Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.30, NO.6, Dec. 2020 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2020.30.6.1131

기계학습을 이용한 소스코드 정적 분석 개선에 관한 연구

박 양 환,[1†] 최 진 영[2‡]
[1,2]고려대학교 정보보호대학원 (대학원생, 교수)

A Study on the Improvement of Source Code Static Analysis Using Machine Learning

Yang-Hwan Park,[1†] Jin-Young Choi[2‡]
[1,2]Graduate School of Information Security, Korea University
(Graduate student, Professor)

요 약

소스코드에 대한 정적 분석은 광범위한 소스코드에 대해서 잔존하는 보안약점을 찾는 것으로 정적 분석 도구를 활용하여 점검을 하고, 그 결과에 대해서 정적 분석 전문가가 정탐 및 오탐 분석을 한다. 이 과정에서 분석양이 많고 오탐의 비율이 높아 많은 시간과 노력이 들어가게 되어 효율적으로 분석하는 방안이 요구되고 있다. 또한 전문가들은 정·오탐 분석을 할 때 결함이 발생한 라인의 소스코드만 보고 분석을 하는 경우는 드물다. 결함의 유형에 따라서 주변의 소스코드를 같이 분석하고 최종 분석 결과를 내리게 된다. 이러한 정적 분석 도구를 사용하여 전문가가 정·오탐을 판별하는 어려움을 해결하기 위해서 본 논문에서는 정적 분석 도구가 찾은 보안약점이 정탐인지 아닌지를 전문가가 아닌 인공지능을 통해 판별하는 방법을 제안한다. 또한 이러한 기계학습에 사용되는 학습 데이터(결함주변 소스코드)의 크기가 성능에 어떤 영향을 미치는지 실험을 통해 최적의 크기를 확인하였다. 이 결과를 통해 정적 분석 후 정·오탐을 분류하는 정적 분석 전문가의 업무에 도움을 줄 것으로 기대한다.

ABSTRACT

The static analysis of the source code is to find the remaining security weaknesses for a wide range of source codes. The static analysis tool is used to check the result, and the static analysis expert performs spying and false detection analysis on the result. In this process, the amount of analysis is large and the rate of false positives is high, so a lot of time and effort is required, and a method of efficient analysis is required. In addition, it is rare for experts to analyze only the source code of the line where the defect occurred when performing positive/false detection analysis. Depending on the type of defect, the surrounding source code is analyzed together and the final analysis result is delivered. In order to solve the difficulty of experts discriminating positive and false positives using these static analysis tools, this paper proposes a method of determining whether or not the security weakness found by the static analysis tools is a spy detection through artificial intelligence rather than an expert. In addition, the optimal size was confirmed through an experiment to see how the size of the training data (source code around the defects) used for such machine learning affects the performance. This result is expected to help the static analysis expert's job of classifying positive and false positives after static analysis.

Keywords: Static analysis, Secure Coding, Deep Learning, Convolutional Neural Networks(CNN)

Received(10. 05. 2020), Modified(10. 29. 2020), Accepted(10. 29. 2020)

† 주저자, redscv@gmail.com
‡ 교신저자, narnia@korea.ac.kr(Corresponding author)

I. 서 론

사이버 공격의 90%가 SW 보안취약점을 악용[1]하고 있어 SW에 결함이나 오류 등이 내포되면 심각한 보안 사고를 초래할 수 있다. 구글 플러스의 SW 보안취약점에 따른 서비스 중단[2], GM의 코딩 오류에 따른 대규모 차량 리콜 사태[3] 등이 최근 대표적인 사례이다.

SW 보안취약점을 발견하기 위해서는 프로그램 실행을 기반으로 한 동적 분석과 프로그램 비실행을 기반으로 소스코드를 대상으로 하는 정적 분석[4]이 사용된다. 이중 정적 분석은 소스코드의 크기가 방대하여 프로그램을 사용하여 분석을 하고 있다.

국내에서는 이러한 프로그램을 "소스코드 보안약점 분석 도구" 라는 제품 유형으로 분류하여 CC(Common Criteria)인증을 받고 있다. 또한, 행정·공공기관이 정보시스템 구축 후 진단 도구 사용 시, CC인증을 받은 도구를 사용('14년부터 적용)하도록 하고 있다.

Fig. 1은 정적 분석을 하는 일반적인 절차[5]를 나타낸다. 환경 분석 → 정적 분석 도구를 사용한 분석 → 정적 분석 도구 결과물에 대한 전문가 분석 → 보고서 작성으로 이루어진다. 이중 전문가 분석 단계에서 가장 많은 시간이 소요된다. 분석 결과가 많고 오탐(False Positive)이 많아 이것을 분석하는데 상당한 시간이 소요된다.

