을 수행하여 유사도가 사전에 정해둔 임계치보다 높을 경우, 동일한 취약점이 존재하는 것으로 판단한다.
Eth2Vec	EVM의 바이트코드와 학습한 바이트코드 간의 유사성을 비교하여 취약점을 탐지하는 방식으로 구현된 SVM(Supprot Vector Machin)기반의 탐지 모델이다. 바이트코드를 벡터화해주는 EVM Extractor와 비지도학습 기반의 PV-DM 모델로 구성되어 있다.
SoliAudit	그레이박스 기반의 퍼징 검사 기법과 머신러닝 모델을 결합한 탐지 도구. Dynamic fuzzer와 학습 모델로 구성된 Vulnerability analyzer로 각 취약점 유형에 대해 탐지를 동시에 수행하여 결과를 도출한다. Vulnerablitiy analyzer는

	Logistic Regression, SVM, KNN, CNN 등 여러 종류로 구현하였고, 각 모델에 대해 취약점 탐지 유무 실험으로 성능 비교까지 검증하였다.
ESCORT	DNN(Deep Neural Network)기반으로 한 취약점 탐지 프레임워크이다. 취약점의 유형까지 분류할 수 있고, 새로운 취약점을 추가하기 위한 전이 학습을 지원한다. 학습을 위한 데이터셋 수집 및 레이블링을 자동으로 수행하는 ContractScraper과 취약점 유형 탐지를 위한 Multi Output Layer(MOL)-DNN 모델로 구성되어 있다.

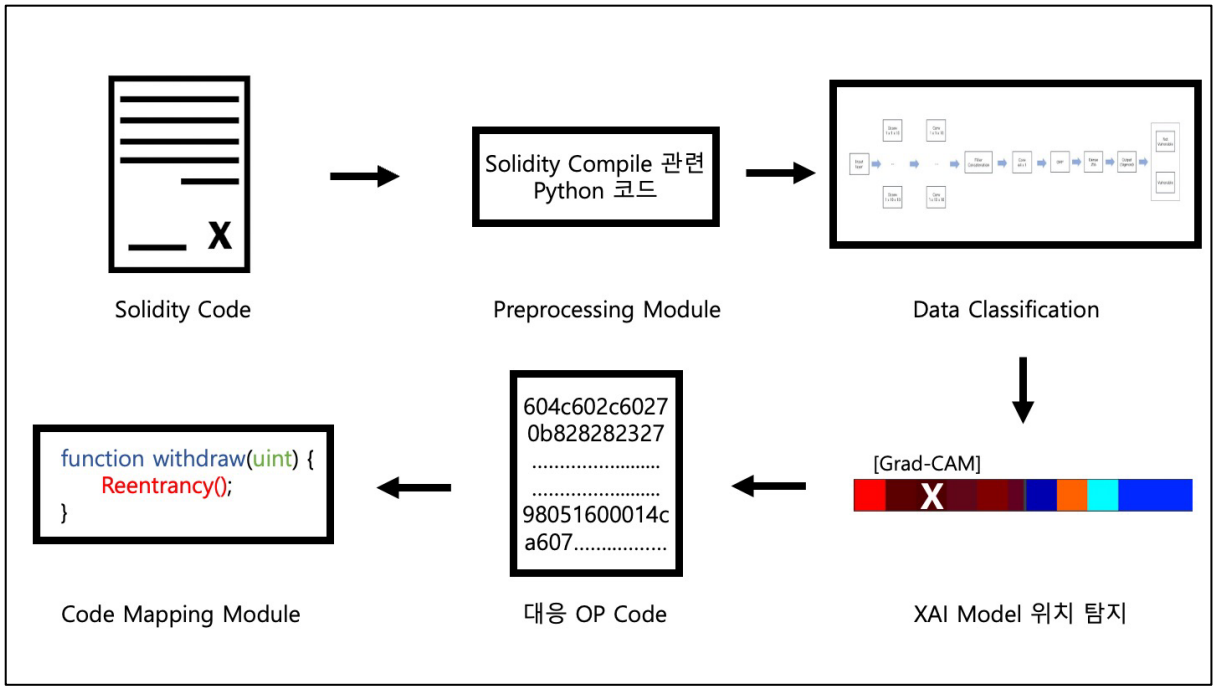
[표 1] 인공지능 기반 스마트 컨트랙트 취약점 탐지 모델

3. 연구 내용

3.1. 세부 과제 내용

- 우리가 개발하고자 하는 '(블록체인 보안) Smart Contract 취약점 탐지 소프트웨어'는 Contract의 Reentrancy(재진입) 취약점의 패턴을 분석하여 어떤 Solidity Code가 취약점을 갖는지를 탐지하고, 그 Code Block을 사용자에게 알리는 모델이다. 이를 통해, 블록체인 네트워크상에서 중개자 없이 계약이 수행되는 Smart Contract의 취약점을 이용한 악용을 방지할 뿐만 아니라 프로그래머가 코드 작성 시 취약점에 유의해 코드를 작성할 수 있도록 돕고자 한다.

3.1.1. 시스템 전체 구조



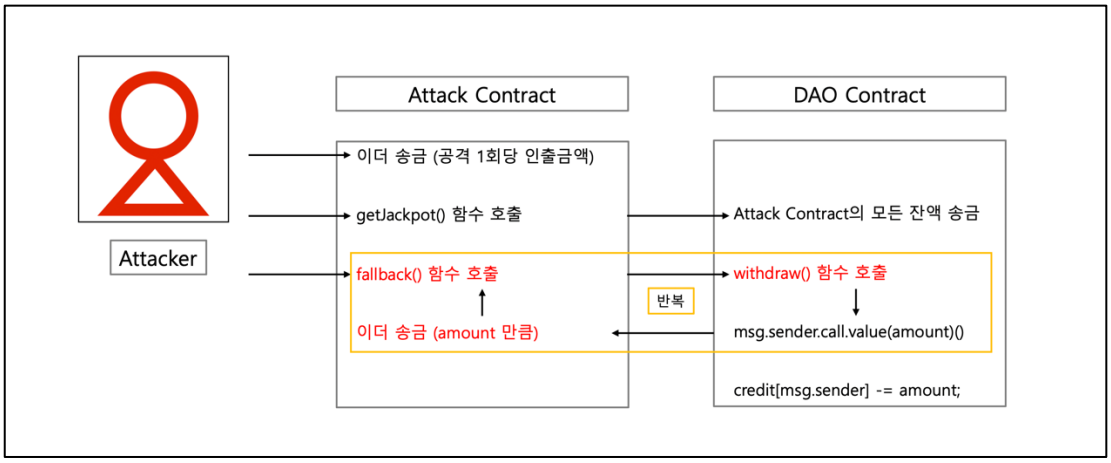
[그림 4] 시스템 전체 시나리오

- User가 자신의 Solidity Code를 UI에 입력 시, Python 을 통해 Solidity Code를 Compile해 Byte Code로 변환한다. 이 변환된 Byte Code를 CNN(이미지 처리)으로 바꾼다. XAI Model을 통해 코드의 취약점을 파악해 Code Mapping Module을 통해 취약점을 갖는 Code Block의 위치를 Mapping해 User가 알아보기 쉽게 탐지된 취약 정보를 UI로 제공한다. 그 결과엔 탐지된 취약점 종류 및 확률, 취약점 발생 위치를 탐지한 Code Block이 포함된다.

