논문 2020-2-5 http://dx.doi.org/10.29056/jsav.2020.12.05

code2vec 모델을 활용한 소스 코드 보안 취약점 탐지

양준혁*, 모지환**, 홍성문*, 도경구**†

Detection of Source Code Security Vulnerabilities Using code2vec Model

Joon Hyuk Yang*, Ji Hwan Mo**, Sung Moon Hong*, Kyung-Goo Doh**†

요 약

소스 코드의 보안 취약점을 탐지하는 전통적인 방법은 많은 시간과 노력을 필요로 한다. 만약 보안 취약점 유형들에 대한 좋은 품질의 데이터가 있다면, 이와 머신러닝 기술을 활용해 효과적으로 문제를 해결할 수 있을 것이다. 이에 본 논문은 정적 프로그램 분석에 머신러닝 기술을 활용하여 소스 코드에서 보안 취약점을 탐지하는 방법을 제시하고, 실험을 통하여 가능성을 보인다. 메소드 단위의 코드 조각의 의미를 해석하여 메소드의 이름을 예측하는 code2vec 모델을 사용하고, 모델을 생성하고 검증 및 평가를 하기 위한 데이터로 흔히 발생할 수 있는 보안 취약점을 모아놓은 Juliet Test Suite를 사용하였다. 모델 평가 결과 약 97.3%의 정밀도와 약 98.6%의 재현율로 매우 희망적인 결과를 확인하였고 오픈 소스 프로젝트의 취약점을 탐지함으로써 가능성을 보였다. 향후 연구를 통해 다른 취약점 유형과 다양한 언어로 작성된 소스 코드에 대해서 대응함으로써 기존의 분석 도구들을 대체할 수 있을 것이다.

Abstract

Traditional methods of detecting security vulnerabilities in source-code require a lot of time and effort. If there is good data, the issue could be solved by using the data with machine learning. Thus, this paper proposes a source-code vulnerability detection method based on machine learning. Our method employs the code2vec model that has been used to propose the names of methods, and uses as a data set, Juliet Test Suite that is a collection of common security vulnerabilities. The evaluation shows that our method has high precision of 97.3% and recall rates of 98.6%. And the result of detecting vulnerabilities in open source project shows hopeful potential. In addition, it is expected that further progress can be made through studies covering with vulnerabilities and languages not addressed here.

한글키워드 : 보안 취약점, SQL 삽입, 보안 취약점 탐지 도구, 머신 러닝, 정적 프로그램 분석

keywords : security vulnerability, SQL injection, security vulnerability detection tool, machine learning, static program analysis

1. 서 론

국제 인터넷 표준화 기구(IETF) RFC 2828[1]은 보안 취약점을 시스템의 디자인 또는 구현, 운

* 한양대학교 컴퓨터공학과
** 한양대학교 소프트웨어학부
† 교신저자: 도경구(email: doh@hanyang.ac.kr)
접수일자: 2020.11.09. 심사완료: 2020.12.14.
게재확정: 2020.12.21.

영관리에서의 결함이나 약점으로, 시스템의 보안 정책을 침해하기 위해 공격당할 수 있는 것으로 정의하고 있다. 보안 취약점은 널리 알려진 SQL 삽입, XSS (Cross Site Scripting), 운영체제 명령 삽입 등이 있으며, 이러한 보안 취약점을 방치할 경우 프로그램이 의도한대로 작동하지 않거나 중요한 내부정보들이 외부로 유출될 수 있다.

보안 취약점을 탐지하는 방법은 크게 프로그램을 실행하면서 분석하는 동적 프로그램 분석과 프로그램을 실행하지 않고 분석하는 정적 프로그램 분석 두 가지로 나뉘어 다양한 방향으로 연구가 진행되고 있다. 기존의 정적 프로그램 분석은 소스 코드의 전체적인 흐름을 이해하고 그 속에서 보안 취약점을 탐지한다. 본 논문에서는 기존의 정적 프로그램 분석 방법이 아닌 머신러닝을 활용한 새로운 보안 취약점 탐지 방법을 제시하고 머신러닝 기반 정적 프로그램 분석 모델을 구현 및 평가하였다. 또한 해당 모델을 사용해 소스 코드 보안 취약점을 탐지함으로써 적은 시간 예산으로 만족할 만한 수준의 정밀도를 갖는 결과를 얻었다.

