ETH2VEC

LEARNING CONTRACT-WIDE CODE REPRESENTATIONS FOR VULNERABILITY DETECTION ON ETHEREUM SMART CONTRACTS

REFERENCE

- Ashizawa, Nami, et al.
  "Eth2Vec: Learning Contract-Wide Code Representations
  for Vulnerability Detection on Ethereum Smart Contracts."
  arXiv preprint arXiv:2101.02377 (2021).

ABSTRACT

ABSTRACT

- 이더리움 스마트 컨트랙트
  - 이더리움 블록체인 위에서 구동되는 프로그램
- 그간 많은 스마트 컨트랙트 취약점들이 밝혀짐
  - 많은 보안 분석 도구들이 개발되었음
  - 그러나 코드가 다시 쓰여지면(rewritten) 성능이 크게 떨어짐

ABSTRACT

- Eth2Vec
  - 취약점 발견을 위한
  - 머신 러닝 기반 정적 분석 도구
  - 코드 재작성에 강인함

ABSTRACT

- 기존의 취약점 발견을 위한 머신러닝 기반 정적 분석 도구
  - 입력으로, 분석가가 수동으로 만든 특징(feature)들이 필요함
- Eth2Vec
  - EVM 바이트코드(bytecodes)를 통해 자동으로 특징들을 학습
  - 목표(target) EVM 바이트코드와 이미 학습된 EVM 바이트코드 간 유사도를 비교

ABSTRACT

- 실험 결과
  - Eth2Vec 성능이 기존 방법들 성능 대비 우위
  - 재사용된 코드 취약점도 잡아냄

INTRODUCTION

INTRODUCTION: BACKGROUNDS

- 스마트 컨트랙트의 보안
  - 솔리디티 등 스마트 컨트랙트를 작성하는 프로그래밍 언어의 복잡성 때문에 어려움
- 블록체인에 배포된 스마트 컨트랙트는 수정이 불가능
  - 지난 몇 년간 많은 공격들이 보고되어 왔음
  - 따라서 배포 전 취약점 분석이 필요함
  - 다양한 분석 도구들이 개발되어 왔음

INTRODUCTION: BACKGROUNDS

- 본 논문에서는
  - 정적 분석 도구 개발에 초점을 맞춤
  - 스마트 컨트랙트 코드의 다양한 취약점을
  - 높은 처리량을 가지고 분석할 수 있는 도구
- 정적 분석에서는
  - 실행 없이, 오직 소스 코드만이 분석 타겟이 됨
  - 따라서 배포 전에 취약점을 찾고 방지할 수 있음

INTRODUCTION: BACKGROUNDS

- 그러니 정적 분석은 일반적으로 두 문제를 가짐
  - 1. 취약점 발견의 정확도에 한계가 있음
  - 2. 분석 시간이 길 수 있음
- 예를 들어, 심볼릭 실행 (symbolic execution)
  - 제어 흐름 그래프(CFG)를 타겟 코드에서 추출할 수 있음
  - 그러나 모든 상태를 탐색하므로 시간이 많이 요구됨

INTRODUCTION: BACKGROUNDS

- 이러한 문제를 해결하는 것은 머신러닝
  - 머신러닝 기반 정적 분석은
  - 코드의 학습 특징들로부터 취약 여부를 판별
- 그러나 이 역시도 한계를 가짐
  - 1. 코드 재작성으로 정확도를 낮출 수 있음
  - 2. 적절한 특징이 정의되어 있지 않음

INTRODUCTION: BACKGROUNDS

- 1. 코드 재작성으로 정확도를 낮출 수 있음
  - 같은 문맥(semantic)의 함수라도
  - 구조가 다르면 다른 분석 결과를 야기할 수 있음
  - 코드의 구조가 분석에 영향을 크게 끼치기 때문

INTRODUCTION: BACKGROUNDS

- 2. 적절한 특징이 정의되어 있지 않음
  - 스마트 컨트랙트의 취약점을 나타내는 특징은 잘 알려진 바 없음
  - 서로 다른 코드를 망라하는 강인한 특징이 명확하지 않음

INTRODUCTION: CONTRIBUTIONS

- Eth2Vec
- 자연어 처리 신경망에 기반함
  - 오직 목표 스마트 컨트랙트 코드만을 입력으로 함
- 스마트 컨트랙트 코드의
  - EVM 바이트코드
  - 어셈블리 코드
  - 추상 구문 트리(Abstract Syntax Tree, AST)로부터 학습

INTRODUCTION: CONTRIBUTIONS

- 특히 코드 재작성에 강인하다는 점이 주요 기여
  - 전형적 머신러닝 알고리즘은
  - 재작성으로부터 잘못된 결과를 도출함
- 이는 코드 그 자체의 기저에 있는 특징을 학습하는 것이 아닌
  - 오직 기술(description)에만 의존하기 때문
- 그렇다고 특징들을 찾는 것 역시 어려운 일
  - 잘 모르기 때문

INTRODUCTION: CONTRIBUTIONS

- Eth2Vec
- 자연어 처리를 위한 신경망을 이용해
  - 특징 추출 페이즈(phase)를 거쳐
  - 학습 특징들을 자동으로 학습함
- 정리하자면, “신경망을 독립적으로 사용해 특징 추출을 수행함”
  - EVM 바이트코드를 입력으로 받는 신경망

INTRODUCTION: CONTRIBUTIONS

- 실험
  - Etherscan으로부터 5,000개의 스마트 컨트랙트 파일을 받아 사용
  - 컨트랙트 당 1.2초만에 취약점을 발견함
  - 정확도 77.0%
- 재진입(reentrancy) 문제가 가장 많이 발견됨
  - 정확도 86.6%

INTRODUCTION: CONTRIBUTIONS

- 기존 서포트 벡터 머신(SVM) 기반 연구 대비
  - 성능이 우수했으며
  - SVM-기반 방법이 찾지 못한 재작성 코드 및 그들의 취약점까지도 발견

FORMULATION

PROBLEM FORMULATION

- 스마트 컨트랙트 분석을 위한 접근법을 공식화
- 각 스마트 컨트랙트 $c_i \in C$ 는 취약점들 $V_i = \{v_1^i, \ldots, v_l^i\} = \mathscr{V}^l$ 을 포함
  - $\mathscr{V}$: 독립적인 취약점들
  - l : 임의의 숫자