3.1.2. 스마트 컨트랙트 재진입 공격 패턴 분석

- 재진입 공격은 외부 계약 호출(external contract call)이 완료되기 전, 해당 계약 호출을 다시 요청해 계약에 재진입이 가능한 경우이다. 이는 코드가 무한정 반복 실행 되며 문제점이 발생하게 되는데, 스마트 컨트랙트 내에서 사용자의 계좌 주소가 아닌 컨트랙트 주소로 이더를 송금할 때에 호출되는 Fallback 함수로 인해 발생한다.

- 다음 그림은 재진입 공격의 시나리오이며, 공격자(Attacker)는 Attack Contract를 이용해 DAO Contract를 공격한다.



[그림 5] 재진입 공격 시나리오

- 다음은 재진입 공격이 가능한 withdraw 함수와 그를 방지하는 송금 함수의 예이다.

재진입 공격 가능	재진입 공격 불가

function withdraw(uint amount) {
  if (credit[msg.sender] >= amount) {
  bool res =
  msg.sender.call.value(amount)();
  credit[msg.sender] -= amount;
  }
}
	function withdraw(uint amount) {
	  if (credit[msg.sender] >= amount) {
	  credit[msg.sender] -= amount;
	  bool res =
	  msg.sender.call.value(amount)();
	  }
	}

[표 2] 재진입 공격 함수 예시

- 재진입 공격을 방지하기 위해, 사전조건 체크 후 credit[msg.sender] 변수의 값을 먼저 변경시키고 다른 컨트랙트와 상호작용 하는 방법으로 코드를 수정할 수 있

다. 이러한 방법을 'check-effects-interactions pattern(체크 효과 상호작용 패턴)'이라고 하며, 이로써 재진입 공격을 방지할 수 있다. 또한 그 외에도 재진입 공격을 방지하는 방법에는, transfer 함수를 사용해 가스 사용량에 제한을 두거나 mutex(뮤텍스)를 사용하는 방법 등이 있다.

3.1.3. 개발 환경 및 기술 스택

1. **Python** : 데이터 전처리, CNN, Grad-CAM에 이용한다.

PIL	Python Imaging Library의 약자로, Python 이미지 처리 라이브러리를 이용한다.
Pandas	데이터 처리와 분석을 위한 라이브러리이며, 대용량 데이터들을 처리할 때 유리하다.
Tensorflow 및 keras	딥러닝 모델 개발을 위한 프레임워크로, 신경망 모델을 구축하고 훈련시키는데 사용된다.[8]
numpy	수학적 연산과 다차원 배열 처리에 사용되며, 데이터 처리 및 수학적 계산에 필수적이다.
cv2(OpenCV)	컴퓨터 비전 작업에 사용되며, 이미지 처리를 수행하는 도구를 제공한다.
os	각 운영 체제와 상호 작용하여 파일 및 디렉토리 관리를 지원한다.
csv	csv 파일 처리를 위해 사용되며, 데이터를 읽고 쓰는 데 도움을 준다.
matplotlib	데이터 시각화 라이브러리로, 그래프 및 플롯을 생성해 데이터의 시각화를 담당한다.
glob	파일 및 디렉토리 검색을 지원하며, 일치하는 파일을 검색할 때 유용하다.
re	정규 표현식 처리를 위한 Python 모듈로, 텍스트에서 패턴을 검색하고 추출하기 위해 사용된다. Solidity 소스 코드에서 패턴을 식별하거나 추출하는 데 도움을 준다.
subprocess	Python에서 외부 프로세스를 실행하고 관리하는 모듈로, 시스템 명령어 실행 및 다른 프로그램과의 상호 작용을 지원한다. Ethereum 노드 또는 Solidity 컴파일러와 상호 작용할 때 사용될 수 있다.

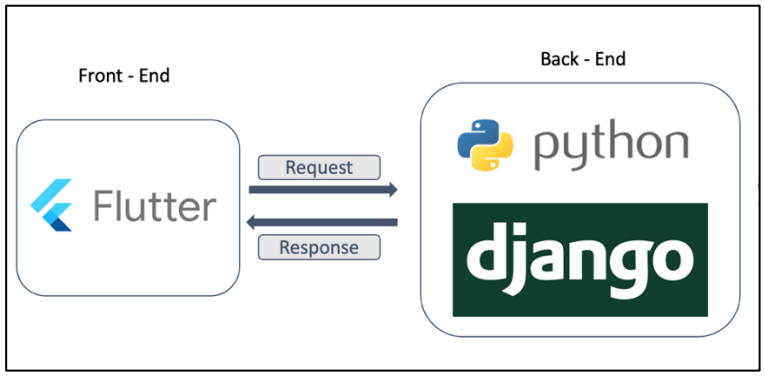
[표 3] Python 관련 개발 환경

2. **Solidity** : 취약점 탐지를 위한 Solidity 파일 분석에 이용한다. 이더리움에서 제공하는 스마트 컨트랙트 개발 언어이다. 정적 타입의 언어이며, EVM에서 구동되도록 설계되었다.

solcx (Solidity Compiler)	Solidity 언어로 작성된 스마트 계약을 컴파일하고, 이더리움 블록체인에 배포하는 데 사용되는 도구이다. 스마트 계약의 소스 코드를 컴파일하여 EVM(Ethereum Virtual Machine)에서 실행 가능한 형식으로 변환한다.

[표 4] Solidity 관련 개발 환경

3. 서비스 관련



[그림 6] 서비스 상호작용 구성도

[Front-End] Flutter	UI(사용자 인터페이스)를 구현하고, 사용자와의 상호작용을 담당한다.[6]
[Back-End] Python, Django	요청-응답 사이의 로직을 처리하고, 데이터를 관리한다. Django는 Python 기반의 웹 어플리케이션 프레임워크이며, 웹 개발을 쉽게 하도록 도와준다.[7]

