파인튜닝을 사용한 대형 언어 모델 기반 취약점 탐지 시스템에 대한 연구

김현준[1], 윤수빈[1], 백윤흥[1]
[1]서울대학교 전기정보공학부, 반도체공동연구소
hjkim@sor.snu.ac.kr, sbyun@sor.snu.ac.kr, ypaek@snu.ac.kr

A Survey on Fine-tuned Large Language Model-based Vulnerability Detection System

Hyun-Jun Kim[1], Subin Yun[1], Yun-Heung Paek[1]
[1]Dept. of Electrical and Computer Engineering and Inter-University Semiconductor Research Center (ISRC), Seoul National University

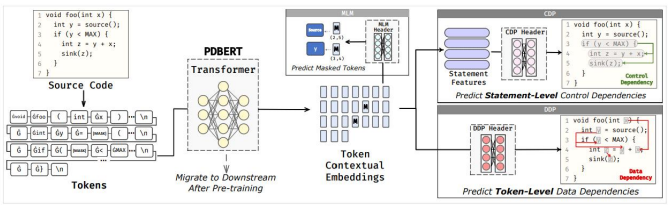
요 약

본 논문은 소스 코드에 내재된 취약점을 탐색하기 위해 대형 언어 모델을 취약점 탐색 태스크에 맞게 파인튜닝하여 사용하는 최신 연구들을 소개한다. 각 연구에서 대형 언어 모델을 활용하여 중점적으로 해결하려는 문제와 솔루션을 설명하고, 향후 연구 방향을 조망하려 한다.

1. 서론

소프트웨어 코드의 규모가 커지고 복잡해짐에 따라 더욱 많은 수의 코드 취약점들이 생겨나고 있으며, 이에 따라 코드 내 취약점을 악용한 공격들 역시 점점 증가하고 있다. 한국인터넷진흥원에서 발표한 사이버 위협 보고서에 의하면 소프트웨어의 취약점을 노린 침해 사례들이 점점 증가하고 있다고 보고하고 있다[1]. 이러한 공격들을 사전에 차단하기 위해서 개발자들은 최대한 코드 내 취약점들을 파악하고, 패치를 통해 해당 취약점들을 제거해야 한다. 취약점 탐색 (vulnerability detection) 기법은 어떤 소프트웨어의 소스 코드를 분석하여 해당 코드 내에 포함되어 있는 취약점을 찾아내는 기법들을 통칭한다. 이 중에서 딥러닝 모델을 활용하여 취약점의 패턴을 소프트웨어 소스 코드 내에서 찾아내는 연구들이 많이 진행되었으며, 데이터 내에서 효율적으로 특정 패턴을 찾아내는 데에 특화된 딥러닝의 특성을 활용하여 높은 탐지 성능을 달성하는 데에 성공하였다. 최근에는 프로그램 코드와 자연어를 동시에 학습하여 코드와 관련된 다양한 태스크를 수행할 수 있는 코드 특화 대형 언어 모델 (large language model)을 코드 내 취약점을 찾는 데에 활용하려는 시도들이 있다. 대형 언어 모델은 추가 학습 없이 퓨샷 (few-shot) 방식 혹은 프롬프트 엔지니어링 (prompt engineering)을 통해 원하는 태스크를 수행하는 것이 가능하다. 또한, 대형 언어 모델 전체 혹은 일부를 파인 튜닝 (fine-tuning)하여 특정 태스크에 특화하는 것이 가능하다. 본 논문에서는 파인 튜닝을 통해 대형 언어 모델을 코드 취약점 탐색 태스크에 알맞게 학습한 연구들을 소개하고, 향후 어떤 방향으로 코드 취약점 탐지 연구를 진행할 수 있을지를 조망하려 한다.

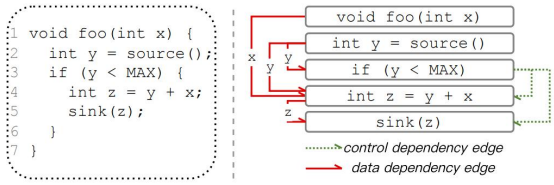
2. PDBERT[2]



(그림 1) PDBERT 구조도

PDBERT는 CodeBERT[3] 라는 코드 특화 대형 언어 모델을 파인튜닝하여 취약점 탐색을 시도하였다. 해당 연구에서는 바로 취약점 탐색 태스크를 학습하는 것이 아니라, CodeBERT 모델이 코드를 구성하는 각 요소들 간의 제어 의존성 (control dependency) 및 데이터 의존성 (data dependency) 관계를 이해할 수 있도록 사전 학습 (pre-training)을 먼저 수

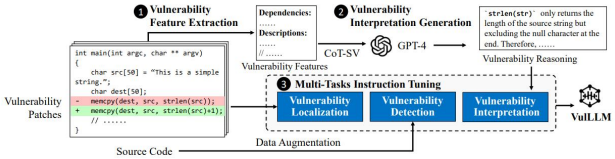
행한다. 첫 번째로, 제어 의존성 학습은 코드 조각 (code statement) 단위로 코드 조각 간의 제어 의존성을 학습하여 어떤 프로그램 소스 코드에 대해 제어 의존성을 예측할 수 있도록 학습한다.



