딥러닝 기반 취약점 탐색 기술에 대한 조망

김현준[1], 안선우[1], 안성관[1], 백윤흥[1]
[1]서울대학교 전기정보공학부, 반도체공동연구소
hjkim@sor.snu.ac.kr, swahn@sor.snu.ac.kr, sgahn@sor.snu.ac.kr, ypaek@sor.snu.ac.kr

A Survey on Deep Learning-Based Vulnerability Detection Technique

Hyun-Jun Kim[1], Sun-woo Ahn[1], Seong-gwan Ahn[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 논문에서는 소프트웨어에 내장된 취약점의 패턴을 인식하여 찾아내는 딥러닝 기반 취약점 탐색 기술에 대해 소개한다. 특정 소프트웨어의 소스 코드 혹은 바이너리 코드를 분석하여 취약점을 찾아내는 여러 기법들을 살펴본 다음, 딥러닝 기반 바이너리 취약점 탐색 기술의 향후 연구 방향을 조망하고자 한다.

1. 서론

최근 개발되는 소프트웨어의 수와 복잡성이 증대해짐에 따라 보고되는 소프트웨어 취약점의 수가 점점 증가하고 있다. CVE (Common Vulnerabilities and Exposures) 프로그램에서 2021년에 수집된 취약점의 수는 약 2만개[1]로, 2020년에 수집된 취약점의 개수보다 약 10% 정도 증가하였다. 이 취약점들은 사이버 공격의 주요 대상이 되며, 해당 소프트웨어가 구동되는 시스템을 망가뜨리는 위협이 된다.

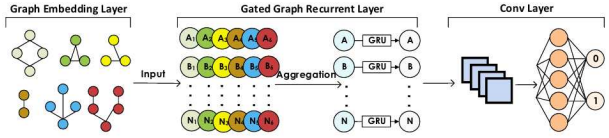
개발자들은 소프트웨어를 배포하기 이전에 보안 위협을 최소화하기 위해 취약점을 제거할 필요가 있다. 소프트웨어에 내재된 수많은 취약점을 사람이 직접 분석하여 찾아내는 데에 많은 시간과 노동력이 소모된다. 이러한 비용을 최소화하기 위해 취약점을 자동으로 찾아내는 기법에 대한 많은 연구가 진행되었다.

취약점을 자동으로 찾아내는 기술 중 한 갈래로 퍼징(fuzzing)[2] 기법이 있다. 해당 기법은 프로그램에 내재된 취약점을 발동(trigger)시키는 입력값들을 찾아내는 것이 목적이며, 프로그램을 분석하여 빠르게 적절한 입력값들을 생성하는 아이디어들이 제안되었다. 하지만 각 입력값에 대해 프로그램을 구동하여 테스트하므로 많은 시간과 리소스(resource)가 소모된다는 단점이 있다.

동적으로 구동되는 퍼징 기법 외에 소프트웨어의 코드를 정적으로 분석하여 취약점을 찾아내는 취약점 탐색 기술(vulnerability detection)이 있다. 해당 기법은 취약점들이 소스 코드 혹은 바이너리 코드 레벨에서 나타내는 패턴을 대상 소프트웨어에서 식별하여 취약점을 찾는 기술이며, 코드 유사도 측정 기술 (code similarity measurement)에 기반을 두어 발전되어 왔다. 최근에는 패턴 식별에 강점이 있는 머신 러닝, 딥러닝 모델이 많이 적용되고 있다. 본 논문에서는 딥러닝을 기반으로 한 취약점 탐색 기술들에 대해 살펴보고, 이를 기반으로 향후 취약점 탐색 기술의 방향을 탐색하고자 한다.

