[부산대학교 정보컴퓨터공학부]

2023 학년도 전기 졸업과제 중간보고서

과제명 - (블록체인 보안) Smart-contract 취약점 탐지 툴

지도교수 : 최윤호 교수님

팀명 : 컴쪽이 (팀 번호 : 51, 분과 : D)		
팀원 번호	**학번**	**이름**
1	201924437	김윤하
2	202055571	윤지원
3	202055616	최지원

목차

1. 요구조건 및 제약 사항 분석에 대한 수정사항

1-1. 기존 요구 조건 및 수정사항

1-1-1. 취약점 탐지 방향 수정

본 졸업과제에서 개발하고자 한 기존 Cross Contract 취약점 탐지 소프트웨어는 Smart Contract의 Reentrancy, Access Control, Tx.origin, Time Manipulation의 4가지 취약점을 탐지하는 모델로 설계하였다. 그러나 Acccess Control, Tx.origin, Time Manipulation의 경우 학습 데이터 수집에 어려움이 있었다. 또한 이 중 재진입(Reentrancy) 공격은 다른 취약점들과 비교해 더 복잡한 공격 유형으로, 코드의 문맥을 파악하고 이해하는 것이 어렵다. 이는 재진입 공격이 상태 변경과 외부 호출과 관련된 복잡한 시나리오에서 발생 가능하며, 특히 계약 간 상호 작용이 많은 경우에 발생할 가능성이 높기 때문이다. 또한 재진입 공격은 특정한 패턴에 의존하지 않고 다양한 방법으로 수행될 수 있어서 취약점 감지가 어렵다. 이러한 이유로 우리는 스마트 컨트랙트 개발에서 가장 주의해야 할 취약점 중 하나인 재진입 공격 취약점에만 초점을 맞추는 모델을 개발하는 것을 목표로 수정하였다.

1-1-2. 학습 데이터 라벨링 작업 수정

기존 사용할 취약점 탐지 툴은 MythX, Security, Oyente, SmartCheck 가 있었고, 수동 검토도 함께 진행할 예정이었다. MythX 의 경우 20,000 개의 데이터 셋을 탐지하는 데에 약 $500 의 유료화 버전이 있어서 그를 대신해 오픈소스 이더리움 가상머신 바이트코드의 보안 분석 도구인 Mythril Classic 을 사용해 보았다. 솔리디티 파일 하나당 약 5 분의 시간이 소요되었고, 총 2 만 개가 넘는 데이터를 분석하기 위해 약 70 일 이상의 시간이 걸리는 것으로 예측되었다. 이는 졸업과제 시기에 상당히 오래 걸리는 작업이었다. 또한 다른 취약점 탐지 도구들도 취약점 탐지율이 매우 낮거나 시간이 매우 오래 걸리는 경우가 많았다. 이에 우리는 기존 분석 도구들을 사용해 레이블링 하는 방식 대신에 CodeNet 논문[1]에서 사용한 데이터 셋을 확보하여 사용하였다. 해당 논문에서는 취약점 삽입 도구인 SolidiFi 로 취약점을 생성한 데이터를 사용해 연구를 진행하였고, 관련하여 총 24,367 개의 데이터 셋을 확보하였다.

1-1-3. CNN 모델 사용

CNN은 이미지 처리에서 주로 사용되는 신경만 구조로, Convolution과 Pooling 레이어를 반복하여 사용하며, 이미지의 특징을 추출하고 분류하는 데 효과적이다. 또한 스마트 컨트랙트 코드의 취약점은 특정 코드 블록에서 발생하며, 이러한 지역적 특징을 잘 파악하는 CNN이 탐지 모델로 적합하다고 판단하였다. 따라서 취약점 탐지 모델으로 CNN을 선택하였다.

