Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.32, NO.6, Dec. 2022 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2022.32.6.1139

BERT를 이용한 딥러닝 기반 소스코드 취약점 탐지 방법 연구*

김 문 회,[1†] 오 희 국[2‡]
[1,2]한양대학교 컴퓨터공학과 (대학원생, 교수)

A BERT-Based Deep Learning Approach for Vulnerability Detection*

Wenhui Jin,[1†] Heekuck Oh[2‡]
[1,2]Department of Computer Science and Engineering, Hanyang University
(Grauate student, Processor)

요 약

SW 산업의 급속한 발전과 함께 새롭게 개발되는 코드와 비례해서 취약한 코드 또한 급증하고 있다. 기존에는 전문가가 수동으로 코드를 분석하여 취약점을 탐지하였지만 최근에는 증가하는 코드에 비해서 분석하는 인력이 부족하다. 이 때문에 기존 Vuldeepecker와 같은 많은 연구에서는 RNN 기반 모델을 이용하여 취약점을 탐지하였다. 그러나 RNN 모델은 코드의 양이 방대할수록 새롭게 입력되는 코드만 학습되고 초기에 입력된 코드는 최종 예측 결과에 영향을 주지 못하는 한계점이 있다. 또한 RNN 기반 방법은 입력에 Word2vec 모델을 사용하여 단어의 의미를 상징하는 embedding을 먼저 학습하여 고정 값으로 RNN 모델에 입력된다. 이는 서로 다른 문맥에서 다른 의미를 표현하지 못하는 한계점이 있다. BERT는 Transformer 모델을 기본 레이어로 사용하여 각 단어가 전체 문맥에서 모든 단어 간의 관계를 계산한다. 또한 MLM과 NST 방법으로 문장 간의 앞뒤 관계를 학습하기 때문에 취약점 탐지와 같은 코드 간 관계를 분석해야 할 필요가 있는 문제에서 적절한 방법이다. 본 논문에서는 BERT 모델과 결합하여 취약점 탐지하는 연구를 수행하였고 실험 결과 취약점 탐지의 정확성이 97.5%로 Vuldeepecker보다 정확성 1.5%. 효율성이 69%를 증가하였다.

ABSTRACT

With the rapid development of SW Industry, softwares are everywhere in our daily life. The number of vulnerabilities are also increasing with a large amount of newly developed code. Vulnerabilities can be exploited by hackers, resulting the disclosure of privacy and threats to the safety of property and life. In particular, since the large numbers of increasing code, manually analyzed by expert is not enough anymore. Machine learning has shown high performance in object identification or classification task. Vulnerability detection is also suitable for machine learning, as a reuslt, many studies tried to use RNN-based model to detect vulnerability. However, the RNN model is also has limitation that as the code is longer, the earlier can not be learned well. In this paper, we proposed a novel method which applied BERT to detect vulnerability. The accuracy was 97.5%, which increased by 1.5%, and the efficiency also increased by 69% than Vuldeepecker.

Keywords: Deep Learning, Vulnerability Detection, Source Code, BERT, Program Slicing

Received(10. 21. 2022), Modified(12. 02. 2022), Accepted(12. 05. 2022)

* 이 논문은 2022년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2022R1A2C2007255)

† 주저자, jinwenhui@hanyang.ac.kr
‡ 교신저자, hkoh@hanyang.ac.kr(Corresponding author)

I. 서 론

SW 산업의 급속한 발전과 함께 소프트웨어는 일상생활에서 필수요소가 되었다. 컴퓨터 소프트웨어, 휴대폰 애플리케이션 이외에도 무인기와 같은 군사용 무기, 자율자동차와 같은 교통 장치, 인프라에 사용되는 소형 디바이스, 심장을 정상적으로 뛰게 도와주는 스마트 펌프와 같은 의료기기 등 다양한 장치에서 커널과 소프트웨어가 구동하고 있다.

새롭게 개발되는 코드에 비례해서 취약점 또한 급증하고 있다. 취약점이 해커에게 악용되면 사람들은 개인 정보의 유출, 경제적 피해, 생명 건강 등과 같은 큰 피해를 받을 수 있어 취약점을 탐지하는 것이 매우 중요하다.

기존에는 전문가가 직접 코드를 수동으로 분석하여 취약점을 탐지해왔지만 오늘날에는 코드의 복잡도와 코드가 늘어나는 양에 비해 이를 분석할 인력이 아주 부족한 상황이다. 시대의 발전에 따라 취약점 탐지 기술에 정확성뿐만 아니라 방대한 소프트웨어 시장 환경에서 신속히 취약점을 탐지하기 위해 자동화 처리 능력이 필요하다. 취약점을 탐지하기 위한 정적분석 기반 자동화 탐지 도구가 존재하지만 대부분 특정한 취약점 유형에만 탐지할 수 있기 때문에 실용성이 제한되어 있다.

최근 몇 년간에 인공지능 기술이 많이 발전하였다. 알파고(Alpha go)로부터 영상처리 및 자연어처리 분야를 넘어 여러 가지 분야에서 많은 연구가 진행되었다[2]. 일부 문제에서는 인공지능 기술이 사람보다 빠르고 정확한 문제 해결 능력을 보여준다.

본 논문에서 해결할 취약점 탐지 문제를 인공지능 입장에서 정의하자면 특정한 순서 및 특정한 유형의 명령어들이 존재하면 해당 코드들이 취약한 코드인지 안전한 코드인지 판단하는 것이다. 예를 들어, Use-after-free 취약점은 메모리공간을 가리키는 포인터가 free 된 후 다시 해당 포인터를 통해 메모리를 읽거나 쓰는 행위 존재하면 해당 코드가 취약하다고 판단한다. 이러한 취약점을 탐지하려면 malloc 함수호출, free 함수호출, 포인터 사용명령어 순서로 취약한 코드를 찾을 수 있다. 물론 더욱 복잡한 메모리 분석이 필요한 취약점 유형도 존재하지만 본 논문의 연구범위가 아니고 특정 유형 취약점 탐지의 정확성 향상 문제는 향후 연구로 진행할 예정이다.