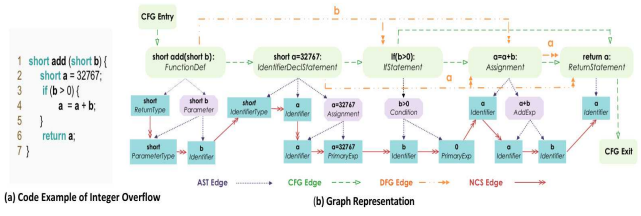
2. 딥러닝 기반 소스 코드 취약점 탐색 기술

2-1. Devign[3]



(그림 1) Devign에서 사용된 취약점 탐색 모델

해당 연구에서는 그래프 신경망(graph neural network)을 활용하여 취약점이 포함된 코드의 시맨틱 표현 (semantic representation, 의미 표현)을 학습한다. 학습의 대상이 되는 그래프는 다양한 취약점들의 특성을 표현할 수 있어야 하므로 프로그램의 특성을 반영 가능한 표현형들의 결합으로 생성된다.



(그림 2) 정수 오버플로우 취약점에 대한 그래프

대상 프로그램을 표현할 그래프는 추상 구문 트리(AST, abstract syntax tree), 제어 흐름 그래프(CFG, control flow graph), 데이터 흐름 그래프(DFG, data flow graph)가 결합되어 만들어지며, 추가로 소스 코드 내 토큰(token)들의 순서(sequential) 특성을 반영하기 위해 소스 코드상에서 이웃한 토큰에 대응되는 노드(node)끼리 간선(edge)을 추가해준다. 각 노드는 추상 구문 트리의 노드를 기준으로 생성되며, 해당 노드에 대응되는 코드 구문을 언어 임베딩 모델 중 하나인 Word2Vec 모델을 사용하여 임베딩 벡터(embedidng vector)를 생성하고, 추상 구문 트리 타입(type)과 함께 노드에 할당한다.

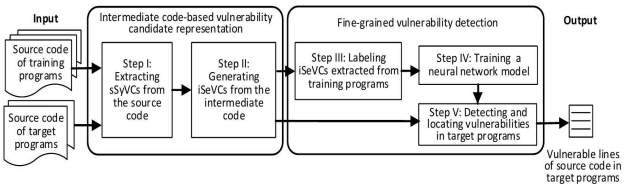
또한 각 노드에 대해 주변 이웃 노드들의 정보를 반영하기 위해 게이트 그래프 순환 레이어 (Gated Graph Recurrent Layer) 모델[4]을 학습하고, 노드별 최종 표현형을 얻는다.

마지막 단계에서 동작하는 'Conv' 모듈은 특정 함수가 취약한지 취약하지 않은지를 출력하는 분류기(classifier)이다. 해당 모듈은 1차원 컨볼루션(1D Convolution) 레이어를 활용하여 현재 그래프 레벨에서 관련성이 높은 노드들을 선별할 수 있으며, 해당 노드들을 바탕으로 해당 함수의 취약한 정도를 정확하게 판단할 수 있다.

위 프로세스를 차례대로 구동하였을 때 실험적으로 리눅스 커널, qemu, wireshark, ffmpeg와 같은 프로그램 내의 취약점을 잘 식별할 수 있음을 보였다. 해당 연구는 취약점의 특성을 그래프 표현형으로 나타내고, 그래프의 특성을 잘 반영하는 특수한 그래프 신경망 모델들을 학습시켜 높은 정확도로 취약점을 찾아낼 수 있다는 것을 보였다.

2-2. VulDeeLocator[5]

해당 연구는 소스 코드를 컴파일할 때 컴파일러 내에서 내부적으로 사용되는 중간 표현(intermediate representation)을 활용하였고, 중간 표현 토큰 레벨로 그래뉼러리티(granularity)를 조정할 수 있게 설계하여 취약한 코드 조각의 위치를 상대적으로 정확하게 찾을 수 있게 하였다.



(그림 3) VulDeeLocator 구조도

먼저 소스코드로부터 sSyVC(source code- and Syntax-based Vulnerability Candidate)라는 코드 조각을 추출한다. 이 코드 조각은 추상 구문 트리로 표현되었을 때 API 함수 콜, 배열(array), 포인터(pointer), 산술 표현식(arithmetic expression) 타입을 가지는데, 대부분의 취약한 코드들은 해당 타입의 코드 조각들과 의존 관계(dependency)를 가진다.