---

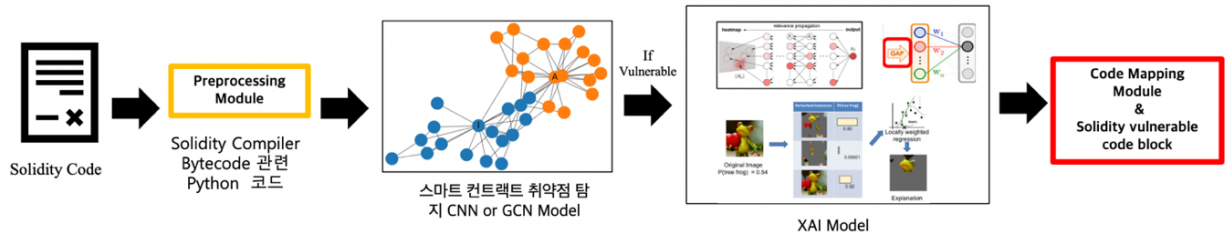
[1] Hwang, Seon-Jin, et al. "CodeNet: Code-targeted convolutional neural network architecture for smart contract vulnerability detection." IEEE Access 10 (2022): 32595-32607.

1-1-4. XAI 모델 선정

XAI(eXplainable Artificial Intelligence)는 인공지능 모델의 결과를 해석 및 설명 가능하도록 만드는 기술이다. 이는 모델의 동작과 의사결정 과정을 사용자가 쉽게 이해할 수 있도록 설명하는 것을 목표로 한다. 우리는 Code Mapping 단계에서 이미지 Localization에 대한 실제 취약점을 확인하기 위해, Grad-CAM의 XAI 기술을 활용한다. Grad-CAM은 특정이미지의 분류 결과에 영향을 미치는 픽셀 영역 시각화 방법 중 하나이다. 이로 해당 OP Code의 위치를 확인해, 그를 기존 Solidity 코드로 매핑하여 어디에서 취약점이 발생했는지 사용자에게 알려주기 용이하다. 이러한 의사결정 과정을 통해 모델의 신뢰성을 높인다.

2. 설계 상세화 및 변경 내역

2-1. 시스템 전체 구조

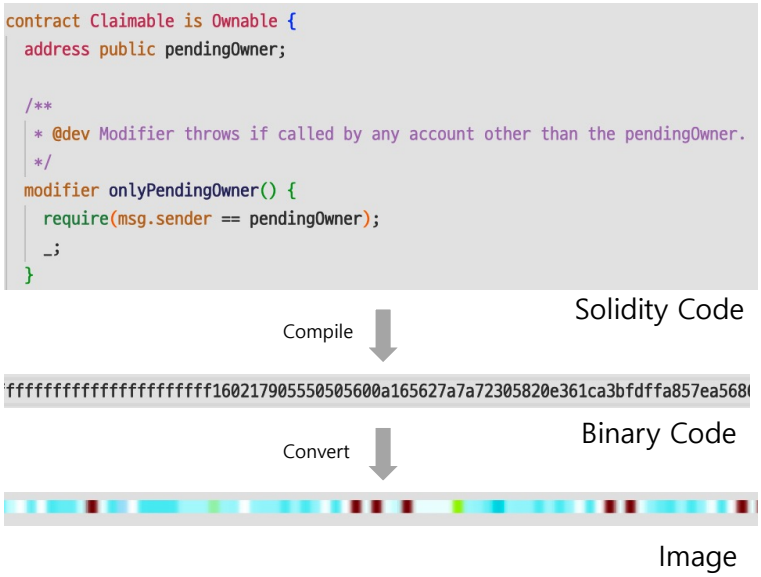


[그림 1] 시스템 전체 구조

우리가 개발하는 '(블록체인 보안) Cross Contract 취약점 탐지 소프트웨어'는 Smart 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 취약점의 악용을 방지할 뿐만 아니라 프로그래머가 코드 작성 시 취약점에 유의해 코드를 작성할 수 있도록 돕는다.