[표 5] 서비스 관련 개발 환경

4. 협업 및 프로젝트 관리

Github	개발 협력 및 프로젝트 관리
Pycharm	개발 IDE
Notion	회의록 및 공통 문서 작업

[표 6] 협업 및 프로젝트 관리

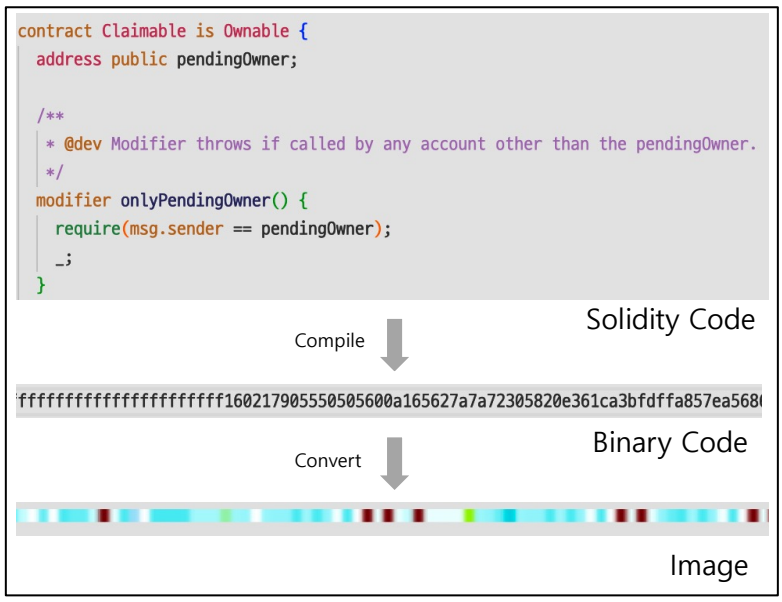
3.2. 데이터 수집

3.2.1. 사용 데이터 셋

- 우리 졸업과제 팀은 기존의 보안 취약점 탐지 도구인 MythX, Security, Oyente, SmartCheck를 사용하여 스마트 컨트랙트의 보안 취약점을 레이블링하는 전통적인 방식을 넘어서 새로운 모델을 개발하기로 결정하였다. 또한 이러한 프로그램들은 데이터 레이블링을 위해 유료 버전(솔리디티 파일 1개를 레이블링하는 데에 평균 $0.25 소모)을 구입해야 하거나, 솔리디티 파일 하나당 약 5분의 시간이 소요되어 시간과 비용이 많이 드는 작업이었다. 따라서, 우리 팀은 CodeNet 논문[2]에 사용된 데이터 셋을 활용하기로 결정하였다. 해당 논문에서는 취약점 삽입 도구인 SolidFi를 사용하여 재진입 공격 취약점을 생성한 데이터를 활용하여 연구를 수행하였으며, Github의 Smartbugs에서 제공하는 취약점이 라벨링 된 curated 데이터 셋[1]을 포함해 총 24,367개의 데이터 셋을 확보하였다.

3.3. 데이터 전처리

- 데이터 전처리 단계에서는 주어진 Solidity 코드의 컴파일을 시행한다. 컴파일이 완료되면 각 컨트랙트가 바이너리 파일(.bin)로 저장된다. 이 바이너리 파일은 Encoding 과정을 거쳐 이미지 형태로 저장된다. 전체 전처리 과정을 그림으로 표현하면 다음과 같다.



[그림 7] 전처리 과정

3.3.1. Solidity Code Compile

- Solidity 코드 컴파일 단계에서는 확보한 데이터 셋의 Solidity 파일들을 컴파일하여 바이트 코드 파일들을 생성하였다. 이 과정은 compileSol.py의 python 파일을 생성하여 수행하였다.

- **compileSol.py**

```
# compileSol.py
import subprocess
from solcx import compile_source
import re

# 소스코드 컴파일
def compile_source_file(file_path):
    with open(file_path, 'r', encoding='UTF8') as f:
        source = f.read()
    return compile_source(source)

# 솔리디티 컴파일러 버전 추출
def extract_compiler_version(sol_file_path):
    with open(sol_file_path, 'r', encoding='UTF8') as file:
        solidity_code = file.read()
        # 정규식 패턴으로 pragma 문 또는 컴파일러 지시어를 찾음.
        pragma_pattern =
re.compile(r'pragma₩s+solidity₩s+["₩']?([^₩s"₩'₩;]+)["₩']?₩s*;',
re.IGNORECASE)
        matches = pragma_pattern.findall(solidity_code)
        if matches:
            # pragma 문 또는 컴파일러 지시어에서 버전 정보 추출
            compiler_version = matches[0]
            return compiler_version
        else:
            return None

# 솔리디티 컴파일러 해당 버전 설치
```

```
def install_version(version):
    cmd_command = f"solc-select install {version}"
    install_process = subprocess.Popen(cmd_command, stdout=subprocess.PIPE,
shell=True)
    output, error = install_process.communicate()
    print(output.decode())

# 솔리디티 컴파일러 버전 선택
def select_version(version):
    cmd_command = f"solc-select use {version}"
    select_process = subprocess.Popen(cmd_command, stdout=subprocess.PIPE,
shell=True)
    output, error = select_process.communicate()
    print(output.decode())

# 솔리디티 컴파일러 버전 확인
def check_version():
    check_cmd = "solc --version"
    check_process = subprocess.Popen(check_cmd, stdout=subprocess.PIPE,
shell=True)
    output, error = check_process.communicate()
    print(output.decode())
```

- **"compileSol.py"**에서는 Solidity 파일의 경로를 입력받아, 컴파일러의 버전을 관리하고 컴파일하는 모듈로 다음과 같은 함수들을 구현했다.

1. **compile_source_file(file_path)** : Solidity 소스 코드가 포함된 파일을 읽어와 컴파일한다. Solcx 라이브러리를 사용하여 소스 코드를 컴파일하고, 컴파일 된 결과를 반환한다.
2. **extract_compiler_version(sol_file_path)** : Solidity 파일에서 컴파일러 버전 정보를 추출한다. 정규식 패턴을 사용하여 소스 코드 안에서 pragma solidity 문이나 컴파일러 지시어를 찾아 솔리디티 컴파일러의 버전을 추출하여 반환한다.
3. **install_version(version)** : 특정 버전의 솔리디티 컴파일러를 설치한다. solc-select install {version} 명령어를 사용하여 지정된 버전의 솔리디티 컴파일러를 설치한다.
4. **select_version(version)** : 설치된 솔리디티 컴파일러 버전을 선택한다. solc-select

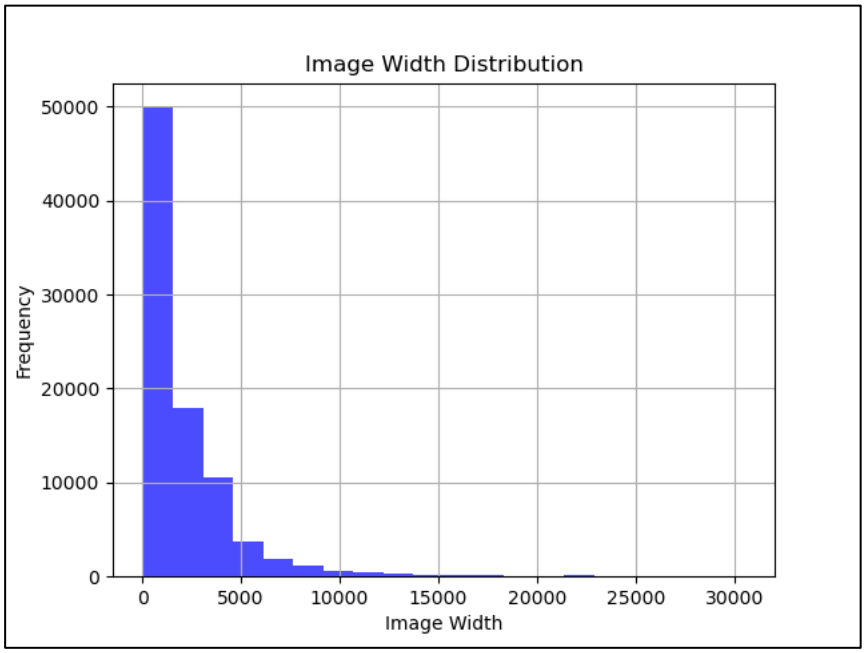
use {version} 명령어를 사용하여 지정된 버전의 솔리디티 컴파일러를 선택한다.

5. **check_version()** : 현재 선택된 솔리디티 컴파일러 버전을 확인한다. solc --version 명령어를 사용하여 현재 선택된 솔리디티 컴파일러 버전을 콘솔에 출력한다.

3.3.2. Byte Code 이미지 변환

- 데이터 전처리 두 번째 단계에서는 컴파일된 바이트코드 파일을 통해 이미지를 생성하였다. 또한 바이너리 파일의 인코딩 시 OP 코드의 크기를 모두 균일하게 맞춰주는 작업을 수행하였다.