본 논문에서는 기계학습을 이용하여 전문가 분석 단계를 효율화하는 방안에 대해서 알아보고자 한다. 자주 발생되는 CWE(Common Weakness Enumeration)에 대하여 학습 데이터(전문가가 이전에 분석한 결과와 해당 소스코드)를 사용하여 학습을 시키고 결과를 분석한다.

또한, CWE별로 최적의 학습 데이터를 제시하기 위해 결함이 발생한 주변의 소스코드의 범위를 변경하면서 실험을 하고 결과를 제시한다.

본 논문의 구성은 다음과 같다. 2장에서는 이전의 유사 논문에 대해서 알아보고 3장에서는 제안하는 방법에 대해서 설명하고. 이를 토대로 4장에서는 실험을 통한 성능 평가를 한다. 5장에서는 결론과 향후 연구에 대해서 기술한다.

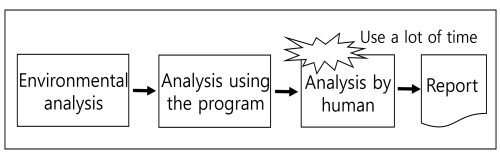


Fig. 1. Typical static analysis procedure

II. 관련연구

소스코드 정적 분석에 기계학습을 활용하는 연구가 최근 많이 시행되고 있다. 소스코드 정적 분석은 언어의 라이브러리에서 제공하는 변수명, 함수명, 클래스명 등의 구문들이 반복적으로 사용된다. 또한 개발자들이 생성하는 변수명, 함수명, 클래스명 또한 규칙이 유사하여 기계학습 분야에서 다양한 방법으로 연구가 진행되고 있다.

소프트웨어 공통 분야에서는 RNN(Recurrent Neural Network)을 사용하여 소스코드에서 유사하거나 일치하는 부분을 찾는 연구[6], LSTM(Long Short Term Memory)을 사용하여 소스코드를 탐지하기 위한 모델에 관한 연구[7], RNN을 사용한 소스코드 자동 생성[8] 등 소스코드 분석 중심의 연구가 수행되고 있다.

보안취약점 분야에서는 CNN(Convolutional Neural Network), RNN 등 알고리즘을 사용하여 버퍼 오버플로우 등 특정 취약점을 찾기 위한 연구[9]가 수행되고 있고, 최근에는 크로스사이트 스크립트 등 보안약점에서 활성 함수(Activation function)의 성능 분석[10]에 관한 연구가 수행되었다.

기존의 연구는 소스코드에서 특정 부분을 탐지하거나 특정 보안취약점을 분류하는 연구가 많이 수행되었다. 본 연구는 정적 분석 도구의 결과물을 효율적으로 분석하는 방법을 제시하는 것으로 차별화된 접근을 한다.

본 논문에서 정적 분석 도구의 결과물 중 하나인 정탐(True Positive)과 오탐(False Positive) 분석하는 문제에 대해서 기계학습을 적용하는 방법을 제안한다.

III. 개요 및 제안 방법

정적 분석 도구의 결과물에 대해서 정·오탐 분석을 할 때 전문가들은 결함라인과 그 주변 코드를 같이 보면서 정·오탐 판별을 위한 정보를 수집하고 분석한다.

결함라인과 아주 가까운 함수명과 입력 변수만을 분석할 수도 있고, 주변의 부분들을 분석할 수도 있다. 그리고 경우에 따라서는 선언 부분 등 멀리 떨어

진 부분까지 이동하면서 분석을 할 수도 있다.

또한, 전문가들은 예상되는 결함의 유형에 따라서 그 주변을 보는 범위를 다르게 하며 분석한다.

본 논문에서는 기계학습에 사용하는 학습 데이터(결함라인 주변의 소스코드의 범위)를 최적화하여 정·오탐을 판별 성능을 향상시켜 정적 분석 과정을 효율화하는 방법을 제시한다.

3.1 제안 방법

본 논문에서는 정적 분석 도구의 결과(결함라인이 포함된 소스코드)와 이 결과물을 전문가들이 실제로 분석한 결과(정·오탐 분석 결과)를 학습 데이터로 사용한다.

기계학습에서 예측한 정·오탐 결과와 실제 정·오탐 분석 결과를 비교하여 지도학습을 시킨다.

이 과정에서 결함라인을 포함한 소스코드의 범위를 변경하여 정·오탐 예측 정확도에 대한 성능을 평가한다.