2-1-1. 전처리 모듈

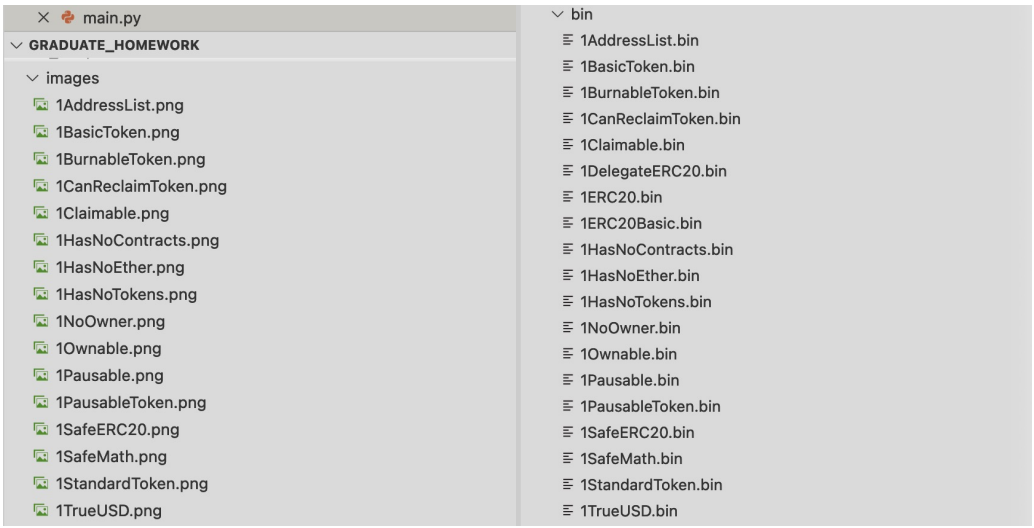
데이터 전처리 단계에서는 Solidity 코드 컴파일을 시행한다. 컴파일이 완료되면 각 컨트랙트가 바이너리 파일로 저장된다. 이 Binary 파일은 Encoding 과정을 거쳐 이미지 형태로 저장된다. 전체 전처리 과정을 그림으로 표현하면 다음과 같다.



[그림 2] 전처리 과정

또한 다음은 하나의 솔리디티 코드를 전처리해 이미지화 시킨 결과의 예이다. 아래 그림을 살펴

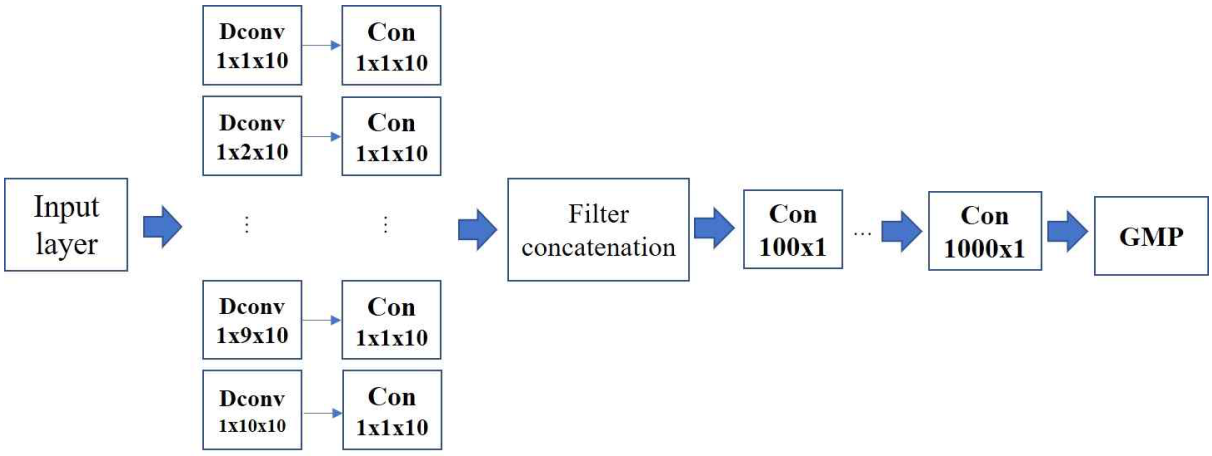
보면, 각 컨트랙트 별로 .bin 파일이 생성되며 image 파일 또한 .png로 생성한 것을 확인할 수 있다. 상세한 설명은 5-1의 Data Preprocessing 단계에 설명되어 있다.