- OP 코드의 크기 조정 과정에서 이미지 크기를 살펴보면, 대부분의 이미지의 길이가 10,000 이하임을 아래 표에서 확인할 수 있다. 또한 제로 패딩이 늘어날수록 학습 효율이 저하될 수 있으므로, 정보 손실을 최소화하면서 학습 효율을 높이기 위해 가로 크기를 10,000으로 조정하여 OP 코드의 크기를 균일하게 조정하였다.



[그림 8] 이미지 크기 분포표

- **createImage.py**

```
# createImage.py
from PIL import Image

push = {'60': 1, '61': 2, '62': 3, '63': 4, '64': 5, '65': 6, '66': 7, '67': 8, '68': 9, '69': 10,
        '6a': 11, '6b': 12, '6c': 13, '6d': 14, '6e': 15, '6f': 16, '70': 17, '71': 18, '72':
```

```
19, '73': 20,
            '74': 21, '75': 22, '76': 23, '77': 24, '78': 25, '79': 26, '7a': 27, '7b': 28, '7c':
29, '7d': 30,
            '7e': 31, '7f': 32}


# 바이트 단위로 나눠 RGB값 지정
def create_pixels(bytecode):
    pixel_colors = []
    i = 0
    push_keys = push.keys()
    while i < len(bytecode):
        byte = bytecode[i:i + 2]
        i += 2
        r = int(byte, 16)
        g = 0
        b = 0
        if byte in push_keys:
            g = int(bytecode[i:i + 2], 16)
            i += 2
            n = push[byte]
            if n != 1:
                b = int(bytecode[i:i + 2], 16)
                i += 2 * (n - 1)
        if len(pixel_colors) < 10000:
            pixel_colors.append((r, g, b))
        else:
            break

    while len(pixel_colors) < 10000:
        pixel_colors.append((0, 0, 0)).          # Zero Padding

    if i < len(bytecode):
        pixel_colors = []
    return pixel_colors
```

```
def create_image(file_name, pixel_colors):
    width = 10000
    if width != 0:
        image = Image.new('RGB', (width, 1))  # 이미지 생성
        pixels = pixel_colors     # 픽셀 색상 설정
        image.putdata(pixels)     # 이미지에 픽셀 색상 적용
        image.save(file_name + '.png')
```

- **"createImage.py"**는 바이트 코드를 기반으로 이미지를 생성하는 모듈로, 바이트 코드를 RGB 픽셀 값으로 변환하여 이미지를 생성하고, 생성된 이미지를 저장하는 기능을 제공한다. PUSH 명령어는 OP코드 중 유일한 오퍼랜드를 갖기 때문에 PUSH 명령어에 따라 컨볼루션 필터의 크기를 고정할 수 있다. PUSH 명령어는 최대 32개까지 오퍼랜드를 인자에 받을 수 있으며, 관련 연구에 따르면 중복 없는 3000여 개의 컨트랙트에서 PUSH 명령어의 통계를 분석한 결과 PUSH1 ~ PUSH2까지의 비율이 전체의 78.8%를 차지하여 PUSH2까지의 정보를 남긴다면, PUSH 명령어에서 정보 손실을 최소화하면서 OP코드의 사이즈를 균일화 할 수 있다. 이를 고려하여, PUSH2를 최대 크기로 정하고 PUSH2보다 큰 PUSH 명령어의 오퍼랜드는 제거하며, PUSH2보다 작은 PUSH1과 다른 명령어들의 남은 부분은 제로 패딩으로 채워주는 방식으로 Encoding하여 각 코드를 RGB에 순서대로 대응하여 픽셀 값으로 변환하고 이미지를 생성한다. createImage.py에서는 다음과 같은 함수들을 사용한다.

1. **push 딕셔너리** : PUSH2보다 큰 PUSH 명령어의 오퍼랜드들을 제거하고 명령어에 따라 RGB값을 매핑하기 위해 push 딕셔너리를 정의하였다.
2. **create_pixel(bytecode) 함수** : 입력으로 받은 바이트 코드를 바이트 단위로 나눠서 RGB 픽셀 색상값을 생성한다. 주어진 바이트 코드를 바이트 단위로 순회하면서 각 바이트를 기반으로 R, G, B 값을 계산하여 리스트에 저장한다. 오퍼랜드를 갖지 않는 OP코드들은 G, B값을 0으로 저장하고, PUSH 명령어들은 push 딕셔너리를 사용하여 특정 바이트 코드에 해당하는 G, B 값을 계산한다.
3. **create_image(file_name, pixel_coloers) 함수** : 생성된 픽셀 색상 값을 이용하여 이미지를 생성하고 지정된 파일 이름으로 이미지를 저장한다. 픽셀 색상값 리스트를 이용하여 OP 코드는 좌우의 순서 외에 위아래 코드와의 관계에서 큰 정보

를 얻기 어려우므로, 이미지의 세로 크기는 1로 고정하고, 가로 크기가 픽셀 수(10,000)와 같은 이미지를 생성하였다. 픽셀 색상값을 이미지에 적용한 뒤 해당 이미지를 지정된 파일 경로에 PNG 형식으로 저장한다. CNN 모델의 입력으로 사용하기 위해 가로 크기를 고정된 특정 값(10,000)으로 지정한다.

3.3.3. 최종 전처리 및 컨트랙트 분류 과정

- **preprocessing.py**를 통해 지정된 폴더 경로에서 모든 솔리디티 파일들의 리스트를 가져온다. 앞서 생성한 compileSol.py와 createImage.py를 이용해 파일들을 이미지로 생성하여 저장한다.

- **preprocessing.py**

```
# preprocessing.py
import glob
import createImage
import compileSol


def preprocessing(folder_path):
    versions = set()  # 설치된 버전
    files = glob.glob(folder_path + '/*.sol')
    file_num = 1
    version = '0.4.26'
    compileSol.install_version(version)
    compileSol.select_version(version)
    for file in files:
        print(file)
        version_in_use = version
        version = compileSol.extract_compiler_version(file)
        if version is None:
            continue
        if version[0] == '^':
            version = '0.4.26'

        if version != version_in_use:
```

```
        if version not in versions:
            compileSol.install_version(version)
            versions.add(version)

        compileSol.select_version(version)

    try:
        compiled_sol = compileSol.compile_source_file(file)
    except Exception as e:
        with open('./exception/compile/' + str(file_num), 'w') as f:
            s = version + '₩n' + file + '₩n' + str(e)
            f.write(s)
        continue

    keys = compiled_sol.keys()
    for key in keys:
        label = 0
        strs = key.split(':')
        bin_file_path = folder_path + '/bin/' + str(file_num) + strs[1] + '.bin'

        for function in compiled_sol[key]['abi']:
            if 'name' in function.keys():
                # 컨트랙트 내에 'reenbug'로 시작하는 이름의 함수 존재
                if function['name'].startswith('reenbug'):
                    label = 1
                    break

        bytecode = compiled_sol[key]['bin']
        with open(bin_file_path, 'w') as f:
            f.write(bytecode)

        try:
            pixel_colors = createImage.create_pixels(bytecode)
        except Exception as e:
            with open('./exception/img/' + str(file_num), 'w') as f:
```

```
                    f.write(version + '₩n' + file + '₩n' + str(e))
                continue

            if len(pixel_colors) != 10000:
                continue

            if label == 0:
                file_name = './images/no/' + str(file_num) + strs[1]
            else:
                file_name = './images/yes/' + str(file_num) + strs[1]

            createImage.create_image(file_name, pixel_colors)
        file_num = file_num + 1


folder_path = 'E:/sol'        # 솔리디티 파일 폴더 위치
preprocessing(folder_path)
```

- **"preprocessing.py"** 에서는 Solidity 컴파일러의 버전 정보를 관리하기 위해, version 변수에 현재 사용 중인 Solidity 컴파일러 버전을 저장한다. 기본값으로 0.4.26 버전을 사용하기 위해 해당 버전을 설치하고 선택한다. 각 솔리디티 파일들을 처리하기 위해 for loop를 사용했다. 이 안에서 다음과 같은 작업들을 수행하였다.