PROBLEM FORMULATION

- 모델 M의 입력
  - 주어진 정수 $n \in N$ 에 대하여
  - 컨트랙트와 취약점의 조합 $CV = \{(c_1, V_1), \ldots, (c_n, V_n)\}$ 와
  - 테스트 컨트랙트 $c_t \in C$

PROBLEM FORMULATION

- 모델 M의 출력
  - $\{\epsilon_i^{c_t}\}_{i\in[1,d]} \subseteq \mathbb{R}^d$
  - $d = |\mathscr{V}|$ 개의 원소를 가짐
  - $\epsilon_i^{c_t}$: $\mathscr{V}$의 취약점에 대한 가능성

PROBLEM FORMULATION

- 정리하자면
- $M(CV, c_t) \rightarrow \{\epsilon_i^{c_t}\}$ 를 최적화하는 도구를 개발하는 것이 목표
  - 컨트랙트와 취약점 집합을 입력으로 받아
  - 각 컨트랙트의 각 취약점에 대한 가능성 정도를 출력

ETH2VEC

ETH2VEC

- Eth2Vec은
  - 심지어 취약점의 본질적 특징이 명확하지 않은 코드에 대해서도
  - 취약점 분석이 가능
  - 신경망은 블랙박스의 관점에서 (block-box manner) 특징을 다루기 때문

ETH2VEC

- 코드 유사도를 계산
  - 각 단어와 단락 등을 벡터화하는 Word2Vec과 유사
- 코드의 보안성이 분석되면
  - 취약한 코드를 모델이 학습하고
  - 이를 통해 유사한 코드의 취약점을 식별할 수 있음

ETH2VEC

- 자연어 처리를 위해
  - EVM 바이트코드를 다루기 위해
  - PV-DM 모델을 사용

ETH2VEC

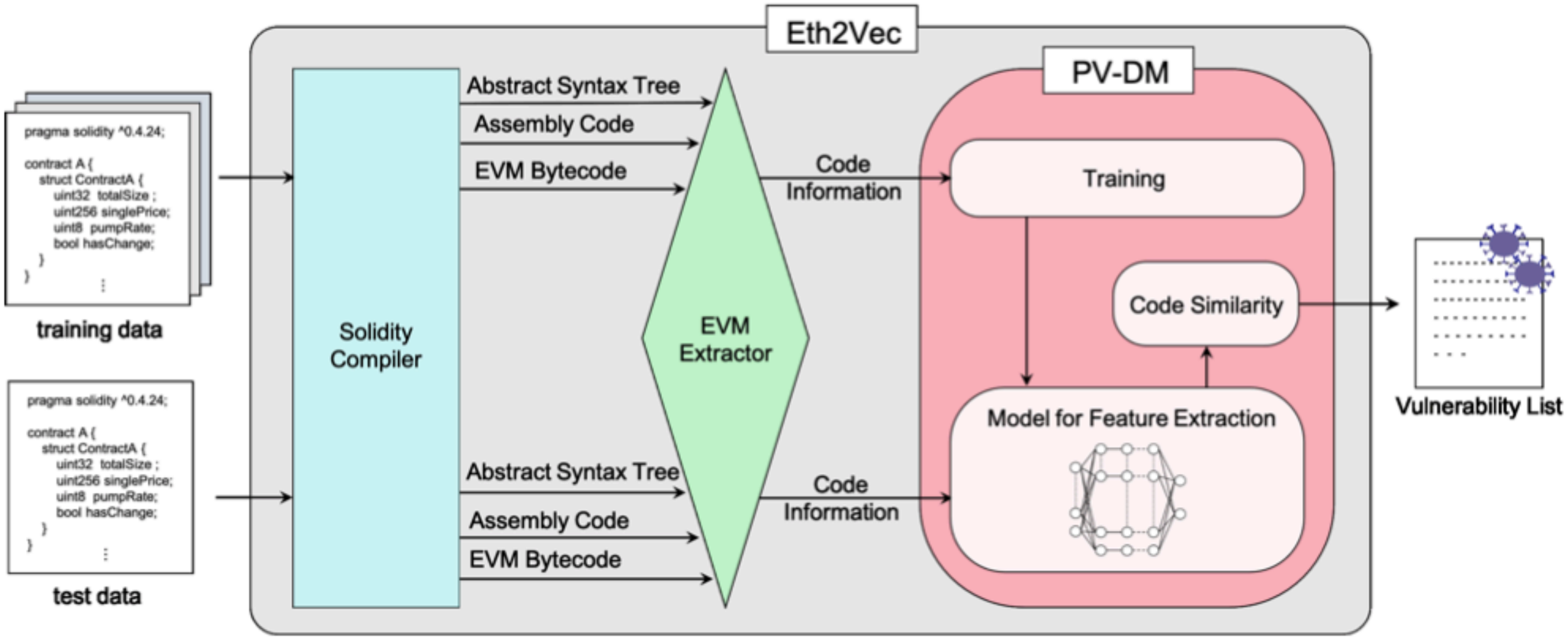
- PV-DM 모델은 문서의 토큰을 기반으로 문서의 표현을 학습
- 그러나 문서와 프로그램 코드는 차이가 있으므로
  - 프로그램 코드는 그래프로 나타낼 수 있음
  - 특정 구문을 가짐
- EVM 추출기(Extractor)라 불리는 모듈을 개발
  - 코드의 AST를 나타냄

ETH2VEC

- 결국 Eth2Vec은 두 개의 모듈로 구성됨
  - 단락을 다루는 신경망인 **PV-DM 모델**
  - 솔리디티 소스 코드로부터 PV-DM 모델의 입력을 만드는 **EVM 추출기**