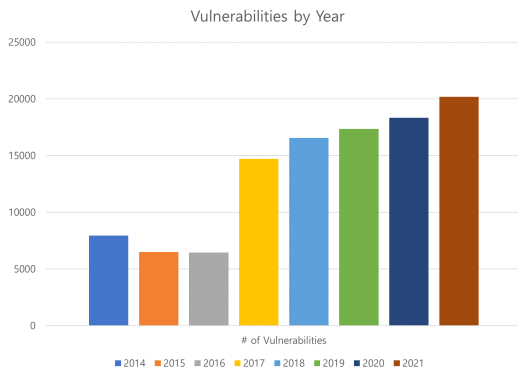

Fig. 1. Reported CVE vulnerabilities by year[1]

최근에는 취약점 탐지 분야에서 연구자들은 기존 문제에 기계학습 방법을 적용하여 더 좋은 결과를 얻기 위한 많은 시도를 하고 있다[3-6]. 예를 들어, CNN 모델은 이미지 처리에 적합한 모델이며 이미지 중에서 눈으로 구분하지 못하는 특징을 정확하게 구분할 수 있다[7]. 이 때문에 악성코드 탐지 연구에서 CNN 모델을 적용하여 코드를 이미지 형태로 처리하여 87% 정확도를 달성한 연구 결과가 존재한다[4]. 이외에도 코드 간 유사도 검사, 패치 코드 검출, 취약점 탐지 문제의 경우에도 인공지능 방법은 좋은 결과를 보였다.

기존 많은 연구에서 취약점 식별 문제에 대해 RNN 기반 모델을 사용하여 해결한다[8]. RNN 모델은 토큰 시퀀스 형태의 입력을 받아 시퀀스에 있는 토큰을 순차적으로 처리하여 학습한다. 그러나 RNN은 현재 입력한 토큰의 출력이 다음 입력할 토큰과 함께 학습되는 모델이다. 토큰이 시퀀스에 있는 위치에 따라 최종 출력까지 학습 과정에 참여하는 횟수가 다르다. RNN의 장점은 데이터 간 의존 관계를 구분할 수 있고 같은 토큰들로 구성된 입력이라 하더라도 데이터의 순서가 다르면 최종 출력된 결과도 다르다. 프로그래밍 코드도 이러한 특징이 있다.

프로그래밍 코드의 경우에도 분기 문과 같은 다양한 구조가 존재하지만, 기본적으로는 코드가 함수 내에서 순차적으로 실행하는 구조다. 이러한 구조적인 특성 때문에 RNN 모델에서는 연속 입력된 명령어 시퀀스로 유효한 의미를 학습할 수 있다.

Vuldeepecker는 RNN 기반 모델을 이용하여 취약점을 탐지하는 대표적인 연구다. 그러나 RNN 모델은 먼저 입력된 토큰보다 뒤에서 입력된 토큰이 최종 출력에 더 큰 영향을 주는 문제점이 있다. 실험 과정에서 SARD 데이터 셋에서 실제로 추출한 program slice인 경우 입력의 길이 2000개 토큰

Table 1. Comparison of RNN, Transformer, BERT model characteristics

Model category	Embedding	Effect of Input Length	Bi-directional context understanding	Next sentence recognition	Parallel computation
RNN	static embedding	relevant	forward direction	No	No
Transformer	static embedding	no relevant	Bi-directional	No	Yes
BERT	context depended	no relevant	Bi-directional	Yes	Yes

을 넘는 경우도 있었다. 따라서 코드의 양이 방대할수록 초기에 처리된 코드의 특징값이 중간에 있는 상당한 양의 데이터 연산 때문에 희석되어 뒤에 있는 코드와의 관계에 반영되지 않고 출력에도 반영되지 않는다. 입력이 너무 길어지면 앞에 있는 코드에는 학습효과가 없는 것이다. 또한 RNN 기반 모델은 입력의 토큰을 순서대로 처리하기 때문에 병열 처리할 수 가 없다. 따라서 실행 효율이 입력의 크기에 따리 좋지 않는 문제점도 있다.

Transformer 모델은 RNN과 달리 시퀀스의 각 토큰이 자신과 시퀀스 내 다른 토큰과의 관계를 계산함으로써 RNN 모델에서 나타나는 장거리 의존성 손실 문제를 해결한다[9]. 하지만 이러한 계산 방식은 RNN 모델에서 토큰이 시퀀스에 존재하는 위치 특징을 학습에 반영할 수 있는 장점을 반영하지 못하는 한계점이 있다. 이를 극복하기 위해 Positional encoding을 추가하여 보완하였다.

BERT 모델은 자연어를 처리하기 위해 설계된 pre-training 방식 모델이다[10]. Table 1. 표와 같이 BERT의 주요한 특징은 다음과 같다. 1) BERT는 Transformer의 Encoder 부분을 이용하여 단어가 문맥의 모든 단어 간의 관계를 모두 계산함으로써 RNN 모델의 입력 길이로 인한 문제를 해결한다. 2) 기존 모델 대부분의 임베딩 방법은 Word2vec과 같은 임베딩 방법을 통해 미리 학습하여 고정된 임베딩 값을 사용한다. BERT에서는 Masked Language Modeling (MLM) 방법을 사용하여 임의의 단어를 "MASK"로 가리고 추측하게 학습한다. 따라서 학습된 단어의 임베딩은 앞뒤 문맥을 고려하여 정확하게 학습하게 한다. 즉, 같은 단어더라도 서로 다른 문맥에서 가진 임베딩 다르며 문맥에 더 맞는 임베딩이 학습된다. 3) Next Sentence Prediction (NSP) 방법을 이용하여 두 문장 간의 순서도 학습한다. 따라서 BERT는 단어 간의 관계뿐만 아니라 문장 간 관계도 학습하기 때문에 더욱 정확하게 전체 입력을 이해할 수 있다. 이를 바탕으로 예측 정확도에서 더 높게 나왔다고 검증되었다[10].

BERT 모델의 단어 간의 양방향 관계 이해 능력과 문장 간의 관계 이해 능력 모두 본 논문에서 해결하고자 하는 취약점 탐지 문제와 적합하다. 그러나 기존의 BERT 모델을 그대로 취약점 탐지 문제에 적용할 수 없으며 취약점을 탐지하기 위해 몇 가지 추가 처리할 사항이 있다.

프로그램 코드 중 모든 코드가 취약점과 관련된 것 아니라 일부 코드만 취약점과 관련되어 있다. Fig. 2.에서 나와 있는 것처럼 Double Free 취약점을 탐지하려면 포인터를 free 하는 코드가 추적해야 할 주요한 코드이며 같은 포인터에 2번 연속 free 하게 되면 취약하다고 판단한다. 기존의 취약점 탐지 방법에서는 program slicing 기술로 이용해서 특정 행위와 관련된 코드로부터 def-use chain을 만들어 해당 값과 관련된 할당 및 사용하는 명령어를 수집한다. 이는 분석 범위를 줄여 더욱 정확하고 효율적으로 문제를 풀 수 있다[11]. 본 논문에서도 소스코드를 program slicing 방법을 적용하여 위험한 라이브러리 함수호출, 포인터 참조와 관련된 코드 조각들을 수집하여 이를 입력으로 사용한다.