2. 정적 프로그램 분석

정적 프로그램 분석[2,3]은 프로그램을 실행하지 않고 소스 코드의 전반적인 이해를 기반으로 찾고자 하는 어떠한 성질, 예를 들어, 시스템 콜에 전달되는 인자 값이 외부에서 입력된 위험한 값인지의 여부 등의 성질을 분석한다. 하지만 일반적으로 프로그램에서 어떠한 성질을 완벽(sound and complete)하게 찾아내는 것은 불가능하기 때문에 정적 프로그램 분석은 프로그램에서 실행 가능한 경로와 변수가 가질 수 있는 범위 등을 추상화하여 분석하고 이를 통해 옳지만(sound) 부정확(incomplete)한 답을 찾는다. 이러한 특성 때문에 정적 프로그램 분석은 오탐(false positive)이 발생할 가능성이 있다.

오탐이 많으면 이를 다시 확인해야 하는 개발자의 부담이 커진다. 오탐을 줄이기 위해 가능한 모든 실행 경로와 변수가 가질 수 있는 모든 값의 범위에 대해 오류가 발생할 가능성이 있는지 검사하도록 할 수 있지만 많은 시간이 걸린다. 따라서 정적 프로그램 분석은 오탐을 줄여 정확도를 높이기 위해서는 많은 시간 예산이 드는 반면, 시간 예산을 줄이면 오탐이 늘어 정확도가 줄어들어, 서로 반비례 관계에 있다.

3. 머신러닝 기반 정적 프로그램 분석 모델

어떤 문제를 해결할 때 좋은 품질의 데이터로 생성된 머신러닝 모델을 활용하면 만족할만한 수준의 해를 빠르게 구할 수 있다. 정적 프로그램 분석을 활용해 보안 취약점을 탐지하는 Clang analyzer[4], Coverity[5], Sparrow[6] 등 다양한 도구들이 개발되어 널리 사용되고 있는데, 분석 도구의 성능을 테스트하기 위해 만들어진 데이터 세트의 규모가 크기 때문에 머신러닝의 학습 데이터로 활용해볼 만하다.

머신러닝을 활용하여 보안 취약점을 탐지하는 모델을 생성하기 위해 그림 1과 같이 분석 도구의 성능을 테스트하기 위한 데이터 세트 중 위험한(bad) 소스 코드와 안전한(good) 소스 코드로 구별 가능한 데이터 셋을 학습 데이터로 사용하였고, 소스 코드를 학습시키기 위해 소스 코드를 벡터로 변환하고, 벡터로 표현된 소스 코드를 학습하는 머신러닝을 모델을 활용해 러닝 모델을 생성하였다. 러닝 모델은 위험한 소스 코드와 안전한 소스 코드의 패턴을 학습해 분석하고자 하는 소스 코드가 입력으로 주어지면 보안 취약점 패턴의 포함 여부에 따라 bad와 good을 출력한다.

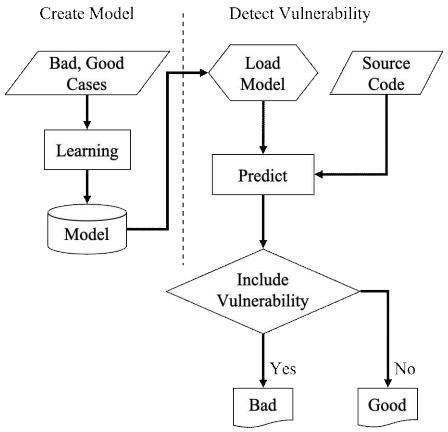


그림 1. 머신러닝 기반 보안 취약점 탐지 전략
Fig 1. Strategy to Detect Vulnerabilities

3.1 학습 데이터

Juliet Test Suite[7]는 프로그램 정적 분석 도구의 성능을 평가하기 위해 NSA(National Security Agency)의 CAS(Center for Assured Software)가 개발했다. 112개의 CWE(Common Weakness Enumeration)[8]를 포함하고 있으며, 28,881개의 많은 수의 개별 케이스가 존재한다. 그림 2에서 볼 수 있듯 각 취약점 유형마다 위험한 소스 코드와 안전한 소스 코드로 나누어 메소드 이름에 알아보기 쉽게 표기되어있어 학습데이터로 사용하기 적합하다.