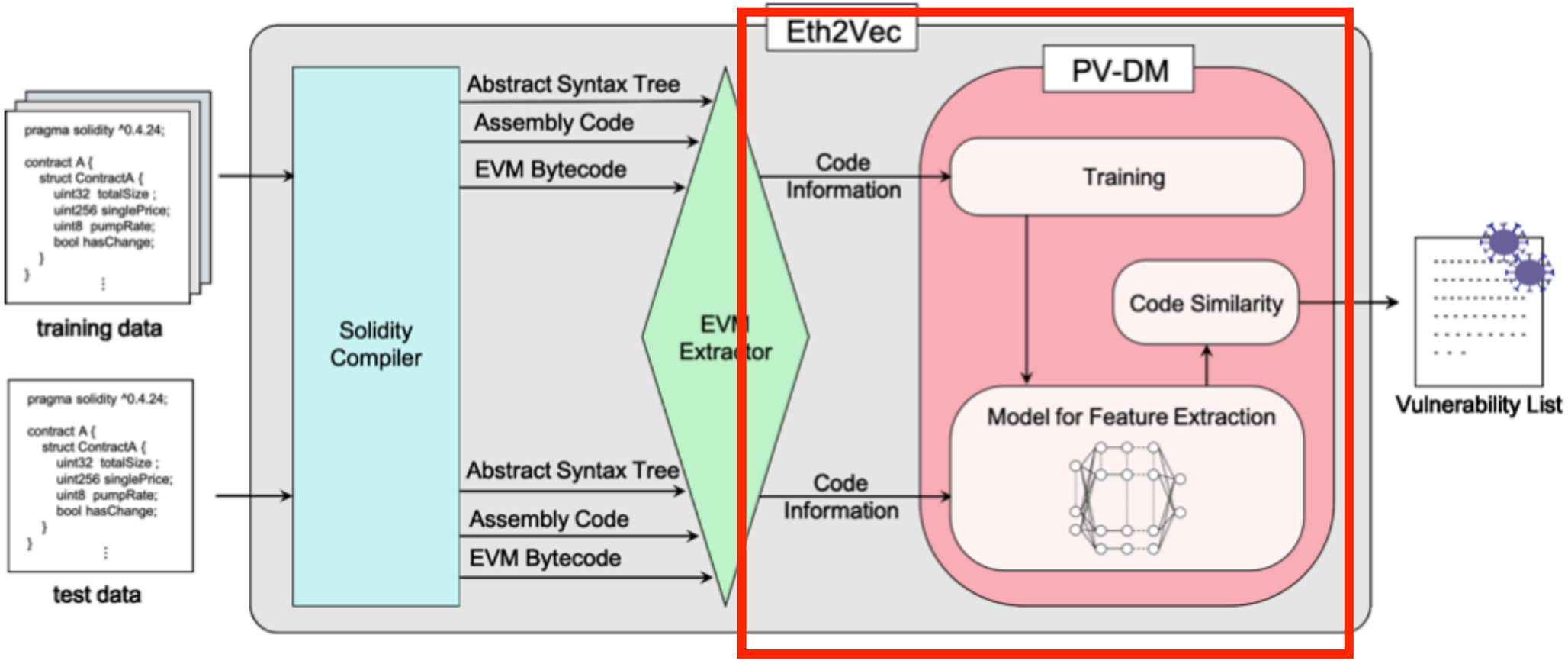
ETH2VEC

- 오버뷰


Eth2Vec
PV-DM
pragma solidity ^0.4.24;
contract A {
struct ContractA {
uint32 totalSize ;
uint256 singlePrice;
uint8 pumpRate;
bool hasChange;
}
}
training data
pragma solidity ^0.4.24;
contract A {
struct ContractA {
uint32 totalSize ;
uint256 singlePrice;
uint8 pumpRate;
bool hasChange;
}
}
test data
Solidity Compiler
Abstract Syntax Tree
Assembly Code
EVM Bytecode
EVM Extractor
Code Information
Abstract Syntax Tree
Assembly Code
EVM Bytecode
Code Information
Training
Code Similarity
Model for Feature Extraction
Vulnerability List