그 다음, 소스코드를 LLVM 컴파일러를 사용해 중간 표현으로 변환하고, 중간 표현 내에서 각 sSyVC에 대해 dependency 관계에 있는 중간 표현 명령어들로 구성된 중간 표현 조각(slice)을 모아서 iSeVC(intermediate code- and Semantics-based Vulnerability Candidate)를 생성한다.

iSeVC를 구성하는 중간 표현 명령어 토큰은 언어 임베딩 모델에 의해 임베딩 벡터로 변환이 되며, 특수한 양방향 LSTM 모델(BRNN-vdl)에 입력으로 들어가게 된다. 해당 모델은 입력으로 들어온 iSeVC가 취약한 코드인지 아닌지를 출력하게끔 학습이 된다.

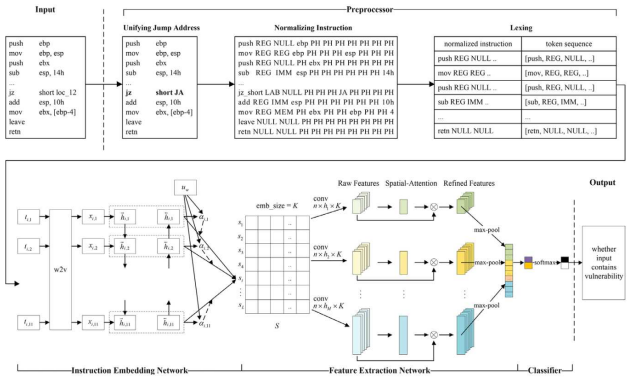
BRNN-vdl은 추가로 어텐션(attention) 레이어와 k-최대값(k-max) 풀링(pooling) 레이어를 사용하여 취약한 코드에 해당하는 토큰들의 위치 정보를 추가로 반영할 수 있게 한다. 따라서 취약하지 않은 코드와 취약한 코드에서 모두 나타나더라도 위치가 달라 다른 역할을 하는 토큰들을 서로 구별할 수 있게 된다. 또한 어떤 iSeVC가 취약하다고 판단되었을 때 k-최대값 풀링 레이어에서 선별된 k개의 토큰들이 해당 iSeVC가 취약하다고 판단하는 데에 큰 영향을 끼쳤으므로 해당 토큰들이 취약한 코드에 해당됨을 알 수 있고, 소스 코드로 역매핑해서 취약한 코드 라인(code line)을 정확하게 알 수 있다.

해당 연구는 wireshark, ffmpeg, libav와 같은 프로그램에서 프로그램 내의 취약점을 잘 식별할 수 있음을 보였으며, 취약점의 위치를 코드 라인 레벨로 정확하게 찾아낼 수 있다는 점에서 큰 의의가 있다.

3. 딥러닝 기반 바이너리 코드 취약점 탐색 기술

바이너리 코드는 컴파일 과정에서 많은 정보가 소실되기 때문에 취약점의 특성을 찾아내기 더 어려우며, 관련된 연구의 수도 적다.

3-1. HAN-BSVD[6]



(그림 4) HAN-BSVD 구조도

본 연구에서는 바이너리의 취약점을 탐색하는 것을 목표로 하며, 이에 따라 바이너리를 디스어셈블리(disassembly) 후 명령어 레벨에서 분석하게 된다. 각 명령어에 대한 임베딩 벡터를 생성하기에 앞서 먼저 전처리(preprocessing) 단계를 거치게 된다. 명령어 중 점프 (jump) 명령어의 address 값까지 고려한다면 노이즈가 포함되어 제대로 임베딩을 할 수 없으므로 address 값은 제거해준다. 따라서 프로그램의 제어 흐름은 임베딩 시 고려하지 않는다. 그 외의 다른 명령어들도 특정 포맷에 따라 정규화(normalize)를 해서 11개의 심볼 요소들로 변환을 해준다. 각 심볼에 대해서는 언어 임베딩 모델인 Word2Vec을 적용해서 토큰별 임베딩 벡터를 생성한다. 그리고 하나의 명령어 내의 11개 심볼 요소에 대해 게이트 순환 유닛 (GRU, Gated Recurrent Unit) 신경망과 어텐션 기법을 적용하여 문맥적 관계가 포함된 임베딩 벡터를 생성한다.