본 논문에서는 CWE 예제 중 CWE89_SQL_Injection을 학습 데이터로 사용했다. 그림 2의 소스 코드를 보면 bad 메소드에서 외부 입력을 통해 들어온 data를 badSink 메소드를 호출하며 파라미터로 넘겨주고 있다, data를 넘겨받은 badSink 메소드는 별다른 처리없이 그대로 excuteQuery 메소드를 호출하며 파라미터로 전달하고 있다. 이는 위험한 코드에 해당한다. 반면에 goodB2G1 메소드는 bad와 같이 외부 입력을 통해 들어온 data가 goodB2G1Sink로 넘어가지만 일반적으로 안전한 방법인 prepareStatement 객체와 setString 메소드를 통해 execute 되므로

```
32  public void bad() throws Throwable
33  {
34    String data;
…   …
38    data = System.getenv("ADD");
39
40    badPrivate = true;
41    badSink(data );
42  }
```

```
44  private void badSink(String data ) throws Throwable
45  {
…     …
55    Boolean result = sqlStatement.execute(
        "insert into users (status) values ('updated')
         where name='"+data+"'");
…     …
96  }
```

```
111 private void goodB2G1() throws Throwable
112 {
113   String data;
…   …
117   data = System.getenv("ADD");
118
119   goodB2G1Private = false;
120   goodB2G1Sink(data );
121 }
```

```
123 private void goodB2G1Sink(String data ) throws Throwable
124 {
…     …
140   sqlStatement = dbConnection.prepareStatement(
        "insert into users (status) values ('updated')
         where name=?");
141   sqlStatement.setString(1, data);
142
143   Boolean result = sqlStatement.execute();
…     …
186 }
```

그림 2. CWE89_SQL_Injection__Environment_execute_21 소스 코드의 일부
Fig 2. Part of The Source Code CWE89_SQL_Injection__Environment_execute_21

안전한 코드에 해당한다.

3.2 머신러닝 모델

본 논문에서는 code2vec[9] 모델을 사용하였다. code2vec은 주어진 코드 블록이 의미하는 작업을 이해하고 이름을 제안하려는 목적을 갖고 만들어진 머신러닝 모델이다. code2vec은 벡터로 표현된 코드 블록을 학습하고 프로그램의 실행과정에 대해 신경망 기술을 적용한다. 생성된 code2vec 모델은 메소드 단위의 코드 조각이 주어질 때 신경망 연산을 통해 코드 조각이 의미하는 작업에 대한 가장 확률이 높은 메소드 이름을 제안한다.

3.3 학습 데이터 전처리

정적 프로그램 분석을 통해 보안 취약점을 탐지하기 위해서는 메소드 호출 관계 등 프로그램의 동작에 대한 전반적인 이해가 필요하다. Juliet Test Suite의 SQL 삽입 보안 취약점 예제는 그림 2에서 볼 수 있듯 소스(source)에 해당하는 외부 입력을 통해서 들어오는 data와 싱크(sink)에 해당하는 execute 메소드 호출이 여러 메소드에 걸쳐 나타난다. 하지만 code2vec의 모델은 기본적으로 하나의 코드 조각 즉 하나의 기능(메소드) 단위로 학습을 진행한다. 하나의 메소드 안에서 메소드 호출 관계를 따라갈 수 있도록 리펙토링(refactoring) 기법의 하나인 메소드 인라인(method inline)을 활용해 학습 데이터를 전처리한다.