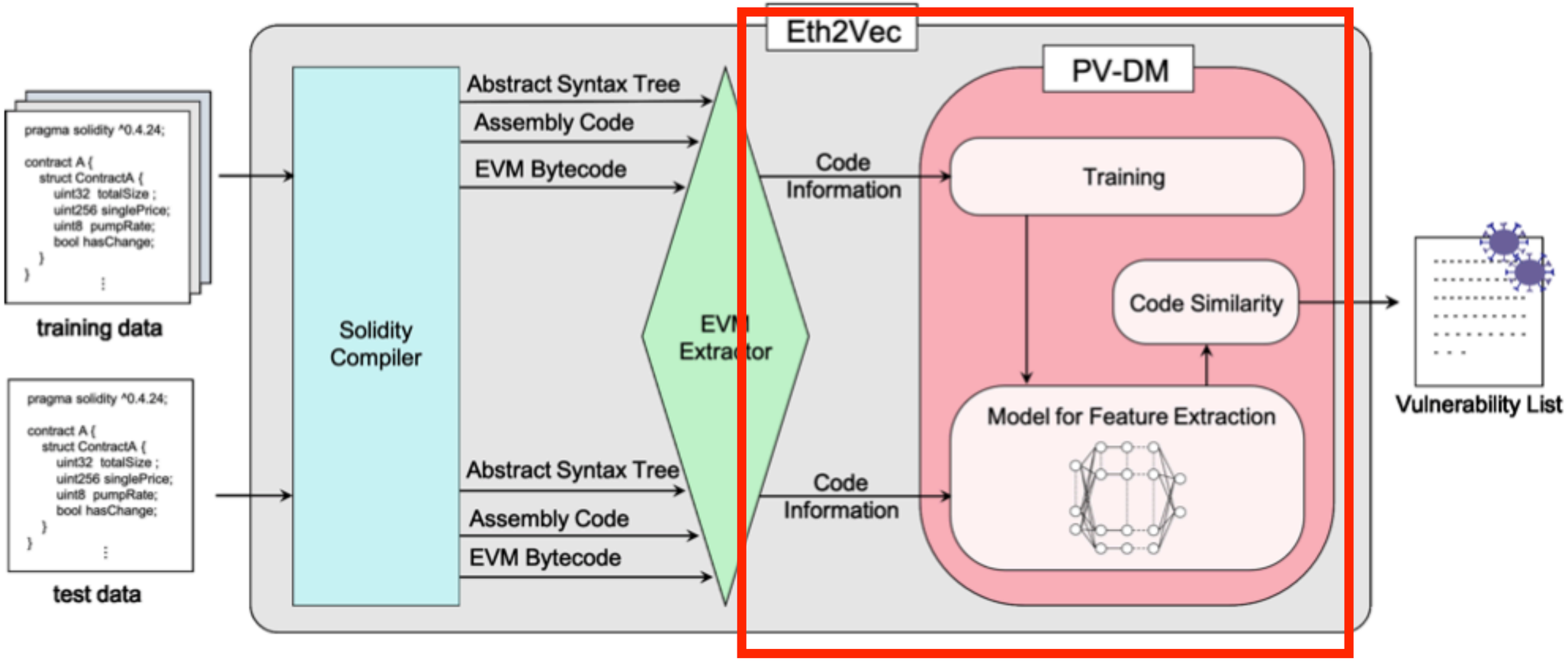
ETH2VEC

- PV-DM 모델은 바이트코드로부터 생성된 JSON 파일을 입력으로 함



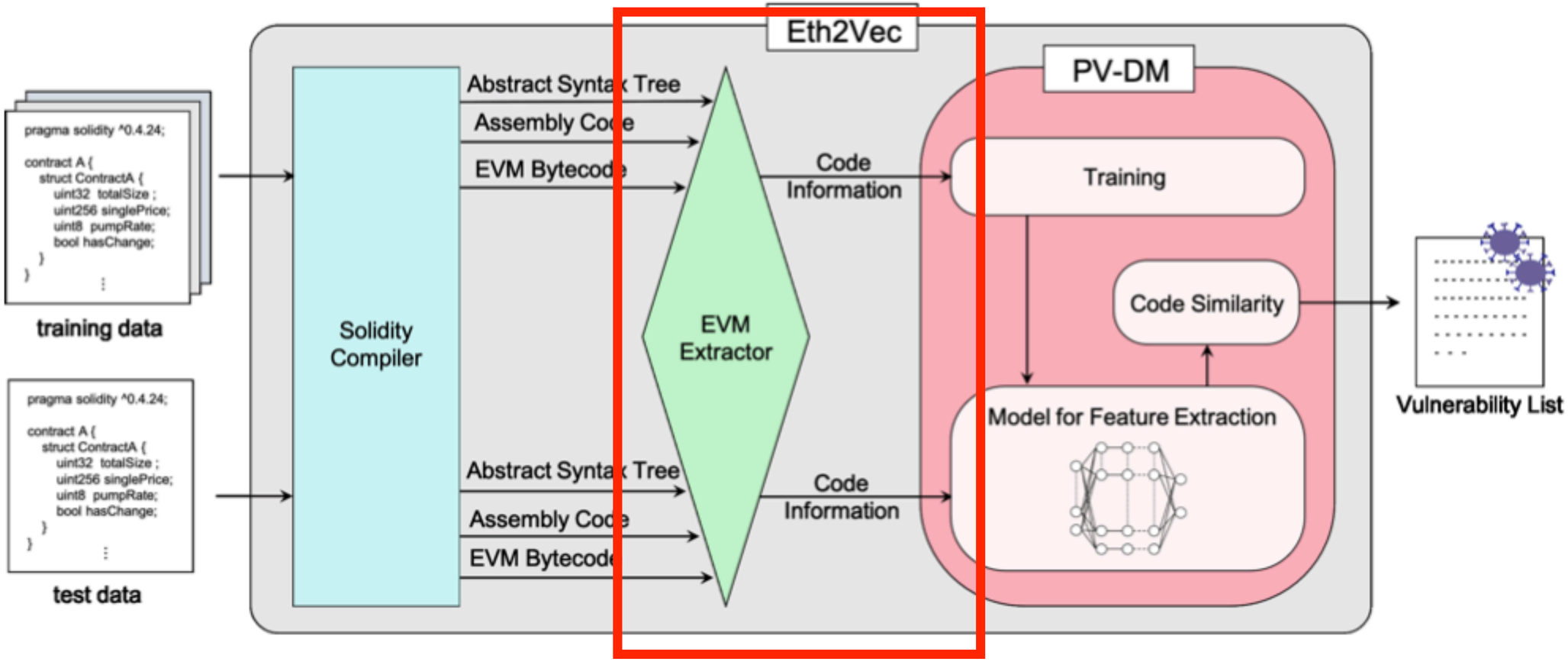
ETH2VEC

- 각 컨트랙트의 코드 유사도를 계산



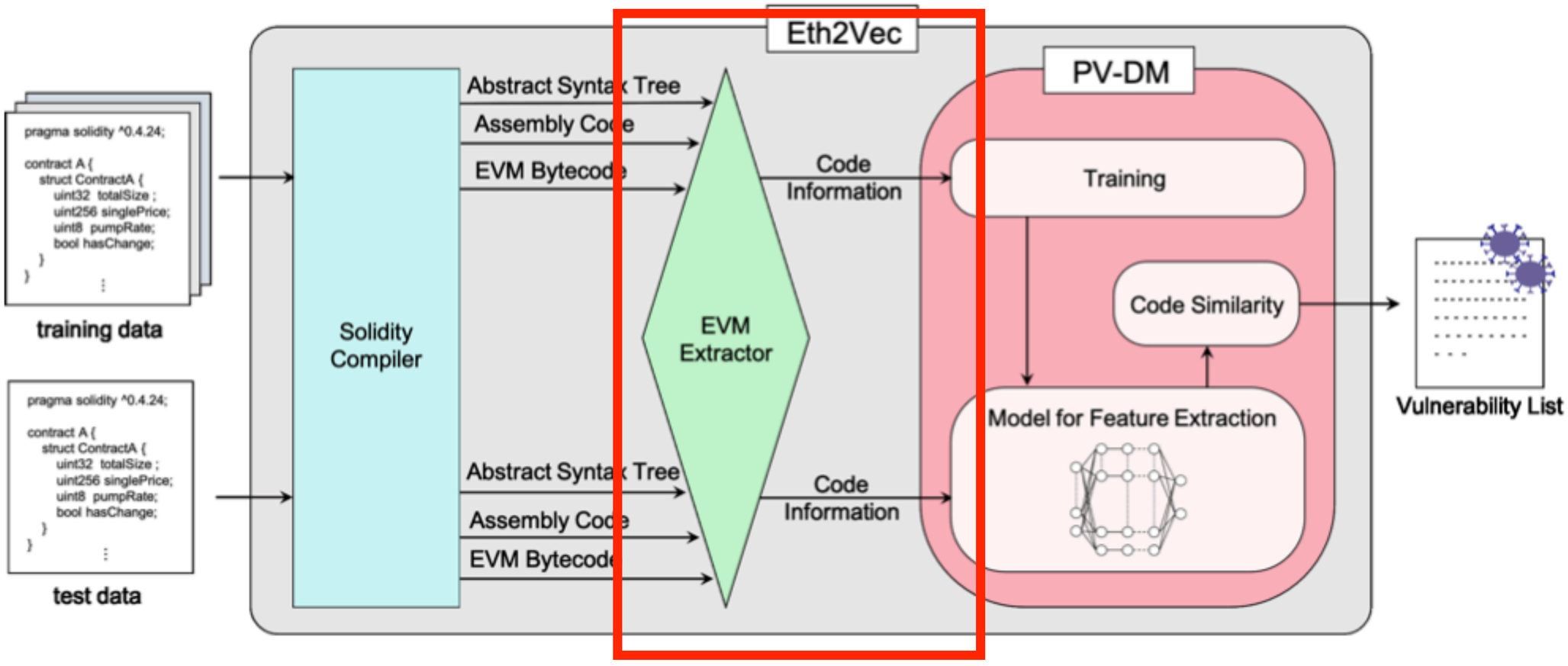
ETH2VEC

- EVM 추출기는 PV-DM 모델의 입력을 생성; PV-DM이 바이트코드를 직접 다룰 수 없으므로



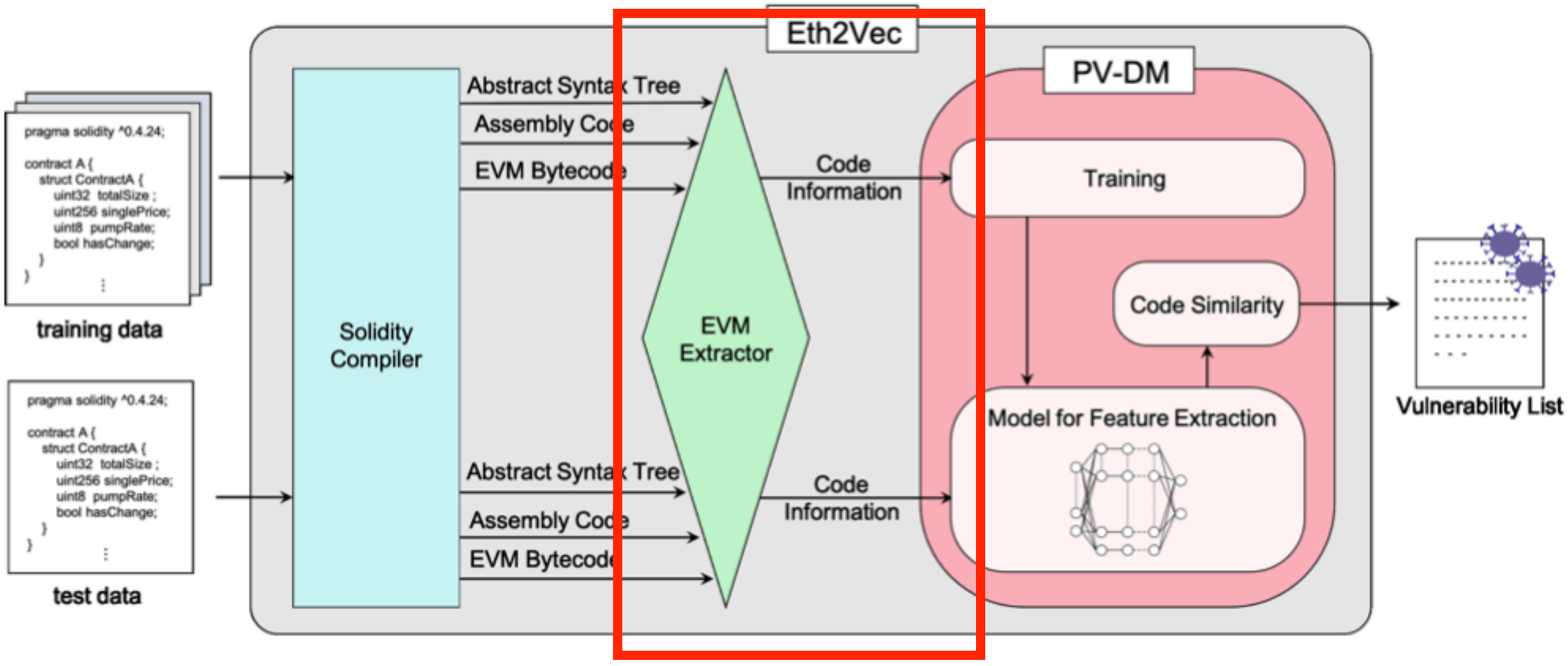
ETH2VEC

- EVM 바이트코드를 구문상으로 파악하고



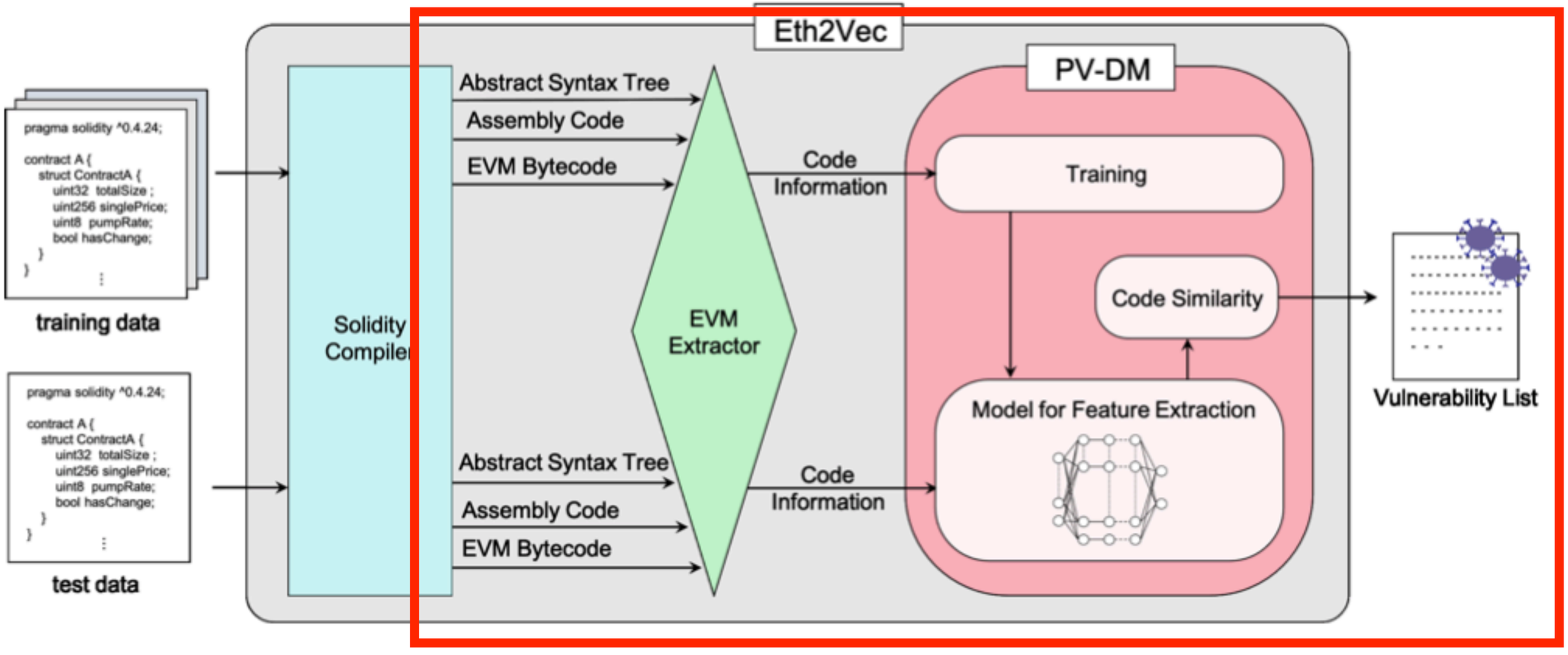
ETH2VEC

- 명령어 수준, 블록 수준, 함수 수준, 컨트랙트 수준으로 JSON 파일을 생성함



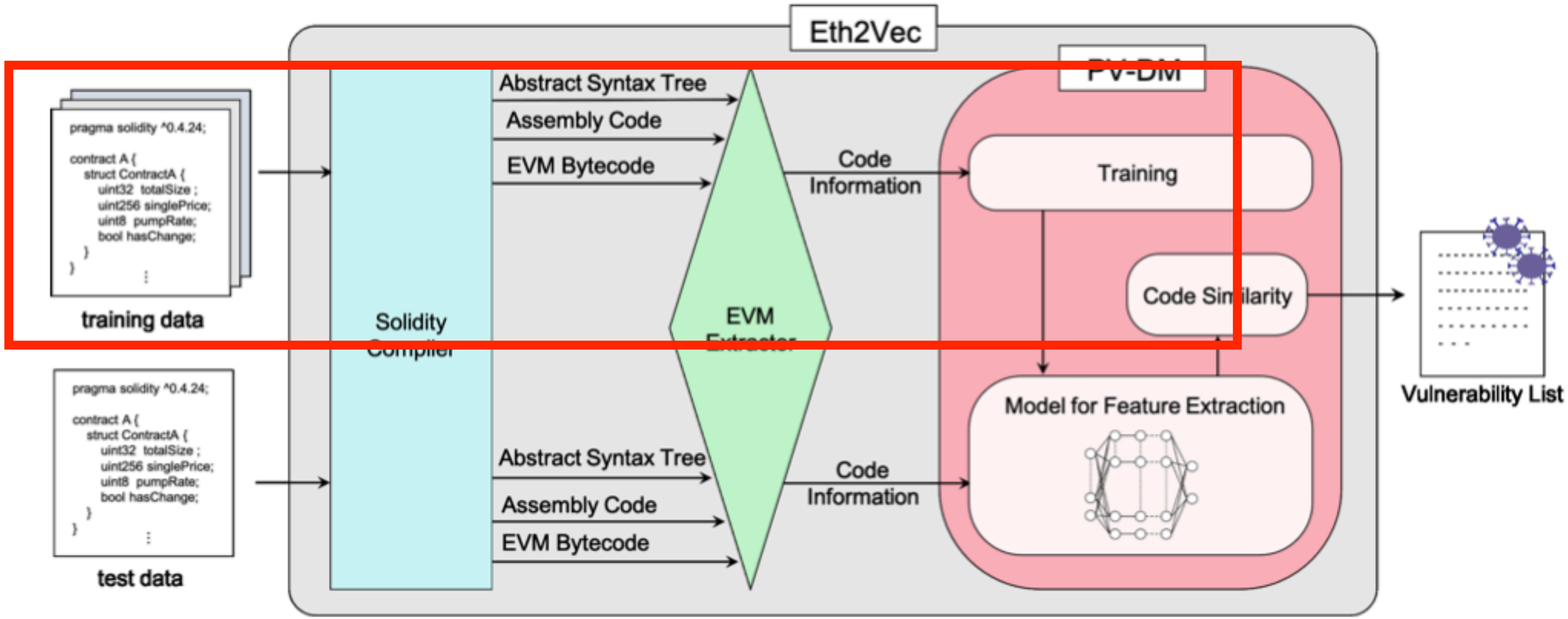
ETH2VEC

- 결론적으로 Eth2Vec은 EVM 바이트코드를 가지고 코드 클론과 취약점 리스트를 반환



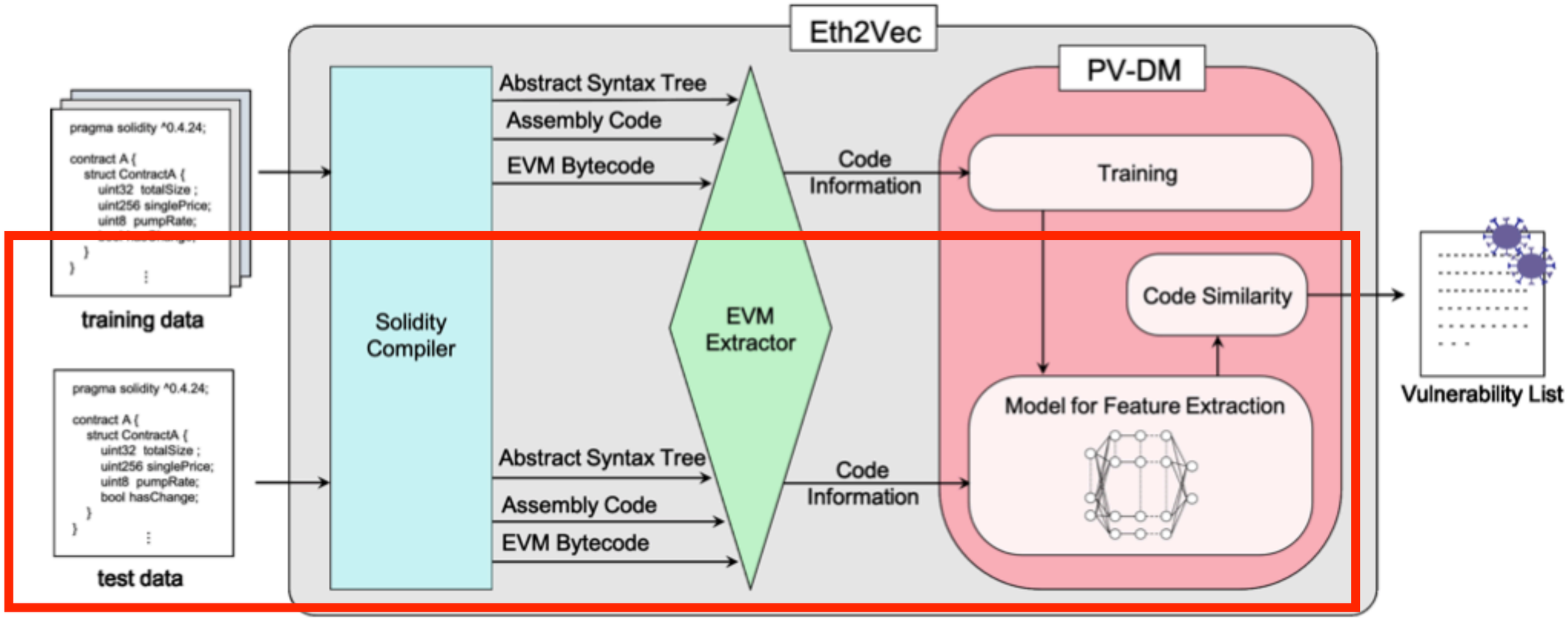
ETH2VEC

- 기존 도구들로부터 사전에 선별된 취약점을 가지고 학습



ETH2VEC

- 테스트 데이터를 가지고 (학습된) 취약한 컨트랙트와의 유사도를 평가



EXPERIMENTS

EXPERIMENTS

- 실험의 목적
- 클론 검출과 취약점 검출의 두 단계로 구성
  - 분석한 코드와 학습된 코드 간 유사도를 잘 나타내는가
  - 문맥상 클론(clone)의 코드 재작성도 잡아내는가