1. 솔리디티 파일의 컴파일러 버전 확인 :
   A. 솔리디티 파일 내에 컴파일러 버전 정보를 추출하여 확인한다.
   B. 버전 정보가 '^'로 시작하는 경우, 해당 버전 이상에서 최신 버전까지 사용 가능하다는 것을 의미한다. 따라서, 추출한 버전 정보가 '^'로 시작하는 경우 0.4x 버전 중 가장 최신 버전인 0.4.26을 사용한다.
2. 컴파일러 버전 설정 :
   A. 버전 정보가 변경되었을 경우, 해당 버전을 설치하고 선택한다.
3. 소스 코드 컴파일 및 바이트 코드 저장 :

A. 솔리디티 파일을 컴파일하여 컨트랙트별 바이트 코드를 추출하였다.

B. 추출한 바이트 코드를 파일로 저장하였다.

4. 바이트 코드를 이용한 이미지 생성 및 저장 :

A. 추출한 바이트 코드를 RGB 픽셀 값으로 변환하여 이미지를 생성한다.

B. 생성된 이미지를 지정된 파일 경로에 PNG 형식으로 저장한다.

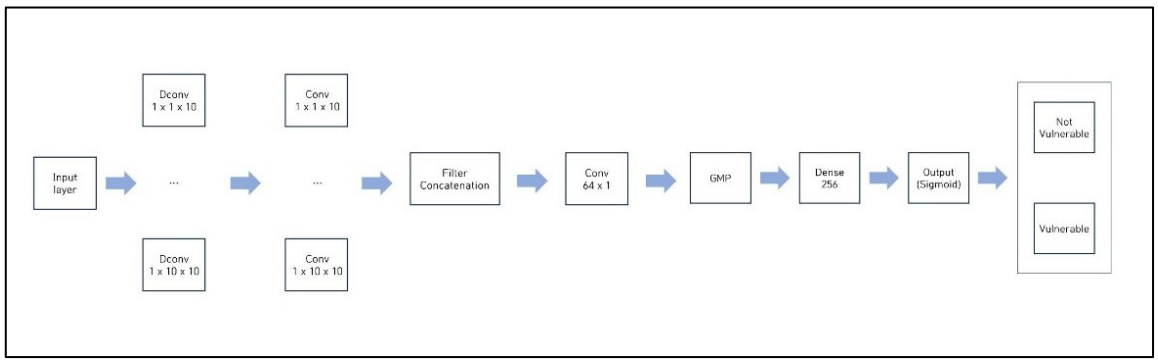
- 위와 같은 과정을 통해 소스 코드들을 컴파일하고, 바이트 코드를 추출해 이미지로 변환하여 저장하는 기능을 수행하였다. 또한, 예외 처리를 통해 오류가 발생하여도 프로그램이 멈추지 않도록 설계하였다.

3.4. 모델 학습

3.4.1. CNN 모델 설계

- **CNN**(Convolutional Neural Network)는 이미지 처리에서 널리 사용되는 신경만 구조로, Convolution과 Pooling 레이어를 반복하여 사용하며, 이미지의 특징을 추출하고 분류하는 데 효과적이다. Convolution Layer에서는 이미지의 특징을 추출하고, Pooling Layer에서는 이미지의 크기를 줄이고 특정한 특성을 강조한다. 마지막으로 Fully Connected Layer에서는 추출한 특징이 무엇을 의미하는 데이터인지 분류하는 작업을 한다.

- CNN Model은 이진 분류를 해야 하며, 해당 Image 파일이 취약점을 갖고 있는가(yes)/아닌가(no)의 두 가지로 분류한다. 모델의 주요 구조와 학습 및 평가 단계는 아래와 같다.



[그림 9] CNN 모델 구조

1. **입력 레이어** : Input 레이어를 사용하여 모델의 입력 형태를 정의하였다. 입력 형태는 (1, 10000, 3)으로 설정되어 있으며, 이는 높이가 1, 폭이 10,000, 채널 수(RGB)가 3인 이미지이다.

2. **Depthwise Separable Convolution 블록** : depthwise_separable_conv_block을 정의하여 구현하였다. 이 블록은 두 개의 합성곱 레이어로 구성된다.

   A. Depthwise Convolution 레이어 : 커널 크기는 입력 받은 크기를 사용하며, 활성화 함수는 'relu'이다.

   B. Pointwise Convolution 레이어 : 10개의 필터와 커널 크기 (1, 1)을 사용하며, 활성화 함수는 'relu'이다.

3. **Convolution 레이어** : Depthwise Convolution 블록에 커널 크기를 (1, 1)부터 (1,10)까지 적용하여 나온 결과를 연결(concatenate)하여 하나의 텐서로 합친다. 이어 Con2D 레이어를 사용하여 64개의 필터와 커널 크기 (1, 3)을 적용한다. 활성화 함수는 'relu'이다.

4. **GlobalMaxPooling2D 레이어** : Global Max Pooling을 사용하여 2D 텐서를 1D 텐서로 변환한다.

5. **Fully Connected 및 Output 레이어** : Fully Connected 레이어에서 256개의 뉴런을 갖는 레이어를 사용하며 활성화 함수는 'relu'이다. 출력 레이어에서 이진 분류를 수행하기 위해 1개의 뉴런을 갖는 레이어를 사용하고 활성화 함수는 'sigmoid'이다.

6. **모델 컴파일** : 모델은 'adam' 옵티마이저를 사용하여 컴파일되며, 손실 함수는 'binary_crossentropy'로 설정된다. 평가 메트릭은 'accuracy'로 설정된다.

7. **데이터 로딩 및 전처리** : ImageDataGenerator를 사용하여 이미지 데이터를 로드하고 전처리한다.

8. **학습 데이터 로딩** : flow_from_directory 메서드를 사용하여 이미지 데이터를 배치로 로드하고 이진 분류를 위한 데이터를 생성한다.

9. **학습 및 검증 데이터 분할** : 학습 데이터는 70%로 설정되고, 나머지 30%의 데이터는 검증 및 테스트를 위해 사용된다.

10. **모델 학습** : 모델은 학습 데이터를 사용하여 1,000번의 에포크 동안 학습된다. EarlyStopping 콜백은 검증 손실이 3 에포크동안 개선되지 않으면 학습을 조기 중지하고, 최상의 가중치를 복원한다.

11. **모델 평가** : 학습이 완료된 후, 모델은 테스트 데이터를 사용하여 평가되며, 손실 및 정확도가 출력된다. 이후 모델의 성능을 더 자세히 살피기 위해 테스트 데이터를 사용해 classification_report와 confusion_matrix를 출력해준다.

3.5. Code Mapping

3.5.1. XAI를 통한 취약점 위치 탐지

- 이미지에서 취약점이 발생하면 Grad-CAM (Gradient-weighted Class Activation Mapping)을 통해 이미지 Localization을 수행한다. Grad-CAM은 XAI로서 인공지능 모델의 동작 원리를 해석하며, 구체적으로 모델이 예측을 수행할 때 어느 부분을 "보고" 있는지 지역화하여 이해하는 데 도움을 준다. 이 결과를 바탕으로, 몇 번째 OP 코드가 취약점을 갖는지 확인한다.

