바이너리 파일에서 Word2Vec 딥러닝 기법을 이용한 복합 자료형 추론 연구

민예슬*, 정현오**, 이혜린**, 이승연**, 우드아커병 오퐁**, 정준호***, 손윤식**
*동국대학교 정보통신공학과
**동국대학교 컴퓨터공학과
***동국대학교 전자상거래연구소
e-mail: sonbug@dongguk.edu

A Study on Composite Data Type Inference using Word2vec Deep Learning Scheme on Binary File

Ye Sul Min*, Hyunoh Jung**, Hyerin Lee**, Sungyeon Lee**,
Junho Jeong***, Yunsik Son**
*Dept. of Information Communication Engineering, Dongguk University
**Dept. of Computer Science and Engineering, Dongguk University
***Electronic Commerce Institute, Dongguk University

요 약

소프트웨어의 보안에 대한 중요성이 점차 높아짐에 따라, 소스코드 기반의 소프트웨어의 보안약점 분석 기법에서 더 나아가 소스 코드가 존재하지 않는 바이너리 파일을 대상으로 분석을 수행하는 연구가 진행되고 있다. 왜냐하면 소프트웨어 개발의 복잡성 증가에 따른 서드파티 라이브러리 활용과 레거시 코드의 관리 부재, 임베디드 소프트웨어의 특성 등으로 인해 소스 코드가 존재하지 않는 바이너리 코드의 사용이 늘어나고 있기 때문이다. 따라서 최근 바이너리 코드에 내제된 보안약점을 분석하기 위해서 중간코드를 이용하여 정적분석을 수행하는 다양한 연구가 진행되고 있다. 중간언어를 사용함으로 실행환경에 따라 달라지는 바이너리 코드가 중간언어로만 변환이 된다면 동일한 형태의 보안약점 분석기술을 통해 효과적인 수행이 가능하다. 본 논문에서는 이러한 바이너리 코드로부터 중간언어로 변환시 컴파일 과정에서 상실된 복합 자료형을 재구성하기 위해 Word2vec 딥러닝 기법을 이용한 추론 기법을 제안한다.

1. 서론

현재 소프트웨어의 보안약점을 분석하기 위한 방법으로, 검출된 결함의 정확도가 높고 회귀 시험에 적합한 동적 분석 기법을 수행하는 것이 일반적이다. 그러나 동적 분석 기법은 명세와 테스트 케이스에 따라 결과의 정확도가 좌우되고, 발생하지 않는 보안 취약점의 경우 실제 프로그램 구동 시 치명적인 보안약점을 초래할 수 있다. 그러므로 정적분석 또한 보안약점 분석기법으로 활용해야 한다. 정적분석 기법은 프로그램의 소스코드를 분석하여 발생할 수 있는 보안약점 및 취약점을 검사하는 기법이기 때문에 소스코드가 존재하는 프로그램에만 제한적으로 사용할 수 있다.

오늘날의 소프트웨어의 경우 그 기능이 다양화, 고도화의 필요성에 따라 이미 개발된 기능은 해당 실행파일 또는 오픈소스 라이브러리를 사용하는 경우가 증가되고 있다[1]. 하지만 이와 같은 기 개발된 소프트웨어는 개발 단계에서 보안검증을 수행하였는지 확인할 수도 없으며 검증을 받지 않았을 확률 또한 높아 다른 소프트웨어와 같이 동작할 경우 심각한 보안약점으로 다양한 보안사고가 발생할 수 있다[2-5].

하지만 이와 같은 소프트웨어는 소스코드가 존재하지 않거나 일부만 존재하는 바이너리 형태의 실행파일일 경우가 대부분이므로 기존의 소스코드를 통한 정적분석 기법으로 보안약점을 검출하는 것이 어렵다.