CWE89_SQL_Injection의 소스 코드에서 나타나는 메소드 호출 유형은 크게 프로시저 호출과 함수 호출 두 가지로 나뉜다. 메소드 호출 유형별로 구분하여 메소드 인라인을 수행하는 알고리즘은 그림 3과 같다. 프로젝트 단위의 소스 코드를 입력으로 받아 그 안에 존재하는 자바 라이브러리 호출을 제외한 모든 메소드 호출 지점을 찾고, 어떤 메소드 안에서 다른 메소드를 호출하는(caller) 지점을 호출당하는(callee)의 메소드의 몸통(body)으로 대체한다.

```
I  for caller each methods in Project
A    invocations = find all invocations in caller except Libraries
B    for callee each invocations
1      cbs = createBlockStmt( )
2      ret = createLocalVariable(callee.getType, "ret", null)
3      cbs.insertEnd(ret)
4      for arg, param each arguments, parameters
a        clv = createLocalVariable(param.getType,
                                   param.getSimpleName,
                                   arg.clone)
b        cbs.insertEnd(clv)
5      i_body = createBlockStmt( )
6      i_body.setLabel(callee.getSimpleName)
7      i_body.setBody(callee.body)
8      for ret_stmt each return statement in i_body
a        cas = createAssignStmt(ret, ret_stmt.getExpression)
b        ret_stmt.replace(cas)
c        bp = createBreakStmt(m.getSimpleName)
d        ret_stmt.insertEnd(bp)
9      cbs.insertEnd(i_body)
10     callee.insertEnd(cbs)
11     callee.replace(ret)
```

그림 3. 메소드 인라인 알고리즘
Fig 3. Method Inline Algorithm

```
1   public void bad() throws Throwable {
2     String data;
…   …
5     data = System.getenv("ADD");
6     badPrivate = true;
7     {
8       String data = data;
9       badSink : {
…         …
18          Boolean result = sqlStatement.execute(
              "insert into users (status)
               values ('updated')
               where name='"+data+"'");
…         …
42      }
…   …
```

그림 4. 인라인이 적용된 소스 코드의 일부
Fig 4. Part of The Source Code After Inline
(CWE89_SQL_Injection__Environment_execute_21)

그림 4는 그림 2의 bad() 메소드가 자동화 코드변환 모듈을 통과한 후의 모습이다. 그림 2의 badSink 메소드를 호출하는 부분이 badSink 메소드의 몸통으로 대체된 것을 확인할 수 있다.

학습 데이터 전처리 과정을 통해 code2vec은 5번 라인에서 호출하는 getenv()를 통해 들어온 외부입력이 18번 라인에서 호출하는 execute()로 전달되는 흐름을 학습할 수 있다.

3.4 분석 모델

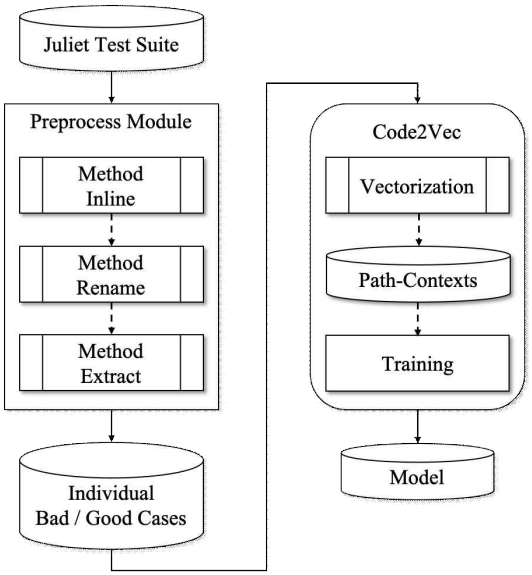


그림 5. 모델 생성 개요
Fig 5. Abstraction of Creating Model

code2vec의 TensorFlow 모델에 Juliet Test Suite for Java의 CWE89_SQL_Injection 유형에 대한 학습을 그림 5와 같이 진행했다.

학습 데이터 전처리 모듈(Preprocess Module)은 Juliet Test Suite for Java 소스 코드를 받아 Inline, Rename, Extract의 과정을 거쳐 위험한(Bad) 경우와 안전한(Good) 경우를 메소드 단위의 개별적인 코드 조각으로 변환한다. 이렇게 변환한 소스 코드 조각을 code2vec에 적용해 모델을 생성한다.

본 논문에서는 Juliet Test Suite for Java의 CWE89-SQL_Injection 예제 8,150개를 학습 데이터로 삼았고, 이를 랜덤하게 8:1:1 비율의 Training Set : Validation Set : Testing Set으로 나눠 학습을 진행했다.