- 주어진 입력 이미지에 대해 Grad-CAM을 계산하는 함수인 'grad_cam' 함수는 입력 이미지를 사용하여 특정 클래스에 대한 관심 영역을 강조하고 결과 이미지 및 heatmap 값을 저장한다. 이는 다음과 같은 형태로 이루어져 있다.

1. **사용되는 매개변수** : input_model(사전 훈련된 신경망 모델), image(Grad-CAM을 생성하려는 입력 이미지), class_index(활성화를 시각화하려는 인덱스), layer_name(활성화를 추출하려는 모델의 레이어 이름)

2. **grad_cam 함수 수행 과정**

   A. 입력 이미지에 대한 클래스 예측을 수행하고, 예측된 클래스를 계산한다.

   B. 모델에서 지정한 레이어의 출력 텐서를 추출한다.

   C. 마지막 합성곱 레이어에서 대상 클래스 출력에 대한 그래디언트(gradient)를 계산한다. 이어 그래디언트(gradient)의 전역 평균을 계산한다.

   D. 예측된 클래스에 대한 이미지의 다른 부분의 중요성을 나타내는 heatmap을 생성한다.

E. Heatmap을 정규화하고, 입력 이미지의 크기에 맞게 조정한다. 또한 이를 RGB 형식으로 변환하고 원본 이미지 위에 겹쳐 표시한다.

F. 반환값은 예측값과 예측에 가장 많은 영향을 끼친 위치의 인덱스들을 반환한다.

3.5.2. 이미지 위치와 Solidity Code 매핑

- Code Mapping Module은 CNN과 Grad-CAM을 통해 얻은 취약점의 바이트 코드 위치를 원래의 솔리디티 파일로부터 문제가 되는 함수의 소스 코드와 매핑하는 역할을 수행한다. 이미지 위치를 솔리디티 코드로 매핑하는 과정은 아래와 같다.
    - 먼저, 솔리디티 컴파일러의 어셈블리(asm) 옵션을 활용하여 어셈블리 정보를 추출한다.
    - 반환된 어셈블리 정보의 인덱스를 바이트 코드의 Op Code 인덱스와 일치시킨다.
    - 이어서, Grad-CAM을 통해 얻은 바이트 코드의 OP Code 인덱스를 사용하여 어셈블리 정보에서 {'begin': 217, 'end': 257, 'name': 'MSTORE'}와 같은 정보를 찾아낸다. 이는 해당 OP Code가 매핑되는 소스 코드 부분의 시작 인덱스와 끝 인덱스를 의미한다.
    - 마지막으로, 시작 인덱스와 끝 인덱스를 활용하여 해당 부분이 포함된 함수를 찾아내고 반환한다.
- 위 절차에 따라 이미지 위치로부터 소스 코드 내의 함수까지의 매핑 과정을 구현해 아래와 같은 결과를 얻었다.

```
--------------------------------------------------
opcode idx: 646   opcode 위치
begin: 3340       소스코드 시작 위치
end: 3361         소스코드 끝 위치
--------------------------------------------------
Begin line: 118
End line: 118
source:
 transferValue_re_ent4)  매핑된 소스코드
--------------------------------------------------
Function source:
function reenbug_3() re_ent3_onlyOwner public {          해당 소스코드가 포함된 함수
        // ensure there is a reward to give
        require(redeemableEther_re_ent3[msg.sender] > 0);
        uint transferValue_re_ent4 = redeemableEther_re_ent3[msg.sender];
        bool success = msg.sender.call.value(transferValue_re_ent4)("");   //bug
        require(success);
        redeemableEther_re_ent3[msg.sender] = 10;
        redeemableEther_re_ent3[msg.sender] = 0;
    }
```

[그림 10] 솔리디티 코드 매핑 결과

3.6. 시각화 UI 설계

- 시각화 UI의 전체 시스템 구성 요소는 다음과 같다.

1. **Frontend(프론트엔드)** : Flutter
   A. UI(사용자 인터페이스)를 구현하고, 사용자와의 상호작용을 담당한다.
2. **Backend(백엔드)** : Python, Django
   A. 요청 – 응답 사이의 로직을 처리하고, 데이터를 관리한다.
   B. Django는 Python 기반의 웹 어플리케이션 프레임워크이다.

- 다음으로 사용자가 서비스를 이용하는 방식은 다음과 같다.

1. 사용자는 웹을 통해 시스템 프론트엔드에 접속한다.
2. 프론트엔드에서 사용자로부터 Solidity Code의 파일 입력을 받고, 사용자의 요청을 받아 백엔드로 전달한다.
3. 백엔드에서는 받은 요청을 처리하여 다음과 같은 로직을 수행한다.
   A. 해당 Solidity Code를 전처리(preprocessing.py)를 통해, 각 파일을 바이트 코드(.bin)으로 변환한 후 그에 대응하는 이미지(.png)를 생성한다.

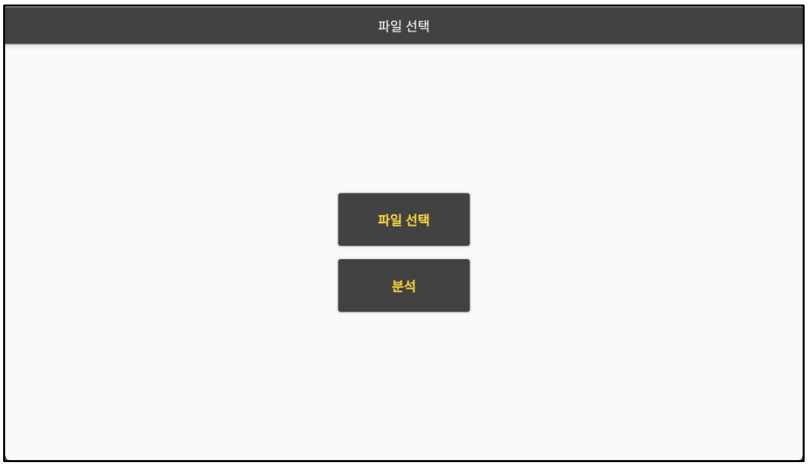
B. 생성된 이미지를 CNN 모델에 넣어 결과값을 받는다.

i. 만약 이미지에 취약점이 존재하지 않는다면, 취약점이 있을 확률 등 그에 대응하는 Response를 생성한다.

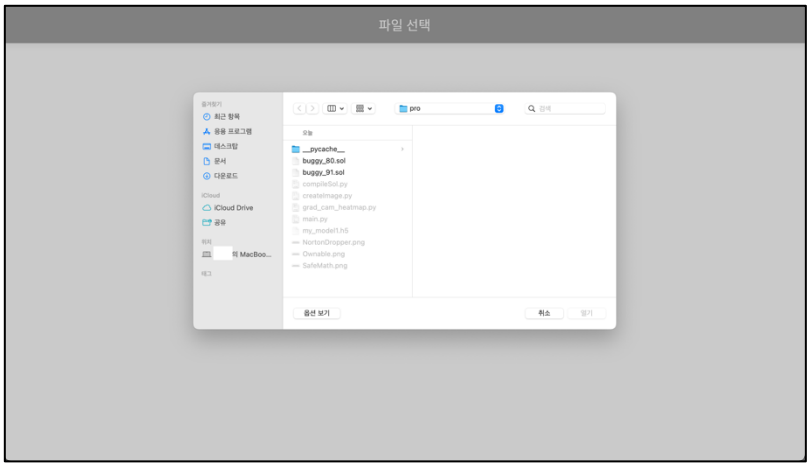
ii. 만약 이미지에 취약점이 존재한다면, 해당하는 결과값을 XAI와 Code Mapping Module을 통해 기존 솔리디티 코드의 어디에서 취약점이 발생했고, 재진입 공격 취약성 확률을 리턴해 Response를 생성한다.