생성된 임베딩 벡터들을 명령어 순서대로 배열한 다음 Text-CNN[7]을 활용하여 중요한 영역의 특성을 중점적으로 반영하여 타겟 바이너리의 특성 벡터를 추출한다. 추출된 특성 벡터를 입력값으로 하는 분류기를 학습시켜 해당 바이너리에 취약점이 있는지 없는지를 판별한다.

해당 연구는 다양한 딥러닝 모델들을 활용하여 효과적으로 바이너리 내의 취약점을 탐색해내는 데에 성공하였다.

4. 차후 연구 방향

소스 코드에 대해 취약점을 탐색하는 연구는 다량 진행되었지만, 바이너리 코드를 대상으로 하는 연구는 충분하지 않은 상황이며, 좀 더 높은 성능을 가지는 취약점 탐색 기술을 개발할 수 있을 것이다.

딥러닝 모델을 기반으로 하는 취약점 탐색 기술을 회피하는 연구도 생각해 볼 수 있다. 실제로는 취약한 코드이지만 취약하지 않은 코드의 패턴을 가지는 바이너리 코드를 생성하는 것을 목표로 둘 수 있다. 이와 같이 악의적으로 취약점이 숨겨진 바이너리가 배포된다면 여러 시스템의 보안성을 해칠 수 있다. 해당 연구에 적합한 기술로 적대적 예제 (Adversarial Examples) 기술을 들 수 있다. 적대적 예제 기술은 취약한 코드의 표현형과 유사하지만 취약하지 않다고 판별되는 코드 표현형을 생성할 수 있다. 하지만 그 표현형을 나타내면서도 취약성을 그대로 유지하는 바이너리 코드를 바로 생성하는 것은 쉽지 않으며, 이에 대한 추가 연구가 필요하다.

5. 결론

본 논문에서는 딥러닝 모델을 활용한 취약점 탐색 기술을 다룬 연구들의 동향을 살펴보았고, 각 연구에서 사용한 딥런이 모델과 취약점의 특성을 반영하는 방법에 대해 알아보았다. 아직 해당 분야에서 개선할 여지가 남아있으며, 취약점 탐색 기술의 성능을 좀 더 개선할 수 있을 것으로 기대된다.

6. Acknowledgement

이 논문은 2022년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(No.2020-0-01840,스마트폰의 내부데이터 접근 및 보호 기술 분석)과 2022년도 BK21 FOUR 정보기술 미래인재 교육연구단의 지원을 받았으며, 2022년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2020R1A2B5B03095204).

참고문헌

[1] “Browse cve vulnerabilities by date.” CVE Details. 2022년 4월 22일 접속, cvedetails.com/browse-by-date.php

[2] Liang, Hongliang, et al. “Fuzzing: State of the art” IEEE Transactions on Reliability, 67, 3, pp. 1199-1218, 2018.

[3] Zhou, Yaqin, et al. “Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks” Advances in neural information processing systems, Canada, 2019.

[4] Ruiz, Luana, Fernando Gama, and Alejandro Ribeiro. “Gated graph recurrent neural networks” IEEE Transactions on Signal Processing, 68, pp. 6303-6318, 2020.

[5] Li, Zhen, et al. “Vuldeelocator: a deep learning-based fine-grained vulnerability detector” IEEE Transactions on Dependable and Secure Computing, Early Access, 2021.

[6] Yan, Han, et al. “HAN-BSVD: a hierarchical attention network for binary software vulnerability detection” Computers & Security, 108, 102286, 2021.

[7] Guo, Bao, et al. “Improving text classification with weighted word embeddings via a multi-channel TextCNN model“ Neurocomputing, 363, pp. 366-374, 2019.