4. 모델 평가

4.1 테스트 데이터

		Predicted	
		bad	good
Original	bad	213	3
	good	6	593

그림 6. 모델 평가 결과
Fig 6. Evaluation Result

그림 6은 3.4절에서 생성한 모델을 815개의 테스트 데이터로 평가한 결과이다. 전체 테스트 케이스 815개 중, 213개의 위험한 코드와 593개의 안전한 코드를 정확히 판별해냈다. 실제로 위험한 코드이지만, 안전한 코드로 예측한 미탐(false negative)이 3개, 실제로 안전한 코드이지만, 위험한 코드로 예측한 오탐(false positive)이 6개밖에 없었다. 이 결과를 가지고 정밀도(precision)와 재현율(recall)을 계산하면, 각각 0.973, 0.986 이 된다. 이는 위험한 코드를 예측할 때 약 97.3%의 정확성을 갖는다는 뜻이고, 테스트 케이스의 전체 위험한 코드 중 약 98.6%를 식별할 수 있다는 것을 뜻한다.

4.2 오픈 소스 프로젝트

학습 데이터로 사용한 Juliet Test Suite로 평가했을 때 매우 희망적인 결과를 얻었다. 하지만 해당 결과는 과적합의 가능성이 있다. 따라서 모델의 일반화를 검증하기 위해 기존의 테스트 케이스에 포함되지 않은 오픈 소스 프로젝트 SQL-Injection-Simulation-Project[10] 소스 코드의 보안 취약점을 탐지해 보았다.

```
32  public JsonObject getInfo(String userId,
                              String userPass) {
    …
37    sql = "select * from users
              where user_id = '"+ userId +"'
              and user_pass = '"+userPass+"';";

39    ResultSet rs = stmt.executeQuery(sql);
    …
70  }
71
72  public JsonObject getInfoProtection(String userId,
                                        String userPass) {
    …
78    sql = "select * from users
              where user_id = ? and user_pass = ?;";
79    pstmt = conn.prepareStatement(sql);
80    pstmt.setString(1, userId.trim());
81    pstmt.setString(2, userPass.trim());
82    ResultSet rs = pstmt.executeQuery();
    …
112 }
```

그림 7. 모의 프로젝트 소스 코드 일부
Fig 7. Part of The Source Code in Simulation Project

그림 7에서 확인 할 수 있듯 SQL - Injection - Simulation-Project는 SQL 삽입 공격을 실험할 수 있도록 두 가지 메소드를 제공하고 있다. GetInfo 메소드는 파라미터를 통해 전달된 외부 입력이 그대로 executeQuery 메소드로 들어가기 때문에 위험한 코드라 할 수 있고, GetInfoProtection 메소드는 일반적으로 안전하다고 하는 prepareStatement 클래스와 set String 메소드를 사용하여 executeQuery 메소드를 호출하는 안전한 코드라고 할 수 있다.

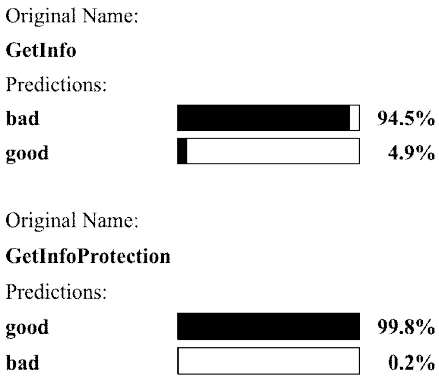


그림 8. 모의 프로젝트 예측 결과
Fig 8. Result of Predicting Simulation Project

앞선 평가에서 사용된 러닝 모델로 GetInfo 메소드와 GetInfoProtection 메소드를 각각 예측한 결과 그림 8과 같이 위험한 코드와 안전한 코드를 매우 높은 확률로 예측하는 것을 확인하였다.

5. 결론 및 향후 연구

Juliet Test Suite와 같이 위험한(bad), 안전한(good) 소스 코드를 분류할 수 있는 데이터 집합이 존재한다면 머신러닝을 활용하여 취약점을 탐지하는 것이 가능하다는 것을 보았다. 머신러닝 모델은 프로젝트 단위의 소스 코드를 일괄적으로 분석하여 취약점을 찾아냄으로써 취약한 정도 확률로 예측해준다.