본 논문에서는 BERT 모델을 사용하지만 수행할 문제의 특성에 따라 fine-tune 방법과 feature extract 방법으로 구분할 수 있다. Fine-tune 방법은 단어와 문장 간의 관계를 잘 학습된 모델에 출력하기 위한 softmax 층을 추가하여 예측하거나 decoder 층을 추가하여 시퀀스를 출력한다. Single Sequence Classification Task이나 Question Answering Task 모두 이러한 문제이다. 이 방법은 문제를 학습하는 과정에서 pre-training 레이어의 파라미터도 같이 학습되고 변경한다. Feature extract 방법은 pre-training 단계에서 학습된 모델을 다시 학습하지 않고 출력한 임베딩을 실제 풀어야 할 문제의 입력으로 사용하여 새로운 문제를 풀기 위한 모델로 학습한다.

본 본 논문에서는 fine-tuning 방법을 사용하고

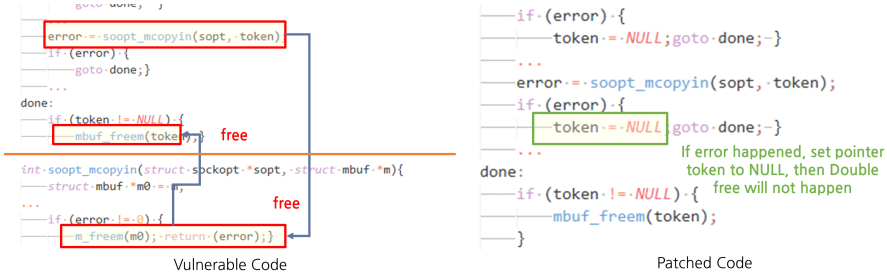

Fig. 2. CVE-2017-13867 Double Free vulnerable code and patched code

기존 BERT pre-training 모델에 FC 층 및 soft max 층을 추가하여 예측값을 출력하고 취약 여부를 판단한다. 실험 결과 취약점 탐지의 정확성은 97.5%로 기존 vuldeepecker 보다 1.5% 개선하였고 탐지 효율성 면에서는 69% 시간을 절약했다. 본 논문에서는 BERT 모델을 이용해서 복잡하지 않은 코드를 대상으로 기존의 data dependency 분석과 취약점 패턴을 이용하여 취약점을 식별한다. 실험 결과 높은 정확성과 자동화 처리하는 능력을 검증하였고 인공지능을 이용해서 취약점을 탐지하는 연구 분야에서 의미 있는 연구를 진행하였다.

II. 배경 지식

2.1 소스코드 취약점 탐지 문제 설명

취약점은 종류가 매우 다양하다. 실제 소프트웨어에서 많이 발견된 취약점은 13가지 유형으로 분류되며 세부적인 취약점 유형은 CWE 사이트에서 정의한 종류의 수에 따르면 총 1332가지 취약점의 유형이 존재한다[12].

프로그램 개발자들은 개발해야 할 기능에만 집중하여 안전성에 대한 고려를 많이 하지 못한다. 또한 보안 전문가가 아니기 때문에 수많은 취약점에 대한 지식을 가지고 있지 않다. 기존의 취약점을 탐지하는 방법은 심각하고 흔히 발견되는 몇 가지 취약점 패턴을 기반으로 정적 분석 방법으로 제어 흐름, 데이터 흐름을 분석하고 취약한 코드 패턴을 찾는 방식으로 탐지한다. 따라서 많은 작업량과 시간이 필요할뿐더러 취약점과 관련된 지식과 경험을 가져야만 탐지할 수 있다.

Fig. 2.은 실제 소프트웨어에서 발견된 Double Free 취약점과 관련된 코드이다. 왼쪽은 취약한 코드이며 오른쪽은 패치된 코드이다. 왼쪽 상단에 있는 코드에서 먼저 soopt_mcopyin(sopt, token) 함수를 호출한다. 그리고 포인터 token이 Null 아니면 mbuf_freem(token)함수 호출하여 해당 포인터가 가리킨 메모리 공간을 free한다. 하지만 token 포인터는 여전에 메모리 공간의 주소를 저장하고 있고 메모리 공간을 가리키고 있다. 실행순서에 따라 soopt_mcopyin 함수 실행 후 token이 Null아니기 때문에 mbuf_freem함수를 호출하여 이미 free된 포인터 token에 다시 free함수를 호출하여 취약점이 발생한다. 패치된 코드를 살펴보면 soopt_mcopyin 함수 호출한 후 soopt_mcopyin함수에서 해당 포인터를 free했으면 포인터 token에 Null 값을 넣어서 다시 free함수 호출 못하게 취약 위협을 보완하였다.

현대 소프트웨어는 복잡도와 코드의 양이 많기 때문에 이미 알려진 취약점을 찾는 경우 이외에는 특정한 목표 없이 취약한 코드를 찾는 것은 매우 어렵다. 이러한 문제를 해결하기 위해 cwe_checker와 같은 정적 자동화 취약점 탐지 기술도 개발되었다. 그러나 취약점 유형마다 특징이 서로 다르기 때문에 취약점 유형별로 탐지 기법을 따로 개발해야 하고 정적 분석의 한계점 때문에 찾지 못하는 취약점 유형도 존재한다.

Fuzzing과 같은 동적 자동화 탐지 기술은 특정한 취약점을 찾는 것이 아니라 임의의 입력을 생성하여 프로그램을 반복적으로 동작시킨다[13-19]. 프로그램 내에 존재하는 모든 실행경로를 탐색하면서 오류가 발생할 때까지 실행한다. 오류를 발견하면 수동으로 오류의 원인을 찾아 문제를 해결하는 방법이다.

Fuzzing은 효율성이 좋지 않지만 탐지의 정확성이 높고 자동화의 정도도 상대적으로 높기 때문에 전문가의 분석 시간을 줄이는 효과가 있다. 그러나 fuzzing 기술도 한계점이 있다. 입력을 무작위로 생성해서 프로그램의 모든 경로 탐색하는 것 매우 어렵고 많은 시도가 필요하다. 또한 fuzzing은 프로그램

자체가 가진 취약한 코드를 찾아낼 수 있지만 외부에서 코드를 주입하여 프로그램 본래의 실행경로가 달라져 생긴 취약한 코드는 탐지할 수 없다. 예를 들면, 중요한 포인터를 사용할 때 boundary check는 다른 함수에서 사용되는 함수에서는 검사하지 않고 역참조하는 코드가 존재하는 경우 정적분석 방법만이 위험한 패턴을 찾아낼 수 있다.