3.6.1. 설계된 화면 및 솔리디티 코드 시연

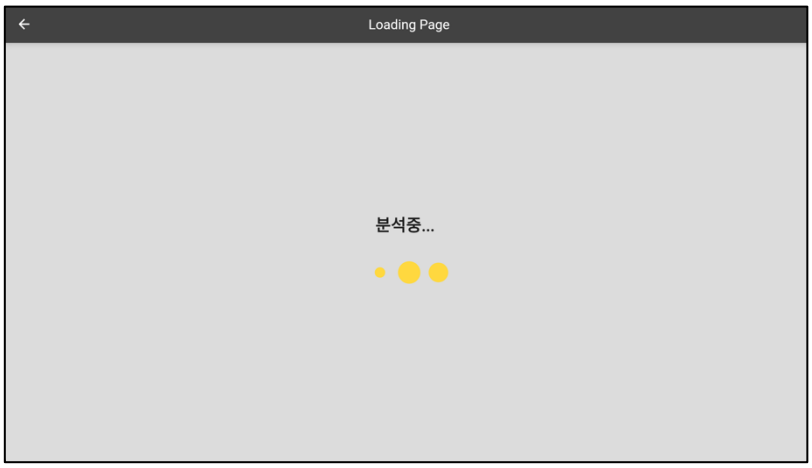
- 다음은 구현한 프론트엔드에서 예시 솔리디티 코드의 작동 예시이다.
  - 파일 선택 화면



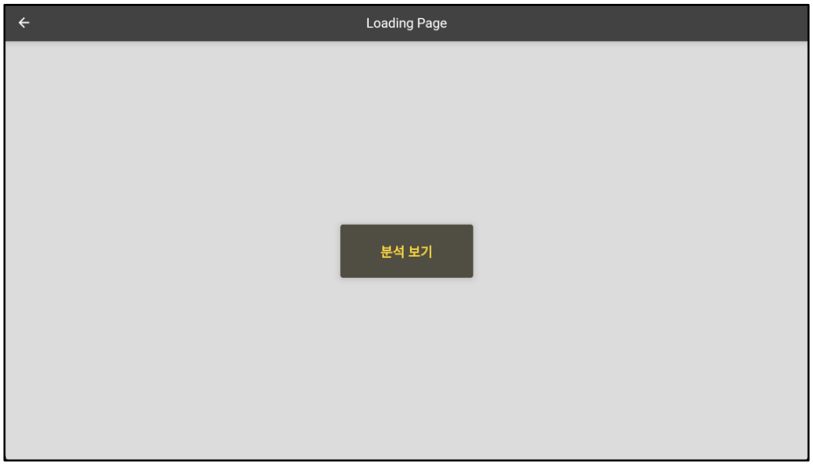
  - 파일 불러오기 화면



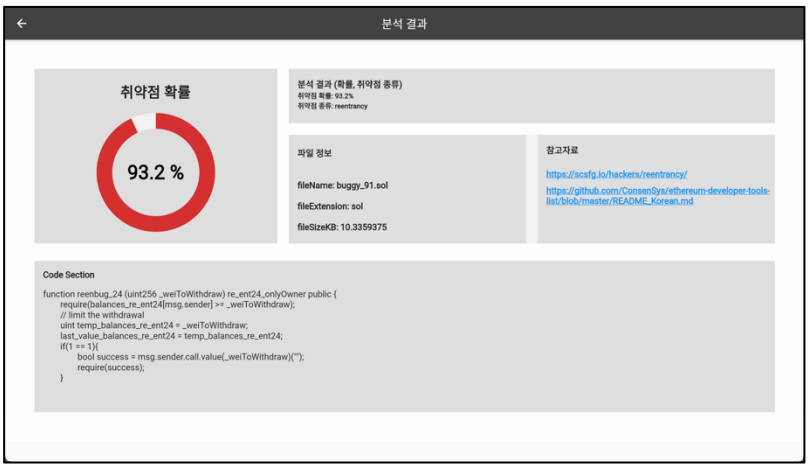
- 분석중 화면



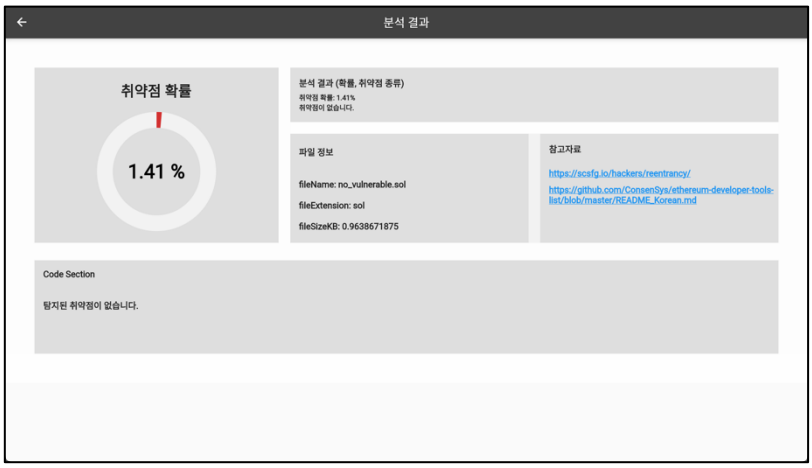
- 분석 완료 시 화면



- 결과 화면 1 : 취약점이 존재한다면 취약점이 있을 확률, 파일 메타 정보, 취약점 종류, 참고 문서, 취약점이 존재하는 함수 내용이 출력된다.



- 결과 화면 2 : 만약 취약점이 없을 경우, 다음과 같은 화면이 출력된다.



4. 연구 결과 분석 및 평가

4.1. 결과 분석

4.1.1. CNN 모델 평가

- 모델 정확도 : 모델을 테스트 셋을 활용해 평가한 결과, classification report는 다음과 같으며 모델 정확도(accuracy)가 0.93으로 확인되었다. 이는 전체 데이터 중에서 올바르게 분류된 비율이 93%임을 나타낸다.

	precision	recall	f1-score	support
no (취약점 X)	0.91	1.00	0.95	7633
yes (취약점 O)	0.99	0.75	0.86	3149

[표 7] CNN Model Classification Report

- 위 [표 7]의 분류 리포트를 해석해보면 다음과 같다. :

  - **Precision (정밀도)** : 모델이 취약점이 있다고 예측한 샘플 중 실제로 취약점이 있는 샘플의 비율은 0.99로 매우 높다.

  - **Recall (재현율)** : 실제로 취약점이 있는 경우, 취약점이 있다고 모델이 올바르게 예측한 재현율은 0.75로 취약점이 있는 파일 중 75%만을 올바르게 예측하였다.

  - **F1-score (F1 점수)** : 정밀도와 재현율의 조화 평균으로, 모델의 성능을 종합적으로 평가한 결과 취약점이 있는 경우의 f1 점수는 0.86으로 모델이 86% 정도의 균형을 이루고 있는 것을 확인할 수 있다.

  - **Support (지지도)** : 각 클래스(yes, no)의 실제 데이터 포인트의 수로, 취약점이 없는 데이터가 7633개, 취약점이 있는 데이터가 3149개이다.