[그림 3] 솔리디티 코드 이미지화

2-1-2. CNN Model 설계

CNN Model 은 해당 Image 파일이 취약점을 갖고 있는가 / 아닌가의 두 가지로 분류한다. 우리 팀은 위에서 언급한[2] "이미지 Localization 과 딥러닝 분류 기법을 활용한 스마트 컨트랙트 재진입 공격 취약점 위치 탐지 방법" 주제로 연구를 진행한 논문을 참고하여 CNN Model 을 설계할 예정이다. 해당 논문에서는 다음과 같은 구조로 모델을 제안하였다. 이는 스트라이드 없이 코드화된 이미지가 학습이 가능한 모델이다.



[그림 4] 해당 논문에서 제안한 모델

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

2-1-3. XAI Model

준비된 모델에 프론트에서 이미지로 처리한 입력이 들어오면 그에 취약점이 있는지 없는지 판별한다. XAI 모델인 Grad-CAM에선, 모델에 넣은 이미지가 취약점이 있다면 어느 Code Block에서 취약점이 발생했는지를 보여주는 위치 매핑 작업을 시행한다.

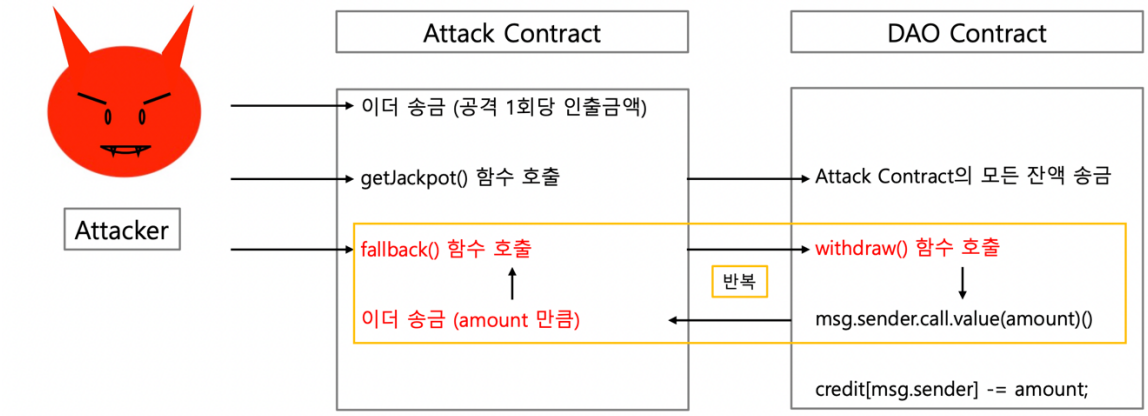
2-1-4. Code Mapping 모듈

위의 Grad-CAM에서 위치 매핑이 완료되어 취약점이 탐지된 이미지의 위치 정보를 얻으면, 그에 대응하는 OP Code를 찾아낸다. 그 후 대응되는 본래의 입력인 Solidity Code를 찾아 사용자에게 출력해준다.

2-2. 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)();
	  }
	}

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

2-3. 전체 데이터베이스 설계

2-3-1. 솔리디티 파일 정보

- 해당 코드의 바이트코드를 통해 얻은 이미지와 연결하여 원본 솔리디티 파일을 추적하는 데 사용하기 위해 취약점을 탐지할 솔리디티 파일의 정보를 저장한다.