(그림 2) 프로그램 의존성 그래프

코드 간의 의존성은 프로그램 의존성 그래프 (program dependency graph, PDG)로 나타내며, 코드 조각은 노트 (node), 제어 의존성과 데이터 의존성은 엣지 (edge)로 나타낸다. 여기서 제어 의존성 관계는 행렬로 표현되며 i번째 행, j번째 열 원소는 i번째 코드 조각이 j번째 코드 조각에 제어 의존성 관계를 가지는 것을 나타낸다. 학습에 사용되는 정답인 레이블 (label)은 joern[4]이란 분석 툴을 통해 코드를 분석해서 획득한다. 두 번째로, 데이터 의존성은 좀 더 세밀한 단위인 토큰 (token) 단위로 따져서 변수 간의 관계를 예측할 수 있도록 한다. 하지만 토큰의 수가 매우 많기 때문에 연속된 토큰 시퀀스 (sequnece) 중 첫 번째 토큰끼리의 의존성 관계만 엣지로 표현한다. 두 개의 태스크 모두 1개의 레이어 (layer)로 구성된 다층 퍼셉트론 (multi-layer perceptron) 분류기 (classifier)를 CodeBERT 뒤에 추가하여 학습을 수행한다. 여기에 기존 마스킹 방식 태스크까지 총 3개의 태스크들의 손실함수 (loss function)들로 구성된 통합 손실함수를 최적화하도록 학습을 수행한다. 그 이후에 함수 내에 취약점 여부를 (없으면 0, 있으면 1) 예측하는 태스크를 학습한다. 코드 내의 의존성 관계를 이해하게 된 모델은 의존성들로 표현되는 취약점의 패턴을 좀 더 잘 이해할 수 있게 되어 높은 수준의 취약점 탐색이 가능하게 되었다.

3. VulLLM [5]



(그림 3) VulLLM 구조도

VulLLM은 기존 코드 특화 대형 언어 모델에 3가지 태스크에 대한 명령어와 정답 쌍을 학습시켜 다중 태스크 명령어 파인튜닝 (Multi-task instruction finetuning)을 수행하였다. 첫 번째 태스크는 어떤 함수가 취약점을 포함되었는지 알아내는 바이너리 태스크이다. 두 번째 태스크는 함수 내에 취약한 코드가 몇 번째 줄에 있고, 해당 줄의 내용을 출력하는 태스크이다. 세 번째 태스크는 함수 내의 취약점을 설명하는 태스크이다. 여기서 정답으로 활용되는 '취약점에 대한 설명'은 ChatGPT를 활용하여 생성하며, 소스 코드 외에 CWE 취약점 타입, 취약점에 대한 정보, 취약한 코드와 함수 내 다른 코드 간의 제어 의존성 및 데이터 의존성을 입력으로 넣어서 ChatGPT가 취약점에 대한 설명을 생성하게 하였다. 그 다음 전문가들이 올바른 설명이 출력된 케이스들만 필터링해서 학습 데이터셋에 추가하였다. 3가지 태스크가 포함된 학습 데이터셋으로 학습된 VulLLM은 취약점의 일반적인 패턴을 충분히 학습하여 테스트 데이터셋의 분포가 학습 데이터셋과 다르더라도 높은 취약점 탐지 성능을 달성하였다.

4. PrimeVul [6]

해당 연구는 기존 취약점 탐색 연구들에서 사용했던 BigVul[7] 데이터셋에 내재된 문제점들을 해결하여 정제된 취약점 데이터셋을 제안하였다. BigVul은 중복된 데이터가 많으며, 취약점 여부에 대한 정답이 잘못 마킹된 경우가 많아 이를 모두 수정하였다. 취약점 데이터셋은 깃헙 (github) 등의 프로그램 리포지토리에서 취약점에 대한 패치가 진행되기 전/후의 코밋 (commit)을 받아서 그 차이를 기반으로 생성되는데, 취약점과 상관 없는 다른 수정 내용이 포함되는 경우가 있어 이를 추가로 보정하였다. 그리고 데이터셋을 패치 전/후 쌍 (pair)으로 구성하여 취약점의 유무를 명확하게 나타낼 수 있도록 구성하였다. 또한 테스트 데이터셋에 있는 함수보다 더 미래에 수정된 동일한 함수가 학습 데이터셋에 포함되면 과거 버전에서 패치된 코드 내용이 미래 버전에 똑같이 포함되어 있을 가능성이 있고, 이는 테스트 데이터셋에 학습 데이터셋에 있는 내용이 그대로 포함되는 것이므로 정당한 검증을 방해할 수 있다. 따라서 테스트 데이터셋은 항상 학습 데이터셋보다 미래의 커밋만 포함하도록 데이터셋을 시간순으로 나누어 구성하였다. 대형 언어 모델을 PrimeVul로 학습해서 확인한 결과, 대형 언어 모델이 BigVul과는 달리 실제로는 취약점 탐색을 잘 하지 못함을 증명