이러한 이유로 최근에는 바이너리 코드에 대한 보안약점 분석에 중간언어를 활용한 정적 분석기법이 많은 연구가 되고 있다[6-10]. 이 연구들은 바이너리 파일을 어셈블리어로 변환한 후 중간언어로 복원하여 기존의 정적분석 기법을 활용해 보안약점을 분석하는 기술이다. 이 기법들의 가장 중요한 요소는 바이너리 코드로부터 소스코드의 자료형을 추론하는 것이다[11]. 하지만 그 중에서 복합 자료형에 대한 추론은 잘 이루어지지 않는 문제가 있다.

본 논문에서는 소스코드의 복합 자료형을 추론하여 그 멤버 변수를 추론하고 재구성하기 위해서 Word2vec 딥러닝 기법을 이용하였다. 2장에서 관련연구로 Word2vec 에 대해 소개하고, 3장에서는 이를 이용한 복합자료형의 멤버 변수 추론 실험 과정 및 결과를 분석한다. 그리고 4장에서

이 성과는 2017년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2017R1C1B5018257).

결론을 맺는다.

2. Word2vec

또한 최근 주목받는 기술인 기계학습은 많은 양의 데이터를 처리하는 알고리즘이 있어 입력데이터들을 이용하여 학습하고 분석하며 이를 기반으로 판단이나 예측한다.

그 중 Word2vec은 자연어를 처리하는데 유용한 2층으로 구성된 인공신경망으로 텍스트로 이루어진 문장을 이해하는 딥러닝 기법이다[12]. Word2vec은 말뭉치(corpus)를 입력으로 받아 말뭉치내의 단어를 벡터로 표현 하는데, 각 단어의 해당하는 벡터는 말뭉치에서 단어가 가지는 의미나 역할을 잘 표현해주는 값이다. 이와 같이 단어의 의미와 문맥을 이용하여 단어를 벡터로 표현한 것을 word embedding라고 한다[13].

Word2vec의 장점은 단어 간 유사성을 구할 수 있다는 것이다. 원래 word embeddings의 목적이 유사한 단어일수록 가까운 거리에 위치하도록 각 단어에 해당하는 벡터 값을 찾는 것이다. 이 학습기술은 입력으로 받은 말뭉치 데이터만을 이용한다.

Word2vec의 학습 방법은 두 종류로, CBOW (Continous Bag Of Words) 모델과 Skip-gram 모델이 있다. CBOW 방식은 타겟 단어 주변의 단어가 만드는 문맥을 이용해 대상 단어를 예측하는 방식이고, Skip-gram 방식은 한 단어를 기준으로 주변에 올 수 있는 단어를 예측하는 방식이다.

이 중 Skip-gram 모델은 예측하는 단어들의 경우 현재 단어 주위에서 샘플링할 때 근접한 단어일수록 현재 단어와 더 연관이 깊은 단어라는 생각을 적용하기 위해 멀리 떨어져있는 단어일수록 낮은 확률로 선택하는 방법을 사용한다.

3. Word2vec을 이용한 복합 자료형 추론

본 논문에서는 Word2vec 딥러닝 기법을 이용하여 바이너리 코드파일에 존재하는 복합 자료형의 멤버변수 추론 방법을 제안한다. 복원 대상 바이너리 코드는 C/C++로 작성된 잘 알려진 자료구조 코드 중 연결 리스트와, 트리의 바이너리 코드이며 그림 1은 연결 리스트 소스코드이다. 전형적인 키와 링크 데이터를 가진 노드를 포함하고 있다.

전체적인 추론 과정은 크게 두 단계이다. 첫 번째는 (그림 1)과 같이 C/C++ 언어로 연결 리스트, 트리 프로그램을 작성한 후 바이너리 코드를 생성한다. 생성된 바이너리 코드를 다시 역어셈블하여 어셈블리 코드로 변환한다.

두 번째 단계에서 이렇게 변환된 어셈블리 코드를 Word2vec 기반의 타입 추론기의 입력으로 프로그램 내에서 변수로 추정되는 [ebp 레지스터-상수]로 표현되는 주소에 대해 계산된 벡터 값들 얻을 수 있다. 이 벡터 값들을 향후 군집화하여 복합 자료형을 복원할 수 있다.