본 논문에서 인공지능을 이용하여 취약점을 탐지하며, 연구의 목표는 기존의 취약점을 분석할 때 정적분, 동적분석 방법의 단점을 극복하여 인공지능 모델이 스스로 많은 종류의 취약한 패턴을 학습하여 탐지하는 것을 목표로 연구를 수행하였다.

2.2 Word Embedding

인공지능 모델 내부에서는 수식 계산을 하므로 입력은 숫자만 가능하다. 이미지를 처리할 때는 이미지의 픽셀값을 처리해서 모델에 입력한다. 자연어처리에서는 처리할 대상이 텍스트 유형이기 때문에 텍스트를 먼저 숫자로 변환한 후 모델에 입력해서 학습할 수 있다. 인공지능 분야에서 각 단어를 다차원 벡터로 바꾸는 방법 word embedding이라고 한다.

One-hot encoding은 가장 기본적인 word embedding 방법이다. Fig. 3.예시처럼 변환될 벡터의 차원 수를 학습데이터 중에 단어의 종류와 일치하게 설정하고 단어를 변환할 때 자신에 할당된 위치는 1로 나머지 위치에는 모두 0으로 설정된다. 따라서 각 단어 간 유사성을 계산하면 모두 0이기 때문에 비슷한 의미를 가진 단어를 구분하지 못한다.

2013년 구글에서 발표한 Word2vec[20]으로부터 Glove[21], FastText[22] 등 많은 word embedding 모델이 연구되었다. 핵심 아이디어는 계산할 수 없는 단어의 의미를 벡터로 표현하여 단어 간의 관계를 계산할 수 있게 한다. 잘 학습된 모델에

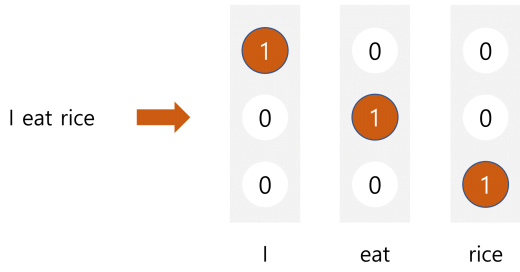

Fig. 3. One-hot encoding example

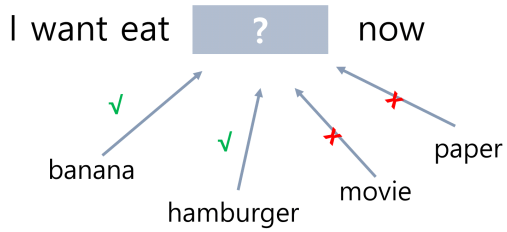

Fig. 4. Word2vec training

서는 같은 의미를 혹은 특징을 가진 단어의 벡터값이 비슷하게 계산되고 이러한 학습된 특징들은 벡터의 각 차원에 녹여져 있다. 이해하기 쉬운 예로, 남과 여를 의미하는 벡터의 차원, 동사와 명사를 의미 하는 벡터의 차원, 식품과 생활용품을 의미하는 벡터의 차원과 같이 계산할 수 있다.

Word2vec의 아이디어는 Fig. 4.와 같이 비슷한 문맥을 가진 단어는 비슷한 의미를 가지고 있다. 인공지능 방법으로 문맥이 주어졌을 때 다음 단어가 무엇인지 예측하는 문제를 풀게 한다. 이 과정을 학습하면서 유사한 단어들은 유사한 벡터 값을 가지게 된다. 기존 연구에서는 word2vec 모델로 입력을 벡터로 변환해서 학습했을 때 기존 one-hot encoding보다 뛰어난 정확성을 보였다[8].

2.3 Bi-directional RNN

서론에서 서술한 것과 같이 RNN 모델의 입력은 시퀀스로 되어 있고 네트워크 형태가 시퀀스 중의 데이터를 한 개씩 처리한다. 앞에서 연산된 출력값이 다음 차례의 입력 데이터와 함께 연산에 포함된다. 따라서 시퀀스 뒤에 있는 데이터의 출력은 자신의 값뿐 만 아니라 앞에 있는 데이터의 정보 모두 담겨 있다. 이러한 네트워크 구조에서 입력 값은 자신 앞에 어떤 데이터가 있는지에 따라 달라지고 그에 따라 출력값이 달라지지만 자신 보다 뒤에 있는 데이터의 정보는 담겨 있지 않다. 앞과 뒤의 문맥을 모두 필요한 일부 문제에서는 적합하지 않아 Fig. 5.처럼 역방향으로 된 레이어를 한 개 더 추가하여 데이터 앞에 있는 문맥과 뒤에 있는 문맥을 모두 반영된 모델이 Bi-directional RNN 모델이다.

III. 관련 연구

자연어처리 분야에서 인공지능의 활발한 발전에

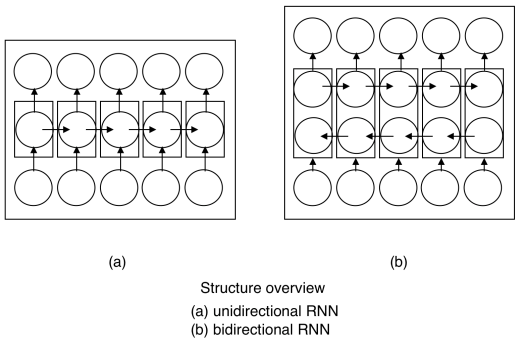

Fig. 5. RNN (left) and Bi-directional RNN (right)

따라 프로그래밍 코드 분석 분야에서도 많은 연구가 진행되어왔다. 예를 들어, 코드 유사도 검사, Stripped 바이너리에서 함수명 예측, 바이너리 디컴파일, 취약점 탐지 등 문제 모두 인공지능 방법이 적합하여 좋은 정확성을 보여줬다. 인공지능 기반 프로그램 방법에 가장 중요한 부분은 수행하고자 하는 문제에 맞게 코드를 표현하는 것이다. 기본적으로 토큰 시퀀스로 표현하는 방식과 이미지로 표현하는 두 가지 분류가 있다. 본 장에서는 기존 연구에서 코드 표현 방법에 대한 연구 동향을 분석하고 논문에서 제시하는 연구 방향에 대한 차별성을 비교한다.