- 특징 추출을 수동으로 하는 기존의 연구와 비교
  - P. Momeni, Y. Wang, and R. Samavi.
    “Machine learning model for smart contracts security analysis.”
    In Proc. of PST 2019, pages 1-6. IEEE, 2019.

EXPERIMENTS: SETTING

- 데이터셋
  - Etherscan에서 5,000 개의 컨트랙트 파일을 수집
  - 솔리디티 버전 0.4.11으로 컴파일되는 컨트랙트만 사용
  - 총 95,152 개의 컨트랙트와 1,193,868개의 블록이 존재

EXPERIMENTS: SETTING

- 기준점
  - Eth2Vec과 SVM 기반 Momeni의 방법과 비교
- Momeni의 방법은
  - 이더리움 스마트 컨트랙트의 AST로부터 16개의 특징을 추출
  - 본 연구에서는 이들 중 16진법 주소를 배제하고 활용함
    - 주소는 배포 없이 소스 코드로부터 얻을 수 없기 때문
    - 본 실험에 사용된 특징 목록은 다음과 같음

EXPERIMENTS: SETTING

- Momeni의 방법에서 차용한 목록
  - 코드 라인 수
  - 컨트랙트 정의
  - 함수 정의
  - 바이너리 연산
  - 함수 호출
  - 블록
  - 수식(expression) 구문
  - 이벤트 정의
  - 바이트
  - 기본 타입
  - 모디파이어(modifier) 정의
  - 플레이스홀더(placeholder) 구문
  - 모디파이어 호출
  - Approve 함수 정의
  - 상수 값
  - ~~16진법 주소~~

EXPERIMENTS: SETTING