다음 (그림 2)는 C언어로 작성된 연결리스트 프로그램을 역공학하여 어셈블리 코드로 변환된 결과의 일부이며 어셈블리 코드의 피연산자로 [ebp - 상수]로 표현된 [ebp-0xcc], [ebp-0x8] 등을 볼 수 있는데 이들은 지역 변수로 복합 자료형 추론의 대상이 된다.

```
#include "stdio.h"
#include "stdlib.h"

struct Node {
   int value;
   Node* next;
};
Node *head;

void InitList() { … }
Node * PutNode(Node *target, Node *temp) { … }
bool Delnode(Node * target) { … }

void main(int argc, char *argv[]) {
   Node *target;
   Node temp;
   InitList();
   target = head;
   for (int i = 1; i <= 10; i++) {
      temp.value = i;
      target = PutNode(Target, &Temp);
   }
   for (target = head->next; target; target = target->next) {
      printf("%d\n", target->value);
   }
}
```

(그림 1) 추론 대상 소스코드 - 연결리스트

```
push ebp
mov ebp, esp
sub esp, 0xcc
push ebx
push esi
push edi
lea edi, [ebp-0xcc]
mov ecx, 0x33
mov eax, 0xcccccccc
rep stosd
mov eax, [ebp+0x8]
mov ecx, [eax+0x4]
mov [ebp-0x8], ecx
cmp dword [ebp-0x8], 0x0
jnz 0x31
xor al, al
jmp 0x55
mov eax, [ebp+0x8]
mov ecx, [ebp-0x8]
mov edx, [ecx+0x4]
mov [eax+0x4], edx
mov esi, esp
```

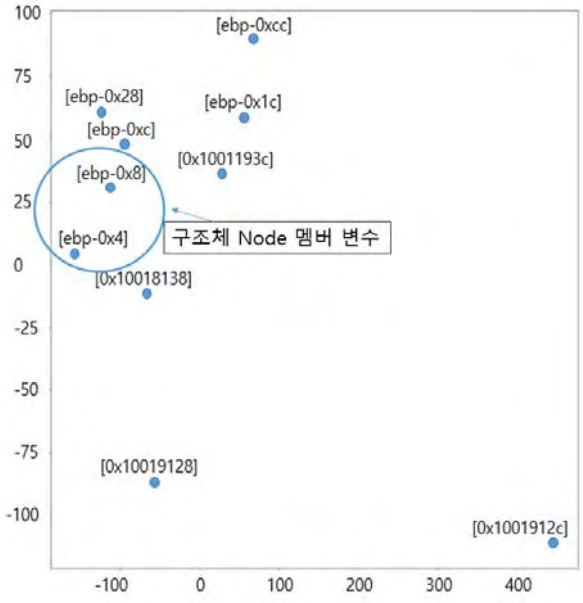
(그림 2) C 언어로 작성된 연결 리스트 어셈블리코드 예

위와 같이 출력된 어셈블리어 텍스트 파일을 Word2vec 기반의 복합 자료형 추론기에 입력한다. 이 자료형 추론기는 입력 파일내의 어셈블리어를 벡터로 매칭한 후 코사인 유사도로 단어 관계의 유사성을 판단하며 학습하고 관련 벡터 값을 최종 결과로 반환한다.

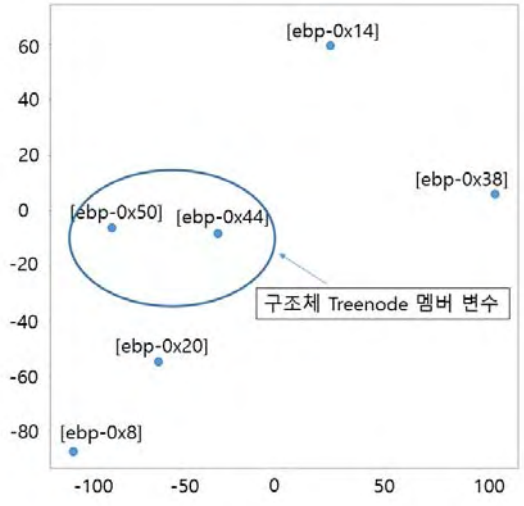
총 10,000 번의 학습이 10 번에 걸쳐 이루어지며 개수가 가장 많은 단어 5개와 가까운 단어 10개를 보여준다. 따라서 최종적으로 신경망은 10 만 번 학습하고, 학습한 결과는 각 단어에 매칭된 벡터가 표시된 좌표가 된다.