3.1 구조적 특징 기반 악성코드 식별

기존의 시그니처 기반 탐지 방법에 대응하기 위해 새롭게 변조된 버전이 지속해서 나오고 있다. 보안 업체에서는 새로 보고된 악성코드의 특징을 분석하고 악성코드 시그니처 데이터베이스 갱신해야 하는 문제가 있다. 인공지능 기술이 영상처리 분야에서 성공한 후 연구자들이 악성코드 식별 문제를 비슷한 방식으로 많은 시도를 해봤다. Saxe et al.이 제안한 방법에서는 바이너리 코드를 고유한 텍스처 기반 이미지로 표시하였다[23]. 이러한 텍스처 기반 이미지 분석의 장점은 전체 CNN 계열 모델을 사용하여 코드 구조를 유지하면서 작은 코드 변경 사항을 표시하여 악성 코드의 구조에 대한 더 많은 정보를 제공할 수 있다는 것이다. 텍스처 기반 악성코드 이미지 분석의 주요 제한 사항은 특정 악성코드 난독화를 쉽게 분석할 수 없다는 것이다. 악성코드 탐지 문제뿐만 아니라 특정한 코드 패턴을 이용하여 해결하는 문제들은 코드를 텍스처 기반 이미지 분석으로 해결하는 것이 적절한 방법으로 볼 수 있다.

3.2 인공지능 기반 바이너리 역공학

디컴파일, stripped 바이너리에서는 함수 바운더리 회복, 함수명 회복과 같은 역공학 문제가 프로그램 보안 분야에서 다루는 중요한 문제이다. 예를 들어, 소스코드 없는 경우 바이러스 바이너리 코드를 디컴파일해서 바이러스의 행위를 이해한다. Fu et al.에서는 프로그램 코드를 AST(Abstract Syntax Tree)로 분석한 후 AST의 노드 시퀀스를 seq2seq 모델에 입력하여 학습하는 디컴파일 방법을 제안하였다[30]. Fig. 6.은 AST의 예시이다. AST를 이용하면 코드의 문법적인 구조 특징을 주로 학습하면서 코드에 필요하지 않은 noise 정보를 제거하고 일반적인 토큰의 시퀀스를 학습하는 것이 보다 정확성이 좋았다. Alon et al.이 발표한 논문에서는 AST를 이용하여 입력을 생성하여 함수명 정보가 없는 바이너리에서 함수명을 예측하는 문제를 풀었다[28]. 이 외에도 취약점 탐지 혹은 코드 유사성 검사와 같은 문법적인 구조적 특징이 필요한 경우에도 AST가 입력으로 많이 사용되어 프로그램을 표현한다[24-27].

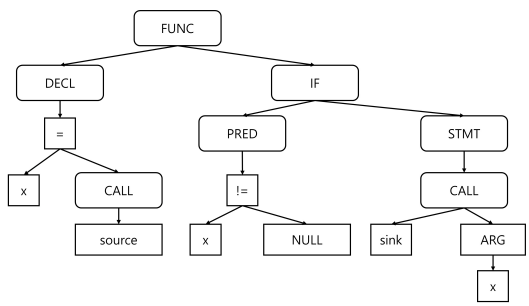

Fig. 6. AST tree

3.3 인공지능 기반 취약점 탐지

최근에 취약점 탐지 관련 분야에서도 인공지능 방법을 많이 사용하고 있다. 취약점이 존재하는 코드의 경우 패치 코드와 비교하면 큰 차이가 없고 대부분 몇 줄만 수정해서 취약점을 패치를 한다. 따라서 취약점 탐지의 문제는 구조적 차이만 학습하는 것은 부족하다. 따라서 CFG(Control Flow Graph), DFG(Data Flow Graph), PDG(Program

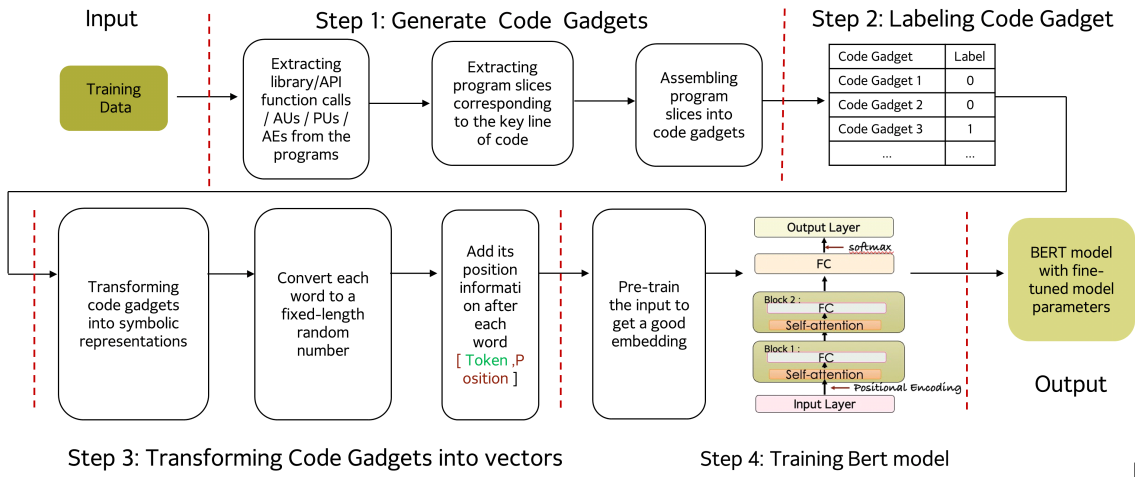

Fig. 7. The overview of proposed approach

Dependency Graph) 등 프로그램의 구조적 특징과 의미적 특징을 가진 형태로 입력을 임베딩하여 취약점을 탐지하여야 한다. Cao et al.이 진행한 연구에서는 코드의 문법적인 구조, context와 값의 흐름을 특징으로 이를 추출하기 위해 AST, CFG, DFG 먼저 생성한 후 한 graph로 융합한다[29]. 따라서 많은 특징이 반영된 입력을 GNN 모델에 사용하여 취약점을 탐지하는 모델을 학습하였다.

프로그램에서 취약점 탐지는 모든 코드를 이해하고 탐지하는 것이 아니고 프로그램 실행 순서에 따라 특정한 행위(명령어 조합)로 해당 코드 취약한지를 판단한다. 관련 없는 코드들이 많을수록 오히려 판단에 영향을 주어 정확성이 높지 않을 수도 있다. Vuldeepecker에서는 무관한 코드의 영향을 줄이기 위해 program slicing 기술을 사용하여 취약점 판단에 필요한 코드만 남겨 RNN 기반 모델에 학습하게 하여 좋은 결과 가졌다. 본 논문에서는 코드 전처리 과정을 program slicing 방법을 이용하여 BERT 모델을 이용하여 RNN 모델로 인한 한계점을 극복하여 보다 더 좋은 결과를 얻었다.