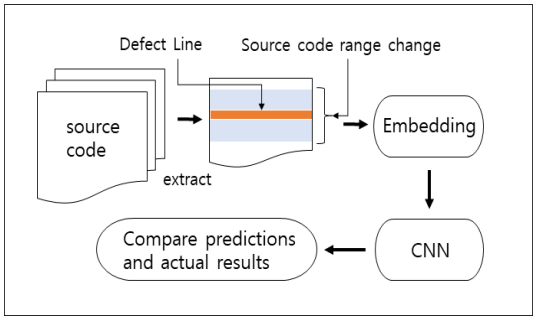


Fig. 2. Conceptual diagram of the proposed method

3.2 성능 평가 방법

실험 결과로 얻어진 정탐 및 오탐에 대한 분류 성능을 확인하기 위해 혼동 행렬(Confusion Matrix)[11]을 사용한다. 혼동 행렬은 분류 예측이 실제와 같은지 다른지를 표기하는 방법으로 얼마나 정확한 결과를 계산하는지를 객관적으로 측정할 수 있다. Table.1은 혼동 행렬의 일반적인 형태를 나타낸다.

혼동 행렬에서는 민감도(TP / [TP+FN]), 특이도(TN / [TN+FP]), 정밀도(TP / [TP+FP]), 정확도([TP+TN] / [TP +FP+FN+TN]) 등을 확인할 수 있다. 본 논문에서는 분류 성능 확인에 정확도를 사용한다.

Table 1. A typical layout of a confusion matrix

	Actual Positive	Actual Negative
Predicted Positive	True Positive(TP)	False Positive(FP)
Predicted Negative	False Negative(FN)	True Negative(TN)

IV. 실험 및 평가

4.1 데이터 세트

데이터 세트의 개수는 전체 109,885개로, 정적 분석 도구의 결과(결함라인이 포함된 소스코드)와 이 결과물을 전문가들이 실제로 분석한 결과(정·오탐 분석 결과)를 사용하였다. 세부적으로는 학습 세트 70,078개, 검증 세트 17,519개, 시험 세트 22,288개로 구성하였다.

또한, 정·오탐 분류 성능이 한쪽으로 편향되는 것을 예방하기 위해서 데이터 세트의 정탐과 오탐 비율을 1:1로 고르게 분포시켰다.

Table 2. Data set configuration

		Number of source codes and result	
Training Data Set	True Positive	35,039	70,078
	False Positive	35,039	
Validation Data Set	True Positive	8,760	17,519
	False Positive	8,759	
Test Data Set	True Positive	11,144	22,288
	False Positive	11,144	

4.2 CNN 모델 구성

CNN 모델은 소스코드에서 특징을 추출하는데 유

용하기 때문에 소스코드의 지역별 특징을 추출하는데 사용한다.

결함라인을 포함한 소스코드는 512개 차원의 임베딩 값으로 변환되어 CNN의 입력 값으로 사용된다.

4개의 합성곱 신경망 계층을 사용하고 중간에 패딩(Padding)을 사용하여 크기를 유지하고 풀링(MaxPooling)을 사용하여 특징을 강조하여 정확성을 높인다.

추출된 특징은 2개의 은닉층을 사용하고 과적합(Overfitting)을 예방하기 위해서 Dropout을 사용한다. 마지막에는 완전 연결(Fully Connected) 계층에서 정탐과 오탐 여부를 출력하게 된다.

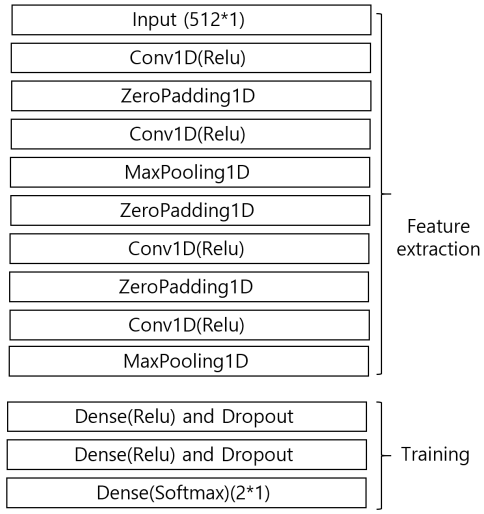


Fig. 3. CNN model

4.3 실험 결과

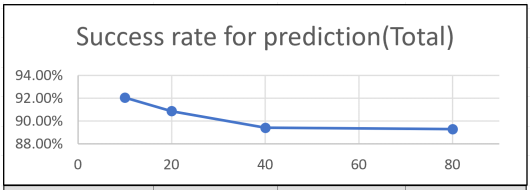
CWE 중에서 많이 발생하는 13개의 CWE에 대해서 실험을 하였고 CWE가 발생한 결함라인과 인접한 소스코드 범위를 변화시켜 이에 따른 정·오탐 분류에 대한 성능을 평가하였다.