- 클론 검출
  - Eth2Vec이 테스트 컨트랙트들과 학습 컨트랙트 간 유사도를 검출할 수 있는가
- 10-Fold 교차 검증 (Cross Validation)
  - 500개의 테스트 컨트랙트가 랜덤으로 선출 —> 성능 평가에 사용
  - 나머지 4,500개의 컨트랙트가 학습 컨트랙트로 사용됨
  - 본 과정을 10번 반복

EXPERIMENTS: SETTING

- 취약점 검출
  - 테스트 컨트랙트가 진짜 취약점을 찾았는지 여부를 확인
  - Oyente와 SmartCheck를 이용해 ground truth를 설정

EXPERIMENTS: RESULTS ON CLONE DETECTION

- 클론 검출의 결과
  - 10번의 10-fold 교차검증의 결과에 대한
  - 평균과 표준편차
- ‘SVM w/o few clones’는 ground-truth에 따라 일부 클론을 제거한 후의 결과

	Eth2Vec	SVM [29]	SVM w/o few clones
Average	74.9%	34.6%	42.7%
Standard Deviation	0.9	34.6	43.6

EXPERIMENTS: RESULTS ON CLONE DETECTION

- Eth2Vec이 SVM-기반 모델 대비 더 많은 특징을 내재적으로 추출
- 표준편차가 현저히 작음
  - 즉, Eth2Vec의 특징 추출이 더 안정적이고
  - 다양한 코드의 특징 표현이 가능하다는 뜻

	Eth2Vec	SVM [29]	SVM w/o few clones
Average	74.9%	34.6%	42.7%
Standard Deviation	0.9	34.6	43.6

EXPERIMENTS: RESULTS ON CLONE DETECTION

- SVM-기반 방법의 정확도가 낮은 이유
  - 테스트 데이터셋이 적은 (함수) 클론을 가지기 때문
  - 이러한 상황에서 음수로 추론하게 됨
- 그러므로 F1-점수가 0이 되어 낮은 정확도와 큰 표준편차를 야기

	Eth2Vec	SVM [29]	SVM w/o few clones
Average	74.9%	34.6%	42.7%
Standard Deviation	0.9	34.6	43.6

EXPERIMENTS: RESULTS ON CLONE DETECTION

- 문맥 클론의 상황
- Concrete example
  - 원본 Transfer 함수

```
1  function _transfer(address _from, address _to, uint
       _value) internal {
2      require(_to != 0x0);
3      require(balanceOf[_from] >= _value);
4      require(balanceOf[_to] + _value > balanceOf[_to
           ]);
5      uint previousBalances = balanceOf[_from] +
           balanceOf[_to];
6      balanceOf[_from] -= _value;
7      balanceOf[_to] += _value;
8      Transfer(_from, _to, _value);
9      assert(balanceOf[_from] + balanceOf[_to] ==
           previousBalances);
10 }
```