2-3-2. 이미지 파일 및 바이트 코드 정보

- 솔리디티 파일을 컴파일해 얻은 바이트 코드와 CNN 모델에 입력으로 사용하기 위해 바이트 코드를 이미지 파일로 변환하여 저장한다.

2-3-3. CNN 모델의 예측 결과 정보

- 이미지에 대한 분류 결과를 추적하고 분석하거나 통계를 내기 위해 사용할 수 있도록 CNN 모델에 이미지를 입력하고 얻은 결과를 저장한다.

3. 갱신된 과제 추진 계획

3-1. 현재까지의 과제 진행

6월				7월				
2주	3주	4주	5주	1주	2주	3주	4주	5주
Smart contract 취약점 스터디								
		취약점 data set 라벨링 작업						
				전처리 모듈 구현				
								중간 보고서 작성

3-2. 향후 과제 계획

8월					9월			
1주	2주	3주	4주	5주	1주	2주	3주	4주
중간 보고								
	CNN Model 구현							
	서버 구축							
		XAI Model feature 생성						
			Code mapping 모듈 개발					
			UI 디자인 설계					
				UI 기능 개발 구현				
					테스트 & 디버깅			
						오류 수정		
						최종 발표 & 보고서 준비		

4. 구성원별 진척도

구성원	구성원별 진척도
김윤하	- Solidity 코드 라벨링
윤지원	- Solidity 코드 컴파일러 파이썬 모듈 개발
최지원	- 프론트엔드 : 플러터 UI 화면 구성 및 설계
공통	- 보고서 작성 - 바이트코드 이미지 변환 작업

5. 보고 시점까지의 과제 수행 내용 및 중간 결과

[참고] 팀원들과의 자료 공유는 컴쪽이 졸업과제 Github(https://github.com/Capstone-2023-1-51/2023-1-51)에서 확인이 가능합니다.

5-1. Data Preprocessing

데이터 전처리 단계에서는 라벨링된 Solidity 파일들을 컴파일하여 바이트코드 파일을 생성하고, 그 바이트 코드들로 이미지를 생성하였다.

5-1-1. 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.

5-1-2. 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)
        pixel_colors.append((r, g, b))
    return pixel_colors

def create_image(file_name, pixel_colors):
    width = len(pixel_colors)
    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('./images/' + 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_pixels(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 OP 코드들은 G, B 값을 0 으로 저장하고, PUSH 명령어들은 push 딕셔너리를 사용하여 특정 바이트 코드에 해당하는 G, B 값을 계산한다.
3. **create_image(file_name, pixel_colors) 함수** : 생성된 픽셀 색상값을 이용하여 이미지를 생성하고 지정된 파일 이름으로 이미지를 저장한다. 픽셀 색상값 리스트를 이용하여 OP 코드는 좌우의 순서 외에 위아래 코드와의 관계에서 큰 정보를 얻기 어려우므로, 이미지의 세로 크기는 1로 고정하고, 가로 크기가 픽셀 수와 같은 이미지를 생성하였다. 픽셀 색상값을 이미지에 적용한 뒤 해당 이미지를 지정된 파일 경로에 PNG 형식으로 저장한다. 추후에 CNN 모델의 입력으로 사용하기 위해 가로 크기를 고정된 특정 값으로 지정할 예정이다.