인접 범위를 최소 5라인 ~ 최대 1,000라인까지 변화시킨 후 샘플링 하여 평가하고 최고의 성능과 최저의 성능을 보여주는 소스코드 범위를 찾았다.

이때, 기계학습의 결과와 전문가가 실제 분석한 정·오탐 분석 결과를 비교하여 혼동 행렬의 정확도를 기준으로 성능 평가를 하였다.

4.3.1 SQL Injection(CWE-89)

결함라인 주변의 소스코드 범위가 10라인일 때 가장 높은 예측 성공률(92.04%)을 보여 주었고 소스코드 범위가 커질수록 예측 성공률이 낮아지는 경향을 보여 주었다.

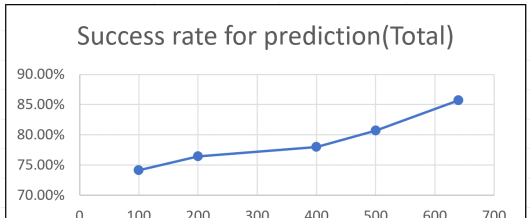


Source code range (number of lines)	Success rate for prediction (True Positive)	Success rate for prediction (False Positive)	Success rate for prediction (Total)
10	95.49%	88.29%	92.04%
20	92.57%	89.00%	90.86%
40	89.21%	89.63%	89.41%
80	87.73%	90.97%	89.28%

Fig. 4. Success rate for prediction of SQL Injection(CWE-89)

4.3.2 Cross-site Scripting(CWE-79)

결함라인 주변의 소스코드 범위가 640라인일 때 가장 높은 예측 성공률(85.71%)을 보여 주었고 소스코드 범위가 작아질수록 예측 성공률이 낮아지는 경향을 보여 주었다.

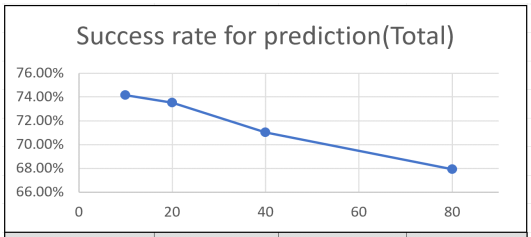


Source code range (number of lines)	Success rate for prediction (True Positive)	Success rate for prediction (False Positive)	Success rate for prediction (Total)
100	76.69%	71.43%	74.13%
200	78.95%	73.81%	76.45%
400	82.71%	73.02%	77.99%
500	77.44%	84.13%	80.69%
640	85.71%	85.71%	85.71%

Fig. 5. Success rate for prediction of Cross-site Scripting(CWE-79)

4.3.3 URL Redirection to Untrusted Site(CWE-601)

결함라인 주변의 소스코드 범위가 10라인일 때 가장 높은 예측 성공률(74.14%)을 보여 주었고 소스코드 범위가 커질수록 예측 성공률이 낮아지는 경향을 보여 주었다.

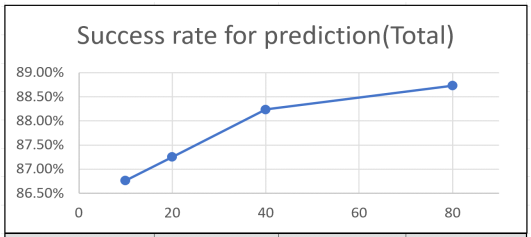


Source code range (number of lines)	Success rate for prediction (True Positive)	Success rate for prediction (False Positive)	Success rate for prediction (Total)
10	76.36%	71.79%	74.14%
20	78.79%	67.95%	73.52%
40	65.45%	76.92%	71.03%
80	70.91%	64.74%	67.91%

Fig. 6. Success rate for prediction of URL Redirection to Untrusted Site(CWE-601)

4.3.4 HTTP Response Splitting(CWE-113)

결함라인 주변의 소스코드 범위가 80라인일 때 가장 높은 예측 성공률(88.73%)을 보여 주었고 소스코드 범위가 작아질수록 예측 성공률이 낮아지는 경향을 보여 주었다.