IV. 제안 방법

본 연구에서 제시하는 BERT 모델을 이용한 딥러닝 기반 소스코드 취약점 탐지 방법의 전체 실행 절차는 다음과 같다: 1) 소스코드에 program slicing 기술을 적용하여 중요한 데이터 흐름과 관련된 코드만 남겨두고 무관한 코드를 모두 삭제한다. 2) Slicing 처리한 후 tokenization 절차를 통해 코드 형태에서 토큰 list로 변환한다. 3) Pre-train단계에서 bert 모델 이용해서 라벨 없이 데이터의 임베딩을 학습한다. 4) Fine-tuning 단계에서는 학습된 모델 weight 값을 초기값으로 취약점 탐지를 계속 학습한다. 전체적 구조와 절차는 Fig. 7.에 있다.

4.1 소스코드 전처리

인공지능을 이용해서 소스코드를 학습하기 위해서는 먼저 처리할 데이터의 기본 단위와 입력 시퀀스의 기본 단위를 정해야 한다. 소스코드 중 한 토큰을 기본 단위로 정하면 각 토큰에 대해 임베딩하여 벡터로 변환한다. 만약 토큰에 있는 캐릭터를 기본 단위로 정하면 각 토큰이 한 벡터로 표현하는 것이 아니라 캐릭터 벡터의 시퀀스로 표현한다. 이외에도 statement, function 등으로 기본 데이터 단위로 설정할 수 있지만 대부분 경우에서는 하지 않는다. 본 연구에서는 토큰을 기본 데이터 단위로 설정하였다.

입력 시퀀스의 단위는 학습할 때 모델에 입력되는 전체 데이터의 단위를 말한다. 예를 들면 함수 간 유사성 비교 문제에는 입력의 단위를 함수 단위로 학습해야 한다. 본 논문에서는 소스코드에서 데이터 흐름 분석을 사용하여 program slicing을 추출한다. 이를 입력 시퀀스의 기본 단위로 정했다.

Program slicing 기술은 Weiser, Mark가 최초로 제시한 방법이며 특정한 코드로부터 backward 나 forward 방향으로 해당 명령어에 사

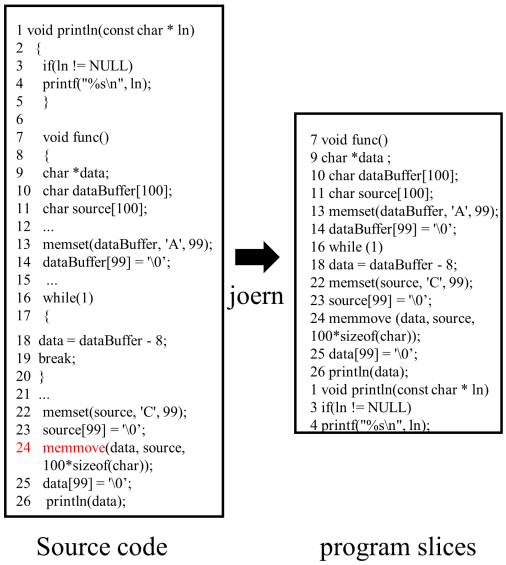

Fig. 8. Program slicing example

용된 변수 값의 유래 및 사용되는 코드를 추적하는 방법이다[31]. Fig. 8.은 소스코드를 slicing 한 예시다. 예시 중 line 24에 있는 "memmove(data, source, 100*sizeof(char))" 명령어를 시작 지점으로 설정한다면 해당 명령어에 사용된 변수인 data와 source의 마지막으로 값을 받은 코드로 추적한다. 이러한 방식으로 최초의 값을 받은 코드까지 추적하여 해당 변수에 영향을 주는 모든 코드만 남겨두고 나머지 코드는 모두 지운다.

program slices는 취약점을 탐지하기 위해 프로그램에서 추출한 중요한 정보이다. 실제로 정적분석에서도 program slicing 기술을 사용해서 취약점을 탐지하며 본질적인 원리는 동일하다. 본 연구에서는 program slicing 추출을 위해 joern에서 제공하는 기능을 이용하였다[32].

추출한 program slicing에서 토큰화(tokenization)를 수행하여 각 keyword, 변수, opcode 등을 모두 토큰 단위로 코드의 순서대로 리스트에 추가한다. 이는 인공지능 모델에 들어갈 입력의 최초의 형태다. 데이터 셋에서 추출한 모든 토큰을 한 번씩 읽어 토큰 사전을 만들고 사전에 각 토큰은 유일한 index 값과 1대1 매칭이 되어 있다.

4.2 BERT 모델 구조

BERT는 기존 RNN+word2vec 방식과 다르다. RNN+word2vec 방식은 먼저 word2vec 모델을 이용하여 각 토큰의 임베딩 값을 학습한 후 고정되어 있다. 취약점 탐지 수행할 때 RNN 모델에 코드를 입력기 전에 해당 임베딩 값을 읽어온 다음에 학습한다.

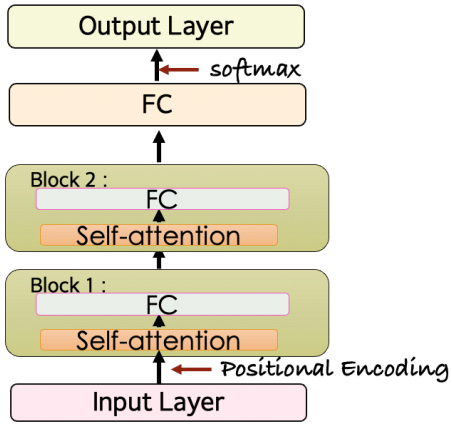

Fig. 9. BERT model structure

BERT는 embedding 과정은 문제해결 과정과 같은 weight를 공유하기 때문에 동시에 학습을 진행한다. 기본적으로 pre-train과 fine-tune 두 단계로 구성되어 있다. Pre-train 단계에서는 라벨 없이 데이터 셋에서 각 데이터의 시멘틱을 학습한다. 그리고 학습된 모델의 weight 값을 그대로 가져와서 취약점 탐지를 수행한다. 각 토큰의 임베딩 값이 고정값이 아니라 시퀀스에 따라 문맥에 맞게 학습되어 더 정확하다.

Positional embedding은 one-hot embed-ding 형태로 되어 있고 토큰이 전체 문맥 중의 위치와 대응한 값만 1로 되어 있다. 이는 토큰이 문맥 중의 위치 특정을 의미한다.