<표 1> PrimeVul 테스트셋을 활용한 기존 연구들 성능 비교

관련 논문	학습 데이터셋	사용 모델 (패러미터 수)	F1	MCC	정확도	정밀도	재현율	TP	TN	FP	FN
VulLLM [5]	mixvul [5]	codeLlama (7B)	0.6462	-0.0032	0.4991	0.4995	0.9149	516	47	517	48
VulLLM [5]	mixvul [5]	codeLlama (13B)	0.6089	-0.0317	0.4876	0.4923	0.7979	450	100	464	114
PDBERT [2]	PrimeVul [6]	codeBERT (125M)	0.6576	0.0298	0.5071	0.5038	0.9468	534	38	526	30

하였다. 이는 취약한 코드의 전반적인 패턴은 어느 정도 학습하지만, 패치 전후의 미세한 차이를 제대로 모델이 학습하지 못하고, 취약점의 궁극적인 원인을 이해하고 학습하기보다는 변수명 등 상황에 따라 바뀔 수 있는 요소들에 의존하기 때문이다.

5. 차후 연구 방향

PDBERT와 VulLLM을 학습한 후 PrimeVul 테스트 셋으로 검증한 결과를 <표 1>에 포함하였다. F1 스코어만 보면 적당한 수준의 성능을 보이는 것처럼 보이지만, 실제로는 대부분의 경우에 1 (취약함)이라고 대답하고 있어 정상적으로 동작하고 있지 않다. <표 1>에서 정탐 (TN, True Negative) 및 미탐 (FN, False Negative)의 수가 낮고, 매튜 상관계수 (MCC, Matthews Correlation Coefficient)가 낮은 것으로부터 이를 알 수 있다. 차후 연구는 패치 전후의 취약한 코드와 취약하지 않은 코드의 차이를 모델이 학습할 수 있게 하는 것이 중요하며, 이를 위해 코드 난독화 (code obfuscation) 등의 방법을 통해 변수명, 함수명 등에 모델이 의존하지 않으면서도 취약점의 원인 자체를 이해하고 학습할 수 있는 기법을 개발할 수 있을 것으로 기대된다.

6. 결론

본 논문에서 대형 언어 모델을 파인튜닝해서 코드 내 취약점을 탐지하는 여러 연구들의 동향을 살펴보았다. 차후에는 성능을 좀더 개선할 수 있을 것으로 기대된다.

7. Acknowledgement

이 논문은 2024년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원(RS-2023-00277326)과 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(IITP-2023-RS-2023-00256081)과 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(No.2020-0-01840,스마트폰의 내부데이터 접근 및 보호 기술 분석)과 2024년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(No.RS-2024-00438729, 익명화된 기밀실행을 이용한 전주기적 데이터 프라이버시 보호 기술 개발)과 2024년도 정부(산업통상자원부)의 재원으로 한국산업기술기획평가원의 지원(No. RS-2024-00406121, 자동차보안취약점기반위협분석시스템개발(R&D))을 받았으며, 2024년도 BK21 FOUR 정보기술 미래인재 교육연구단의 지원을 받았으며, 반도체 공동연구소 지원을 받아 수행된 연구임.

참고문헌

[1] 황찬웅, “2023년 하반기 사이버 위협 동향 보고서”, p. 4, 2024, https://www.kisa.or.kr/20205/form?postSeq=1025&page=1

[2] Liu, Zhongxin, et al., "Pre-training by Predicting Program Dependencies for Vulnerability Analysis Tasks.", Proceedings of the IEEE/ACM 46th International Conference on Software Engineering, Portugal, 2024.

[3] Feng, Zhangyin, et al., "Codebert: A pre-trained model for programming and natural languages.", arXiv preprint arXiv:2002.08155, 2020.

[4] Fabian Yamaguchi, joern [Computer software], joernio, https://github.com/joernio/joern.

[5] Du, Xiaohu, et al., "Generalization-Enhanced Code Vulnerability Detection via Multi-Task Instruction Fine-Tuning", The 62nd Annual Meeting of the Association for Computational Linguistics Findings, Thailand, 2024.

[6] Ding, Yangruibo, et al., "Vulnerability Detection with Code Language Models: How Far Are We?", IEEE/ACM 46th International Conference on Software Engineering, Canada, 2025.

[7] Fan, Jiahao, et al., "AC/C++ code vulnerability dataset with code changes and CVE summaries.", Proceedings of the 17th International Conference o n Mining Software Repositories, Online, 2020.