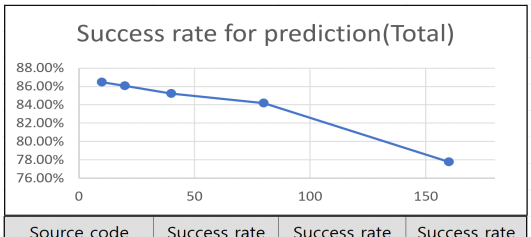


Source code range (number of lines)	Success rate for prediction (True Positive)	Success rate for prediction (False Positive)	Success rate for prediction (Total)
10	88.89%	84.76%	86.76%
20	86.87%	87.62%	87.25%
40	89.90%	86.67%	88.24%
80	87.88%	89.52%	88.73%

Fig. 7. Success rate for prediction of HTTP Response Splitting(CWE-113)

4.3.5 Integer Overflow(CWE-190)

결함라인 주변의 소스코드 범위가 10라인일 때 가장 높은 예측 성공률(86.49%)을 보여 주었고 소스코드 범위가 커질수록 예측 성공률이 낮아지는 경향을 보여 주었다.

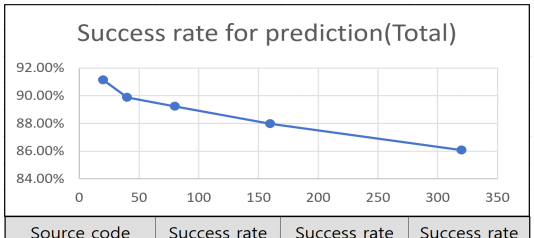


Source code range (number of lines)	Success rate for prediction (True Positive)	Success rate for prediction (False Positive)	Success rate for prediction (Total)
10	82.43%	90.50%	86.49%
20	79.08%	92.98%	86.07%
40	85.77%	84.71%	85.24%
80	78.24%	90.08%	84.20%
160	73.64%	81.82%	77.75%

Fig. 8. Success rate for prediction of Integer Overflow(CWE-190)

4.3.6 Use of Insufficiently Random Values(CWE-330)

결함라인 주변의 소스코드 범위가 20라인일 때 가장 높은 예측 성공률(91.14%)을 보여 주었고 소스코드 범위가 커질수록 예측 성공률이 낮아지는 경향을 보여 주었다.

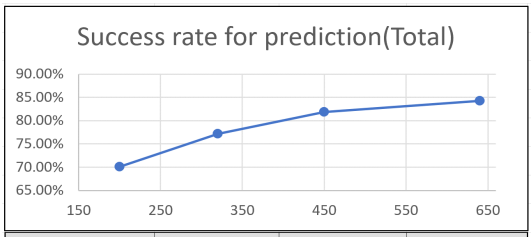


Source code range (number of lines)	Success rate for prediction (True Positive)	Success rate for prediction (False Positive)	Success rate for prediction (Total)
20	92.41%	89.87%	91.14%
40	88.61%	91.14%	89.87%
80	86.08%	92.41%	89.24%
160	84.81%	91.14%	87.97%
320	79.75%	92.41%	86.08%

Fig. 9. Success rate for prediction of Use of Insufficiently Random Values(CWE-330)

4.3.7 Race Condition(CWE-367)

결함라인 주변의 소스코드 범위가 640라인일 때 가장 높은 예측 성공률(84.25%)을 보여 주었고 소스코드 범위가 작아질수록 예측 성공률이 낮아지는 경향을 보여 주었다.

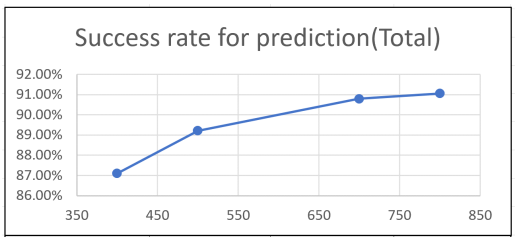


Source code range (number of lines)	Success rate for prediction (True Positive)	Success rate for prediction (False Positive)	Success rate for prediction (Total)
200	60.32%	79.69%	70.08%
320	61.90%	92.19%	77.17%
450	76.19%	87.50%	81.89%
640	73.02%	95.31%	84.25%

Fig. 10. Success rate for prediction of Race Condition(CWE-367)

4.3.8 Information Exposure(CWE-209)

결함라인 주변의 소스코드 범위가 800라인일 때 가장 높은 예측 성공률(91.05%)을 보여 주었고 소스코드 범위가 작아질수록 예측 성공률이 낮아지는 경향을 보여 주었다.