V. 실 험

본 논문에서 제안한 방법의 성능과 비교할 대상을 Bi-directional LSTM + word2vec 방식을 사용하는 vuldeepecker를 선택하여 동일한 실험환경에서 실험하였다. 본 논문에서 제안하는 방법의 성능을 평가하기 위해 NIST에서 제공하는 Juliet Test Suite C/C++를 사용하였다[33]. Juliet Test Suite는 다양한 취약점 CWE 유형의 데이터를 제공하고 코드 중 빈번하게 발생하는 취약점 코드 패턴이 포함된 데이터 셋이다. 총 프로그램의 수는

Table 2. CWE IDs included in the dataset

CWE IDs									
CWE 114	CWE 121	CWE 122	CWE 123	CWE 124	CWE 126	CWE 127	CWE 134	CWE 15	CWE 176
CWE 188	CWE 190	CWE 191	CWE 194	CWE 195	CWE 196	CWE 197	CWE 222	CWE 223	CWE 226
CWE 23	CWE 242	CWE 244	CWE 252	CWE 253	CWE 256	CWE 259	CWE 272	CWE 284	CWE 319
CWE 321	CWE 325	CWE 327	CWE 328	CWE 338	CWE 36	CWE 364	CWE 366	CWE 367	CWE 369
CWE 377	CWE 390	CWE 391	CWE 396	CWE 398	CWE 400	CWE 401	CWE 404	CWE 415	CWE 416
CWE 426	CWE 427	CWE 457	CWE 459	CWE 464	CWE 467	CWE 468	CWE 469	CWE 475	CWE 476
CWE 478	CWE 479	CWE 481	CWE 484	CWE 506	CWE 510	CWE 511	CWE 526	CWE 534	CWE 535
CWE 562	CWE 563	CWE 571	CWE 587	CWE 588	CWE 590	CWE 591	CWE 605	CWE 606	CWE 615
CWE 617	CWE 620	CWE 665	CWE 666	CWE 667	CWE 672	CWE 675	CWE 676	CWE 680	CWE 681
CWE 685	CWE 688	CWE 690	CWE 758	CWE 761	CWE 762	CWE 773	CWE 775	CWE 78	CWE 780
CWE 785	CWE 789	CWE 832	CWE 843	CWE 90					

15,591개 이며 C/C++ 소스코드다. 그중에 취약한 샘플은 14,780개이며 취약점이 없는 샘플은 811개이다. 그리고 전체 데이터 셋에 105가지 C WE 유형을 포함하고 있다. 세부 내용은 Table 2.에서 볼 수 있다.

인공지능 모델에 입력할 데이터의 단위가 vuldeepecker와 동일하게 program slice로 정하고 전체 데이터 셋에서 총 611,890개의 slice를 추출했다. 추출한 기준은 취약점과 관련된 Library 함수, array 사용, 포인터 사용과 수식 계산 4가지를 sink로 정해서 추출하였고 각자의 데이터 수는 Table 3.에서 확인할 수 있다.

학습 데이터 및 테스트 데이터를 8:2로 나누어 실험하였다. 정확성을 평가하는 기준은 다음과 같이 4가지로 하였다:

1. accurate rate (A)
2. false-positive rate (FPR),
3. false-negative rate (FNR)
4. F1

Vuldeepecker 논문에서 Bi-LSTM과 Bi-GRU 모델의 정확성을 모두 실험을 하였기 때문에 본 논문에서는 아래 Table 4.에 있는 3개의 모델의 정확성을 비교한다. 기존 연구의 탐지 정확성은 96%로 높은 정확도를 기록하지만, 본 연구에서 제안하는 방법에서는 97.5%로 1.5%의 정확성을 개선하였다.

데이터 셋의 양이 너무 많았기 때문에 본 연구에서는 전체 데이터를 14개의 batch로 나누어 학습하였다. 그리고 총 소비시간의 평균값을 구한 결과를 표 Table 5.에 기록되어 있다. 실행시간이 기존 방법보다 69% 감소하였다. 전체적으로 vuldeepecker과 비교했을 때 BERT 모델을 이용한 취약점 탐지 방법이 정확성 면에서 Bi-directional GRU 모델을 사용하는 방법보다 정확하고 매우 빠르다.

Table 3. The number of program slices for each vulnerability feature.

	Total	Vulnerable	No Vulnerable
Library/API Function	216,690	16,990	199,700
AU	59267	7,651	51,616
PU	332,685	14,972	317,713
AE	8,248	880	7,368
Total	616,890	40,493	576,397

Table 4. The comparison of accuracy (metrics unit: %)

Model	FPR	FNR	A	F1
Bi-LSTM	2	15.7	96.00	84.3
Bi-GRU	2	14.7	96.00	85.8
BERT	2	13.6	97.51	87.2

Table 5. The comparison of efficiency

Model	Training time	Accuracy
Bi-GRU	About 42 min.	96.00%
Transformer	About 13 min.	97.51%

VI. 결 론

본 논문에서는 최근 취약점 탐지 분야의 연구 발전의 진도를 설명하였고 많이 사용되는 몇 가지 취약점 탐지 도구를 소개하였다. 정적 탐지 방법은 자동화 처리를 하지 못하는 문제점과 모든 취약점 유형에

대한 탐지를 지원하지 못하는 문제점이 있다.

동적 탐지 방법은 자동화 탐지가 가능하고 탐지의 정확성이 보장되지만 좋지 못한 한계점이 있다. 그리고 프로그램 내에 존재하는 실행경로만 탐지할 수 있으며, 취약점의 특정 패턴이나 휴리스틱으로 탐지하지 못하는 한계가 있다.

인공지능 기술은 자동화 탐지가 가능하고 많은 취약점 유형에 대한 패턴 학습이 가능하여 오늘날에 방대한 양의 소프트웨어 분석에 대한 좋은 해결책이 될 수 있다. 본 연구에서 제안하고 개발한 탐지 방법은 기존 RNN 기반 모델보다 정확성 면에서 1.5% 개선하였고 효율성 면에서 69% 개선하였다.

정확성이 개선된 원인을 분석하면 BERT에서 사용한 새로운 embedding 방법 때문에 개선된 것으로 분석되고 문맥에 맞게 임베딩을 수행하는 방법이 고정 임베딩보다 좋은 것으로 보인다.

효율성이 개선된 원인은 RNN 기반 모델이 순차적으로 입력을 처리하기 때문에 하나씩 처리해야 하는 문제점이 있었기 때문이며, BERT 같은 경우 각 단어를 전체 문장에 있는 모든 단어와의 관계를 계산하기 때문에 많은 데이터를 동시에 처리가 능하기 때문에 효율성이 개선된 것으로 분석된다.

References

[1] CWE Top 25 2021, "CWEtop25 2021" https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html, Dec. 2022.