오탐이 얼마나 적은가는 어떠한 정적 프로그램 분석기가 실무에 효과적으로 사용 가능할지를 판단할 수 있는 중요한 지표다. 4.1절의 평가 결과에서 볼 수 있듯, 머신러닝 기반 정적 프로그램 분석 모델의 오탐은 매우 적게 나타났다. 또한 4.2절에서 학습에 사용된 Juliet Test Suite의 데이터가 아닌 실제로 동작하는 모의 프로젝트의 취약점을 탐지함으로써 실제 프로젝트에도 적용할 수 있다는 가능성을 보았다.

본 논문에서는 CWE 중에서도 SQL 삽입 취약점 유형에 대해서만 학습을 진행했다. Juliet Test Suite의 다른 CWE 유형에 대해서도 학습 데이터의 레이블에 레퍼런스를 추가하여 학습을 진행하면 취약점 탐지 및 분류도 가능할 것이다.

code2vec의 머신러닝 모델은 Java 언어 외의 다른 언어에도 유연하게 대응하는 모델이므로 각 언어에 맞는 코드 변환 모듈을 구현하여 적용하면 여러 언어로 작성한 소스 코드에 대해서도 보안 취약점 탐지가 가능할 것이다.

Acknowledgement

"본 연구는 과학기술정통신부 및 정보통신기획평가원의 SW중심대학지원사업의 연구결과로 수행되었음"(2018-0-00192)

참 고 문 헌

[1] Internet Engineering Task Force RFC 2828 Internet Security Glossary, 2020.10.22. https://tools.ietf.org/html/rfc2828#page-190

[2] P. Emanuelsson, U. Nilsson, "A comparative study of industrial static analysis tools", Proc. of the 3rd International Workshop on Systems Software Verification (SSV 2008), 2008.

[3] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, D. Engler, "A few billion lines of code later: Using static analysis to find bugs in the real world", CACM, vol.53, no.2, pp.66-75, 2010. https://doi.org/10.1145/1646353.1646374

[4] Clang Analyzer, 2020.11.05. https://clang-analyzer.llvm.org

[5] Coverity, 2020.11.05. https://scan.coverity.com

[6] Sparrow, 2020.11.05. http://wisestone.kr/solution/sparrow.php

[7] Juliet Test Suite for Java, 2020.08.10. https://samate.nist.gov/SARD/around.php#juliet_documents

[8] CWE, 2020.08.10. https://cwe.mitre.org

[9] URI ALON, MEITAL ZILBERSTEIN, OMER LEVY, ERAN YAHAV, code2vec: Learning Distributed Representations of Code, ACM Program. Lang. 3, POPL, 40(29 pages), 2019. https://doi.org/10.1145/3290353

[10] SQL-Injection-Simulation-Project, 2020.10.29. https://github.com/mndarren/SQL-Injection-Simulation-Project

저 자 소 개

양준혁(Joon Hyuk Yang)

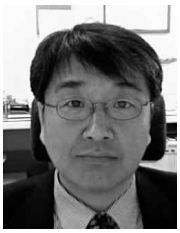
2020 한양대학교 ERICA
소프트웨어학부 학사
2020-현재 : 한양대학교 대학원 컴퓨터공학과 석사과정
<주관심분야> 프로그램 분석, 소프트웨어 보안

모지환(Ji Hwan Mo)

2017-현재 : 한양대학교 ERICA
소프트웨어학부 학사과정
<주관심분야> 프로그래밍 언어, 프로그램 분석

홍성문(Sung Moon Hong)

2012 위덕대학교 컴퓨터공학과 학사
2014 한양대학교 컴퓨터공학과 석사
2014-현재 : 한양대학교 대학원 컴퓨터공학과 박사과정
<주관심분야> 프로그래밍언어, 프로그램 분석, 소프트웨어 보안



도경구(Kyung-goo Doh)

1980 한양대학교 산업공학과 학사
1987 아이오와주립대학 컴퓨터과학 석사
1992 캔자스주립대학 컴퓨터과학 박사
1993-1995 일본 아이주 대학 교수
1995-현재 : 한양대학교 ERICA
소프트웨어학부 교수
<주관심분야> 프로그래밍언어, 프로그램 분석, 소프트웨어 보안, 소프트웨어 공학