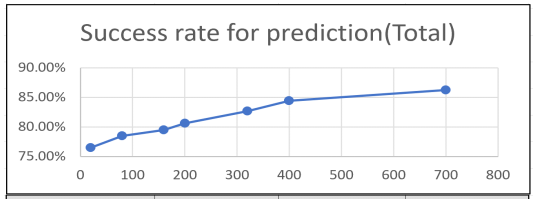


Source code range (number of lines)	Success rate for prediction (True Positive)	Success rate for prediction (False Positive)	Success rate for prediction (Total)
400	83.60%	90.58%	87.11%
500	80.95%	97.38%	89.21%
700	89.95%	91.62%	90.79%
800	89.42%	92.67%	91.05%

Fig. 11. Success rate for prediction of Information Exposure(CWE-209)

4.3.9 Improper Check for Exceptional Conditions(CWE-754)

결함라인 주변의 소스코드 범위가 700라인일 때 가장 높은 예측 성공률(86.26%)을 보여 주었고 소스코드 범위가 작아질수록 예측 성공률이 낮아지는 경향을 보여 주었다.

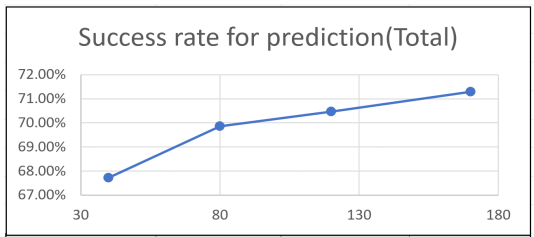


Source code range (number of lines)	Success rate for prediction (True Positive)	Success rate for prediction (False Positive)	Success rate for prediction (Total)
20	76.20%	76.76%	76.49%
80	72.44%	84.22%	78.47%
160	72.29%	86.37%	79.50%
200	81.63%	79.63%	80.60%
320	79.07%	86.08%	82.66%
400	84.79%	84.07%	84.42%
700	80.72%	91.54%	86.26%

Fig. 12. Success rate for prediction of Improper Check for Exceptional Conditions(CWE-754)

4.3.10 NULL Pointer Dereference(CWE-476)

결함라인 주변의 소스코드 범위가 170라인일 때 가장 높은 예측 성공률(71.28%)을 보여 주었고 소스코드 범위가 작아질수록 예측 성공률이 낮아지는 경향을 보여 주었다.

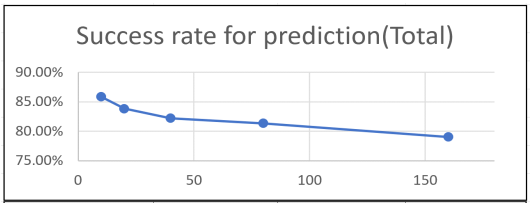


Source code range (number of lines)	Success rate for prediction (True Positive)	Success rate for prediction (False Positive)	Success rate for prediction (Total)
40	71.84%	63.62%	67.72%
80	66.12%	73.58%	69.86%
120	73.06%	67.89%	70.47%
170	66.73%	75.81%	71.28%

Fig. 13. Success rate for prediction of Improper Check for NULL Pointer Dereference(CWE-476)

4.3.11 Improper Resource Release(CWE-404)

결함라인 주변의 소스코드 범위가 10라인일 때 가장 높은 예측 성공률(85.88%)을 보여 주었고 소스코드 범위가 커질수록 예측 성공률이 낮아지는 경향을 보여 주었다.

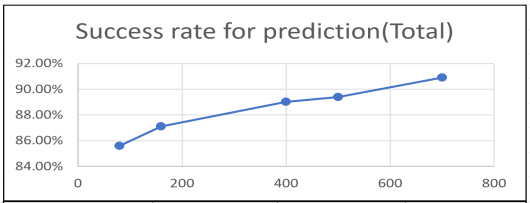


Source code range (number of lines)	Success rate for prediction (True Positive)	Success rate for prediction (False Positive)	Success rate for prediction (Total)
10	86.61%	85.12%	85.88%
20	85.71%	81.86%	83.83%
40	90.18%	73.95%	82.23%
80	83.48%	79.07%	81.32%
160	79.46%	78.60%	79.04%

Fig. 14. Success rate for prediction of Improper Resource Release(CWE-404)

4.3.12 Exposure of Data Element to Wrong Session (CWE-488)

결함라인 주변의 소스코드 범위가 700라인일 때 가장 높은 예측 성공률(90.91%)을 보여 주었고 소스코드 범위가 작아질수록 예측 성공률이 낮아지는 경향을 보여 주었다.