EXPERIMENTS: RESULTS ON CLONE DETECTION

- 문맥 클론의 상황
- Concrete example
  - Eth2Vec이 찾은 클론
- 이전 코드에서
  - Line 5와 9만 뺀 것

```
1  function _transfer(address _from, address _to, uint
       _value) internal {
2      require(_to != 0x0);
3      require(balanceOf[_from] >= _value);
4      require(balanceOf[_to] + _value > balanceOf[_to
           ]);
5      balanceOf[_from] -= _value;
6      balanceOf[_to] += _value;
7      Transfer(_from, _to, _value);
8  }
```

EXPERIMENTS: RESULTS ON VULNERABILITY DETECTION

- 취약점 검출의 결과
  - Eth2Vec의 표준편차가 SVM-기반의 방법보다 낮음
  - 보다 안정적으로 취약점을 검출한다는 뜻

Vulnerability	Severity	Eth2Vec			SVM [29]		
		Precision [%]	Recall [%]	F1-Score [%]	Precision [%]	Recall [%]	F1-Score [%]
Reentrancy	3	86.6	54.8	61.5	30.0	7.8	12.3
Time Dependency	2	75.2	17.0	27.3	55.0	2.8	5.3
ERC-20 Transfer	1	95.6	58.4	72.4	89.0	95.3	92.0
Gas Consumption	1	48.0	29.0	32.4	10.0	3.1	4.7
Implicit Visibility	1	68.9	82.0	74.8	71.5	77.5	73.8
Integer Overflow	1	89.9	57.6	70.1	84.9	73.1	78.3
Integer Underflow	1	74.6	56.0	63.7	75.1	39.2	50.0
Average		77.0	50.7	57.5	59.3	42.7	45.2
Standard Deviation		14.7	19.7	18.0	27.3	36.4	34.7

EXPERIMENTS: RESULTS ON VULNERABILITY DETECTION

- 데이터의 분포와 무관하게 강인함
  - 재진입(reentrancy)이나 시간 의존성(time dependency) 취약점을 가진 컨트랙트가 적음
  - 즉, 학습을 위한 이러한 레이블이 많지 않다는 뜻

Vulnerability	Severity	Eth2Vec			SVM [29]		
		Precision [%]	Recall [%]	F1-Score [%]	Precision [%]	Recall [%]	F1-Score [%]
Reentrancy	3	86.6	54.8	61.5	30.0	7.8	12.3
Time Dependency	2	75.2	17.0	27.3	55.0	2.8	5.3
ERC-20 Transfer	1	95.6	58.4	72.4	89.0	95.3	92.0
Gas Consumption	1	48.0	29.0	32.4	10.0	3.1	4.7
Implicit Visibility	1	68.9	82.0	74.8	71.5	77.5	73.8
Integer Overflow	1	89.9	57.6	70.1	84.9	73.1	78.3
Integer Underflow	1	74.6	56.0	63.7	75.1	39.2	50.0
Average		77.0	50.7	57.5	59.3	42.7	45.2
Standard Deviation		14.7	19.7	18.0	27.3	36.4	34.7

EXPERIMENTS: RESULTS ON VULNERABILITY DETECTION

- 취약점의 상황
- Concrete example
  - 정수 오버플로우가 가능

```
1  function _transfer(address _from, address _to, uint
       _value) internal {
2      require(_to != 0x0);
3      require(balanceOf[_from] >= _value);
4      require(balanceOf[_to] + _value >= balanceOf[_to
           ]);
5      uint previousBalances = balanceOf[_from] +
           balanceOf[_to];
6      balanceOf[_from] -= _value;
7      balanceOf[_to] += _value;
8      emit Transfer(_from, _to, _value);
9      assert(balanceOf[_from] + balanceOf[_to] ==
           previousBalances);
10 }
```

EXPERIMENTS

- 비록 (1, 2)가 (1, 3)보다 유사해 보이지만
  - 정수 오버플로우가 존재함으로 3은 1의 클론이 아님
- Eth2Vec은 코드 유사도를
  - (1, 3)보다 (1, 2)을 더 유사한 것으로 판단함
- 이는 강인한 취약점 검출의 요소가 됨

```
1  function _transfer(address _from, address _to, uint
       _value) internal {
2      require(_to != 0x0);
3      require(balanceOf[_from] >= _value);
4      require(balanceOf[_to] + _value > balanceOf[_to
           ]);
5      uint previousBalances = balanceOf[_from] +
           balanceOf[_to];
6      balanceOf[_from] -= _value;
7      balanceOf[_to] += _value;
8      Transfer(_from, _to, _value);
9      assert(balanceOf[_from] + balanceOf[_to] ==
           previousBalances);
10 }
```

```
1  function _transfer(address _from, address _to, uint
       _value) internal {
2      require(_to != 0x0);
3      require(balanceOf[_from] >= _value);
4      require(balanceOf[_to] + _value > balanceOf[_to
           ]);
5      balanceOf[_from] -= _value;
6      balanceOf[_to] += _value;
7      Transfer(_from, _to, _value);
8  }
```

```
1  function _transfer(address _from, address _to, uint
       _value) internal {
2      require(_to != 0x0);
3      require(balanceOf[_from] >= _value);
4      require(balanceOf[_to] + _value >= balanceOf[_to
           ]);
5      uint previousBalances = balanceOf[_from] +
           balanceOf[_to];
6      balanceOf[_from] -= _value;
7      balanceOf[_to] += _value;
8      emit Transfer(_from, _to, _value);
9      assert(balanceOf[_from] + balanceOf[_to] ==
           previousBalances);
10 }
```

CONCLUSION

CONCLUSION

- Eth2Vec
  - 이더리움 스마트 컨트랙트의
  - 취약점을 검출하는
  - 머신러닝 기반 정적 분석 도구
- 각 스마트 컨트랙트의 특징을 자동으로 추출함이 특징

CONCLUSION

- 코드가 재작성되더라도
  - 77%의 정확도로 취약점 검출이 가능함
- 가장 중요한 취약점인 재진입(reentrancy) 역시도
  - 86.6%의 정확도로 검출됨
- Eth2Vec이 SVM-기반 방법의 성능을 압도함
  - 정확도, Recall, F1-score 측면에서

ETH2VEC

LEARNING CONTRACT-WIDE CODE REPRESENTATIONS FOR VULNERABILITY DETECTION ON ETHEREUM SMART CONTRACTS