[2] Xiao, Xusheng, and Shao Yang, "An image-inspired and cnn-based android malware detection approach," Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 1259-1261, November. 2019

[3] Li, Zhen, et al., "Vuldeepecker: A deep learning-based system for vulnerability detection," Proceedings of the 25th Network and Distributed System Security (NDSS) Symposium, pp. 1-15, Feb. 2018

[4] Qiang, Gao, "Research on Software Vulnerability Detection Method Based on Improved CNN Model." Scientific Programming," vol. 2022, pp. 4442374, Jul. 2022

[5] Wu, Fang, et al., "Vulnerability detection with deep learning," Proceedings of the 3rd IEEE International Conference on Computer and Communications (ICCC), pp. 1298-1302, Dec. 2017

[6] Russell, Rebecca, et al., "Automated vulnerability detection in source code using deep representation learning," Proceedings of the 17th IEEE international conference on machine learning and applications (ICMLA), pp. 757-762. Dec. 2018

[7] Girshick Ross, "Fast r-cnn," Proceedings of the 2015 IEEE International Conference on Computer Vision, pp. 1440-1448, Dec. 2015

[8] Mikolov Tomas, et al., "Recurrent neural network based language model," Jonornal of the Interspeech, vol. 2, no. 3, pp. 1045-1048, Sep. 2010

[9] Vaswani Ashish, "Attention is all you need," Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS), vol. 30, pp. 1-11, Dec. 2017

[10] BERT, "BERT" https://arxiv.org/abs/1810. 04805, Oct. 2018

[11] CWE, "CWE" https://cwe.mitre.org/index. html, Dec. 2022

[12] Fotify, "fotify" https://www.microfocus.com/en-us/cyberres/application-security/static-code-analyzer, Dec. 2022

[13] Checkmarx, "Checkmarx" https://checkma rx.com/, Dec. 2022

[14] AFL, "AFL" https://github.com/google/AFL, Dec. 2022

[15] Böhme Marcel, et al., "Directed greybox fuzzing," Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 2329-2344, Oct. 2017

[16] Gan Shuitao, et al., "Collafl: Path sensitive fuzzing," Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP), pp. 679-696, May. 2018
[17] Rawat Sanjay, et al., "VUzzer: Application-aware Evolutionary Fuzzing," Proceedings of the Network and Distributed System Security (NDSS) Symposium, pp. 1-14, Feb. 2016
[18] Cha, Sang Kil, Maverick Woo, and David Brumley, "Program-adaptive mutational fuzzing," Proceedings of the 2015 IEEE Symposium on Security and Privacy(S&P), pp. 725-741, May. 2017
[19] Ganesh, Vijay, Tim Leek, and Martin Rinard, "Taint-based directed whitebox fuzzing," Proceedings of the 2009 IEEE International Conference on Software Engineering(ICSE), pp. 474-484, May. 2009
[20] Chen, Peng, and Hao Chen, "Angora: Efficient fuzzing by principled search," Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP), pp. 711-725, May. 2015
[21] Lin Guanjun, et al., "Software vulnerability detection using deep neural networks: a survey," Proceedings of the IEEE, vol.108, no. 10, pp. 1825-1848, Jun. 2020
[22] Fan, Ming, et al., "Text backdoor detection using an interpretable rnn abstract model," Jonornal of the IEEE Transactions on Information Forensics and Security, 16(0), pp. 4117-4132, Aug. 2021
[23] Saccente, Nicholas, et al., "Project achilles: A prototype tool for static method-level vulnerability detection of Java source code using a recurrent neural network," Proceedings of the 2019 IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW), pp. 114-121, Nov. 2019
[24] Grieco, Gustavo, et al., "Toward large-scale vulnerability discovery using machine learning," Proceedings of the 6th ACM Conference on Data and Application Security and Privacy, pp. 85-96, Mar. 2016
[25] Wu, Fang, et al., "Vulnerability detection with deep learning," Proceedings of the 3rd IEEE international conference on computer and communications (ICCC). pp. 1298 - 1302, Dec. 2017
[26] Zou, Deqing, Sujuan Wang, Shouhuai Xu, Zhen Li, and Hai Jin, "VulDeePecker: A Deep Learning-Based System for Multiclass Vulnerability Detection," IEEE Transactions on Dependable and Secure Computing, 18(5), pp. 2224-2236, Sep. 2019
[27] Shar, Lwin Khin, and Hee Beng Kuan Tan, "Predicting SQL injection and cross site scripting vulnerabilities through mining input sanitization patterns," Information and Software Technology, vol. 55, no. 10, pp. 1767 - 1780, Oct. 2013
[28] Wang, Song, Taiyue Liu, and Lin Tan, "Automatically learning semantic features for defect prediction," Proceedings of the 2016 IEEE/ACM International Conference on Software Engineering (ICSE), pp. 297 - 308, May. 2016
[29] Lin, Guanjun, et al., "POSTER: Vulnerability discovery with function representation learning from unlabeled projects," Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications

Security, pp. 2539 - 2541, Oct. 2017
〔30〕 G. Lin et al., "Cross-project transfer representation learning for vulnerable function discovery," IEEE Transactions on Industrial Informatics, vol. 14, no. 7, pp. 3289-3297, Jul. 2018
〔31〕 Automatic feature learning for vulner ability prediction, "Automatic feature learning for vulnerability prediction," http://arxiv.org/abs/1708.02368, Dec. 2022
〔32〕 Li, Zhen, et al., "Sysevr: A framework for using deep learning to detect software vulnerabilities," IEEE Transactions on Dependable and Secure Computing, vol. 19, pp. 2244-2258, Aug. 2022
〔33〕 Weiser, Mark. "Program slicing," IEEE Transactions on Software Engineering, vol. 4, pp. 352-357, Jul. 1984
〔34〕 Joern, "joern" https://joern.io/, Dec. 2022

〈저자소개〉

김 문 회 (Wenhui Jin) 학생회원
2013년 9월: HEILONGJIANG UNIVERSITY 소프트웨어공학 학사
2016년 3월~현재: 한양대학교 컴퓨터공학과 석박사통합과정
〈관심분야〉 정보보호, 전자공학, 통신공학

오 희 국 (Heekuck Oh) 종신회원
1982년: 한양대학교 전자공학과 졸업
1989년: 아이오와주립대학 전자계산학과 석사
1992년: 아이오와주립대학 전자계산학과 박사
1993년~1994년: 한국전자통신연구원 선임연구원
1995년 3월~현재: 한양대학교 ERICA 소프트웨어융합대학 교수
〈관심분야〉 암호기술응용, 시스템보안