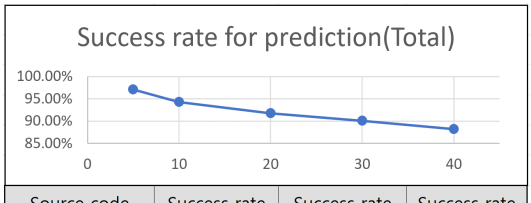


Source code range (number of lines)	Success rate for prediction (True Positive)	Success rate for prediction (False Positive)	Success rate for prediction (Total)
80	87.12%	84.09%	85.61%
160	82.58%	91.67%	87.12%
400	84.85%	93.18%	89.02%
500	82.58%	96.21%	89.39%
700	86.36%	95.45%	90.91%

Fig. 15. Success rate for prediction of Exposure of Data Element to Wrong Session(CWE-488)

4.3.13 Exposure of System Data(CWE-497)

결함라인 주변의 소스코드 범위가 5라인일 때 가장 높은 예측 성공률(97.05%)을 보여 주었고 소스코드 범위가 커질수록 예측 성공률이 낮아지는 경향을 보여 주었다.



Source code range (number of lines)	Success rate for prediction (True Positive)	Success rate for prediction (False Positive)	Success rate for prediction (Total)
5	96.86%	97.26%	97.05%
10	94.12%	94.47%	94.29%
20	92.39%	91.04%	91.74%
30	90.90%	89.11%	90.03%
40	86.80%	89.62%	88.16%

Fig. 16. Success rate for prediction of Exposure of System Data(CWE-497)

4.3.14 소스코드 범위에 따른 최대·최소 성능 비교

결함라인 주변의 소스코드 범위에 따른 최고 성능과 최소 성능을 비교하면 실험 대상 모두에서 성능이 향상됨을 보여주었다.

Race Condition(CWE-367)의 경우에 14.17%로 성능 향상률이 가장 높았고 HTTP Response Splitting(CWE-113)의 경우에 1.97%로 낮았다.

Table 2. CWEs' performance improvement rate

CWE	Lowest Success rate (Total)	Highest Success rate (Total)	Rise rate (Total)
Exposure of System Data(CWE-497)	88.16%	97.05%	△8.89%
SQL Injection (CWE-89)	89.28%	92.04%	△2.76%
Use of Insufficiently Random	86.08%	91.14%	△5.06%

CWE	Lowest Success rate (Total)	Highest Success rate (Total)	Rise rate (Total)
Values(CWE-330)			
Information Exposure(CWE-209)	87.11%	91.05%	△3.94%
Exposure of Data Element to Wrong Session(CWE-488)	85.61%	90.91%	△5.30%
HTTP Response Splitting(CWE-113)	86.76%	88.73%	△1.97%
Integer Overflow(CWE-190)	77.75%	86.49%	△8.74%
Improper Check for Exceptional Conditions (CWE-754)	76.49%	86.26%	△9.77%
Improper Resource Release(CWE-404)	79.04%	85.88%	△6.84%
Cross-site Scripting(CWE-79)	74.13%	85.71%	△11.58%
RaceCondition (CWE-367)	70.08%	84.25%	△14.17%
URL Redirection to Untrusted Site (CWE-601)	67.91%	74.14%	△6.23%
NULL Pointer Dereference (CWE-476)	67.72%	71.28%	△3.56%

4.3.15 기계학습을 활용한 정적 분석과 기존의 정적 분석 방법의 비교

CWE-497의 경우 정·오탐 분류 예측 성공률이 97.05%로 상대적으로 높은 성공률을 보여주고 있어서 기존의 사람이 하던 전문가 분석을 기계학습을 활용하는 방법으로 실험적인 적용이 가능하다. 이 경우의 절차적인 비교는 Fig 17과 같다.

홈페이지 1개당 보안약점은 평균 2,400[12]개 정도 발견된다. 그리고 발견된 보안약점에 대해서 전문가가 정·오탐 분석을 하면 평균 1일(8시간) 정도 소요가 된다. 이것을 1개의 보안약점을 분석하는 것으로 계산하면 12초가 소요된다.

본 연구의 기계학습을 이용한 정·오탐 분석의 경우 보안약점 2,400개를 분석하는데 432초(7분 12초)가 소요된다. 이것을 1개의 보안약점을 분석하는 것으로 계산하면 0.18초가 소요되어 사람에 의한 전문가 분석과 비교하여 많은 시간을 절약되는 것을 기대할 수 있다.