- 재현율(Recall)이 취약점이 있는 경우에 0.75로 상대적으로 낮은 것을 확인할 수 있다. 이는 모델 학습 시 취약점이 없는 데이터의 개수보다 취약점이 있는 데이터의 개수가 현저히 적어, 클래스 감지가 상대적으로 더 어려운 경향이 있는 것으로 판단된다. 취약점이 있는 클래스의 재현율을 높이기 위해서는 더 많은 취약점 데이터를 수집하거나, 취약점이 없는 클래스

를 언더샘플링하여 사용하거나 가중치를 조정하는 등의 향후 연구가 필요하다.

4.2. 이미지 Localization 오류 분석

- 스마트 컨트랙트 취약점의 위치를 식별하는 과정에서, Grad-CAM을 활용한 CNN 모델이 정확하지 않은 결과를 반환하는 상황이 관찰되었다. 이러한 현상에 대한 분석은 다음과 같다.
    - CNN의 설계 특성 : CNN 모델은 원래 이미지 분류 문제에 초점을 맞추어 설계되었다. 이로 인해 각 클래스에 대응하는 특징이 활성화되는 패턴을 학습하도록 구성되어 있지만, 그 특징들이 이미지 내에서 정확히 어디에서 발생하는지 까지는 명시적으로 학습하지 않는다.
    - 특징 활성화 패턴의 불명확성 : CNN 내부에서 생성되는 특징 활성화 패턴은 종종 불분명하게 나타난다. 이로 인해 동일한 클래스의 다른 인스턴스들 사이에서 일관된 위치 정보를 제공하지 못할 수 있음을 시사한다.
    - Grad-CAM 기법의 한계 : Grad-CAM 기법은 결정 요인이 되는 영역을 고수준에서 시각화하는데 집중한다. 그러나 이 방법론은 세부적인 위치 정보를 제공하기 어렵다는 한계가 있다.
- 따라서 위와 같은 요소들로 인해 스마트 컨트랙트 취약점의 정확한 위치 탐색에 어려움이 있음을 확인하였으며, 이 문제를 해결하기 위해서 추가적인 연구가 필요함을 결론지었다.

5. 결론 및 향후 연구 방향

- 본 연구에서는 스마트 컨트랙트 재진입 공격 취약점을 탐지하기 위해 CNN 기반의 모델을 활용하는 도구를 개발하였다. 이미지 지역화 기술을 활용해 취약점을 소스 코드로부터 효과적으로 식별할 수 있었으며, 딥러닝을 사용하여 신속한 탐지를 가능하게 하였다. 이러한 연구 결과를 토대로, 다음과 같은 향후 연구 방향을 제시하고자 한다.

- 향후 연구 방향

    - 더 넓은 범위의 스마트 컨트랙트 보안 취약점 탐지 : 현재의 연구는 재진입 공격에 중점을 두었지만, 스마트 컨트랙트의 다른 보안 취약점에 대한 탐지 기능을 확장하는 연구가 필요하다. 더 많은 데이터셋을 확보하여 재진입(Reentrancy) 공격 외에도 접근 제어(Access Control), 주소(Tx.origin), 시간 조작(Time Manipulation)의 취약점을 식별하는 향후 연구를 통해 스마트 컨트랙트에서의 보안을 강화하고자 한다.

    - 지속적인 데이터 학습 및 라벨링 작업의 개선 : 공개된 스마트 컨트랙트 데이터 셋을 기반으로 더 정확한 라벨링 가이드라인을 개발하고 개선해야 한다. 특히 취약점이 있는 데이터의 개수가 상대적으로 적어, 모델에서 취약점이 있는 클래스의 재현율이 떨어지는 것을 확인하였다. 이는 모델의 학습과 성능을 향상시키는 데에 데이터의 개수와 라벨링 작업이 중요한 원인으로 작용하기 때문이다.

    - 지속적인 CNN 모델의 평가와 성능 향상 : CNN 모델의 성능을 지속적으로 평가하고 향상시키는 연구가 필요하다. 새로운 아키텍쳐나 기술의 도입을 검토하고, 모델의 정확성과 신뢰성을 개선해야 한다.

    - 더 정확한 이미지 Localization을 통한 취약점 분석 : 이미지 지역화 기술을 더 정교화하여 취약점의 위치와 그 원인까지 식별할 수 있는 방법에 대한 연구가 필요하다.

5.1. 멘토의견서 반영 결과

1 프로젝트 검토 의견에 대한 수정 사항

1.1 학습 데이터 라벨링 작업 수정 : **3.2.1.**에서 언급하였듯이, 보안 취약점 삽입 도구인 SolidiFi를 이용해 적절한 라벨링 가이드라인을 수립하였다.

1.2 CNN 모델 사용 : **3.4.** 에서 CNN 모델을 사용해 93%의 성능을 달성하였다.

1.3 XAI 모델 선정 : Grad-CAM을 이용하여 취약점을 찾는 데에 가장 많은 영향을 끼친 이미지의 인덱스를 찾아 웹 인터페이스에 통합하였다.

1.4 프로젝트 전체 시스템 흐름 : 프론트엔드와 백엔드에서 요청 및 응답을 주고 받아 사용자에게 취약점의 위치와 원인을 해석해주었다.

2 향후 진행 계획에 대한 조언에 따른 수정 사항

2.1 데이터셋 확장과 개선 : Reentrancy 취약점 외의 Access Control, Tx.origin, Time Manipulation의 경우 학습 데이터 수집에 어려움이 있었다. 학습에 필요한 수만큼의 충분한 데이터를 수집할 수 없음을 판단해, Reentrancy 취약점 유형에 초점을 맞추는 모델을 개발하였다. 또한 재진입 공격은 다른 3가지 공격 유형에 비해 더 복잡한 시나리오에서 발생이 가능하며, 특정한 패턴에 의존하지 않고 다양한 방법으로 수행될 수 있어 취약점 감지가 어렵다. 이러한 이유로 재진입 공격 취약점과 관련된 데이터셋을 수집하였다.

2.2 웹 인터페이스(UI) 완성 : 사용자 편의성을 고려하여 안드로이드, iOS, 웹에서 동작 가능한 크로스 플랫폼 플러터(Flutter)를 사용하였다.

6. 참고 문헌

[1] smartbugs. smartbugs-curated: https://github.com/smartbugs/smartbugs-curated

[2] Hwang, Seon-Jin, et al. “CodeNet: Code-targeted convolutional neural network architecture for smart contract vulnerability detection” IEEE Access 10 (2022): 32595-32607

[3] 황도연; 김정구; 최윤호. “인공지능 기반 스마트 컨트랙트 취약점 탐지 연구 동향 분석”, 한국통신학회 학술대회논문집, 66-67, 2021

[4] 방지원; 최미정. “스마트 컨트랙트 취약점 탐지 도구 동향 분석”, KNOM Review 25, no.1, 49-61, 2022

[5] ConsenSys, (2018). 이더리움 개발자 도구 리스트 [Online]. Available : https://github.com/ConsenSys/ethereum-developer-tools-list/blob/master/README_Korean.md

[6] Flutter. Flutter documentation [Online]. Available : https://docs.flutter.dev

[7] Django. Django Documentation [Online]. Available : https://docs.djangoproject.com/en/3.2/

[8] Tensorflow. Tensorflow v2.14.0. Python. [Online]. Available : https://www.tensorflow.org/api_docs/python/tf