5-1-3. main.py

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

def main():
    versions = set()    # 설치된 버전
    folder_path = '솔리디티 파일 위치'
    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:
```

```
        version_in_use = version
        version = compileSol.extract_compiler_version(file)
        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'+str(file_num)+'.txt', 'w') as f:
                s = version + '₩n' + file + '₩n' + str(e)
                f.write(s)
            continue

        keys = compiled_sol.keys()  # 해당 파일 안에 있는 컨트랙트 이름
        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():
                    if function['name'].startswith('reenbug'):  # 컨트랙트 내에 'reenbug'로 시
작하는 이름의 함수 존재
                        label = '1'
                        break

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

            file_name = str(file_num) + strs[1]
```

```
            try:
                pixel_colors = createImage.create_pixels(bytecode)
            except Exception as e:
                with open('./exception' + str(file_num), 'w') as f:
                    f.write(version + '₩n' + file + '₩n' + str(e))
                continue
            createImage.create_image(file_name, pixel_colors)
        file_num = file_num + 1

main()
```

"main.py"는 glob 모듈을 활용하여 지정된 폴더 경로에서 모든 솔리디티 파일들의 리스트를 가져온다. 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 루프를 사용했다. 이 과정에서 다음과 같은 작업들을 수행한다.

1. 솔리디티 파일의 컴파일러 버전 확인 :
    - 솔리디티 파일 내에 컴파일러 버전 정보를 추출하여 확인한다.
    - 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 :
    - 버전 정보가 변경되었을 경우, 해당 버전을 설치하고 선택한다.
3. 소스 코드 컴파일 및 바이트 코드 저장 :
    - 솔리디티 파일을 컴파일하여 컨트랙트별 바이트 코드를 추출한다.
    - 추출한 바이트 코드를 파일로 저장한다.
4. 바이트 코드를 이용한 이미지 생성 및 저장 :
    - 추출한 바이트 코드를 RGB 픽셀 값으로 변환하여 이미지를 생성한다.
    - 생성된 이미지를 지정된 파일 경로에 PNG 형식으로 저장한다.

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

5-2. 프론트엔드 진행 상황

5-2-1. 전체 시스템 구성도

[그림 6] 전체 시스템의 상호작용 구성도

전체 시스템 흐름은 다음과 같다. 이는 웹 실제 구현에 따라 상세한 부분이 달라질 수 있다.

1. 사용자는 모바일 앱을 통해 시스템 프론트엔드에 접속한다.
2. 프론트엔드에선 사용자로부터 Solidity Code의 입력을 받고, 사용자의 요청을 받아 백엔드로 전달한다.
3. 백엔드에선 받은 요청을 처리하여 필요한 로직을 수행하며, 필요한 데이터를 DB에 요청해 얻는다. 해당 로직은 다음과 같다.
   A. 해당 Solidity Code를 main.py를 통해, 각 컨트랙트를 먼저 .bin으로 변환한 후 그에 대응하는 이미지를 .png로 생성한다.
   B. 생성된 이미지를 CNN 모델에 넣어 결과값을 받는다.
      i. 만약 이미지에 취약점이 존재하지 않는다면, 그에 대한 response를 생성한다.
      ii. 만약 이미지에 취약점이 존재한다면, 해당 결과값을 XAI와 Code Mapping 모듈로 이미지에 대응되는 OP Code의 위치를 확인해 기존의 Solidity Code의 어디에서 취약점이 발생했는지를 리턴해 response를 생성한다.

전체 시스템의 각 구성 요소들은 다음과 같다.

1. Frontend(프론트엔드) : Flutter
   - UI(사용자 인터페이스)를 구현하고, 사용자와의 상호작용을 담당한다.
   - 플러터는 모바일 어플리케이션 개발을 위한 프레임워크로, Android와 IOS 둘 모두에서 작동하는 단일 코드 베이스로 앱 개발이 가능하다는 장점이 있다.
2. Backend(백엔드) : Python, Django 프레임워크
   - 요청-응답 사이의 로직을 처리하고, 데이터를 관리한다.
   - Django는 Python 기반의 웹 어플리케이션 프레임워크이며, 웹 개발을 쉽게 하도록 도와준다.
3. Databases(데이터베이스) : MySQL
   - MySQL은 오픈소스 기반의 DB 관리 시스템(RDBMS)으로, 데이터를 효율적이고 안정적으로 관리할 수 있다.