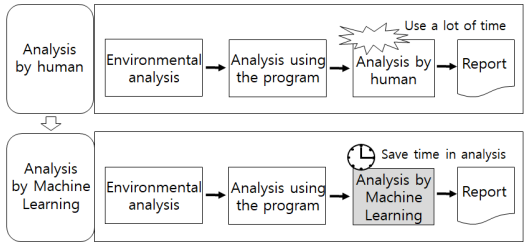


Fig. 17. Analysis procedure comparison

V. 결 론

본 논문에서는 기계학습을 이용하여 정적 분석 도구의 결과에 대한 정·오탐 분석 과정을 효율화하는 방법을 알아봤다.

먼저 기계학습에 사용하는 학습 데이터(결함라인 주변의 소스코드)의 범위에 변화를 주었을 때 정·오탐 판별 성능 변화를 실험하고 평가하였다.

CWE-89, CWE-601, CWE-190, CWE-330, CWE-404, CWE-497의 경우에는 결함라인 주변의 소스코드 범위를 작게 하여 학습 데이터로 사용할 때 정·오탐 판별 성공률이 높아지는 것을 확인하였다.

CWE-79, CWE-113, CWE-367, CWE-209, CWE-754, CWE-476, CWE-488의 경우는 결함라인 주변의 소스코드 범위를 크게 하여 학습 데이터로 사용할 때 정·오탐 판별 성공률이 높아지는 것을 확인하였다. 결국 기계학습에 사용하는 학습 데이터(결함라인 주변의 소스코드)의 범위를 최적화 하였을 때 정·오탐 판별 성능이 향상되는 것을 확인하였다.

또한, 사람에 의한 전문가 정·오탐 분석 시간과 기계학습을 이용한 분석 시간 비교를 통해서 기계학습을 이용하면 많은 시간을 절약할 수 있는 기대 효과를 확인할 수 있다.

하지만 기존 전문가 분석을 대체할 수준의 성능은 보이지 않아 전문가 분석의 보조적인 역할로 활용이 필요하다고 생각된다.

향후 더욱더 다양하고 많은 학습 데이터를 확보하고 소스코드의 전처리에 대한 추가적인 연구를 수행한다면 정·오탐 판별을 포함한 전문가들의 소스코드 정적 분석에 도움을 줄 수 있을 것으로 예상된다.

References

[1] U.S. Department of Homeland Security (DHS), Software Assurance, https://us-cert.cisa.gov/sites/default/files/publications/infosheet_SoftwareAssurance.pdf

[2] Google Plus Will Be Shut Down After User Information Was Exposed, https://www.nytimes.com/

[3] Coding Error Sends 2019 Subaru Ascents to the Car Crusher, https://spectrum.ieee.org

[4] Hongjun Choi, Jinyoung Choi, Software Development Life Cycle and Static Analysis Tool, 818-819, 2019

[5] Software security weakness diagnosis guide for e-government software development security inspectors, pp. 26-27, 2019

[6] Martin White, Michele Tufano et al, deep learning code fragments for code clone detection, 2016

[7] Hoa Khanh Dam, Truyen Tran, Trang Pham, A deep language model for software code, 2016

[8] Xi Victoria Lin, Chenglong Wang, et al, Program Synthesis from Natural Language Using Recurrent Neural Networks, 2017

[9] Rebecca L. Russell, Louis Kim, et al, Automated Vulnerability Detection in Source Code Using Deep Representation Learning, 2018

[10] Won-kyung Lee, Min-Ju Lee, Dongsu Seo, Application of Machine Learning Techniques for the Classification of Source Code Vulnerability, pp. 6-7 2020

[11] Youngho Lee, Seong-Yun Hong, A machin learning approach to the prediction of indifidual travel mode choices, 1011-1024, 2019

[12] Jiho Bang, Rhan Ha, Evaluation Methodology of Diagnostic Tool for Security Weakness of e-GOV Software, pp. 336 The Korean Institute of Communications and Information Sciences 2013-04 Vol.38C No.04

〈저자소개〉

박 양 환 (Yang-Hwan Park) 정회원
2003년: 인천대학교 컴퓨터공학과 (학사)
2016년: 고려대학교 정보보호학과 석사과정
2015년~현재: 한국인터넷진흥원 수석연구원
〈관심분야〉 시큐어코딩, AI기반 취약점 분석, 정보보호

최 진 영 (Jin-Young Choi) 종신회원
1982년: 서울대학교 컴퓨터공학과 (학사)
1986년: 미국 Drexel University, Dept. of Mathematics and Computer Science (석사)
1993년: 미국 Univ. of Pennsylvania, Dept. of Computer and Information Science (박사)
1996년~현재: 고려대학교 정보보호대학원 정보보호학과 교수
〈관심분야〉 정형기법, 시큐어 소프트웨어공학, 소프트웨어보증