다음 (그림 3)은 연결 리스트에 대해서 사전 파악된 추론 대상을 기준으로 2차원 평면으로 표현한 결과이며

[ebp-0x8], [ebp-0x4] 이 군집화 될 수 있으며 실제 연결 리스트 코드에서 Node 구조체의 멤버변수인 것을 확인할 수 있었다.



(그림 3) 복합 자료형 추론 결과 - 연결 리스트



(그림 4) 복합 자료형 추론 결과 - 트리

또한 (그림 4)는 트리 프로그램에 대한 추론 결과로 [ebp-0x50], [ebp-0x44]를 군집으로 볼 수 있으며 트리 프로그램의 실제 코드에서 Treenode 구조체의 멤버변수인 것을 확인 할 수 있었다. 특히 이진트리의 경우 연결리스트보다 구조체의 멤버변수끼리 군집화가 잘 되어있는 것을 확인 할 수 있었다.

따라서 군집화 알고리즘을 잘 적용한다면 Word2vec의 결과를 통해 복합 자료형에 대한 추론이 가능함을 확인할 수 있었다.

4. 결론 및 향후 연구

본 논문에서는 바이너리 파일의 보안약점 분석을 위해서 바이너리 코드를 중간언어의 자료형 정보를 재구성하는 과정에 Word2vec 기법을 이용하여 복합 자료형에 대한 추론을 수행하였다. 해당 연구를 통하여 Word2vec 기법을 통해 복합 자료형의 멤버 변수의 자료형을 추론하는 것이 제한적으로나마 가능함을 확인하였다.

따라서 향후의 복합 자료형 추론의 정확도 및 자동 분류를 위해 더 많은 테스트 케이스를 확보하고 실험을 수행하고자 한다.

참고문헌

[1] Find Defects in Third-Party Code, http://www.grammatech.com/products/binary-analysis

[2] codenomicon, The Heartbleed Bug, http://heartbleed.com/

[3] US-CERT, GNU Bourne-Again Shell 'ShellShock' Vulnerability, https://www.us-cert.gov/ncas/alerts/TA14-268A

[4] Bodo Möller, This POODLE Bites : Expoiting The SSL 3.0 FallBack, google

[5] Nimrod Aviram, DROWN : Breaking TLS using SSL v2, https://drownattack.com

[6] Song, Dawn, et al. "BitBlaze: A new approach to computer security via binary analysis." Information systems security. Springer Berlin Heidelberg, pp. 1-25, 2008.

[7] GRAMMATECH, "Eliminating Vulnerabilities in Third-party Code with Binary Analysis", White Paper http://www.grammatech.com/products/codesonar

[8] Necula, George C., et al. "CIL: Intermediate language and tools for analysis and transformation of C programs," Compiler Construction. Springer Berlin Heidelberg, pp. 209-265, 2002.

[9] T. Dullien, and P. Sebastian, "REIL: A platform-independent intermediate representation of disassembled code for static code analysis," Proceeding of CanSecWest (2009).

[10] S. Cesare, and Y. Xiang. "Wire-A Formal Intermediate Language for Binary Analysis," on IEEE 11th International Conference Trust, Security and Privacy in Computing and Communications (TrustCom), pp. 515-524, 2012.

[11] 민예슬, 정현오, 손윤식, 정준호, 고광만, 오세만,

"중간언어 생성을 위한 바이너리 코드 자료형 및 변수 추론 기술 조사 분석," 정보처리학회 춘계학술발표대회 논문집, 제24권, 제1호, pp.283-286, 2017.

[12] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, & J. Dean, "Distributed representations of words and phrases and their compositionality," In Advances in neural information processing systems, pp. 3111-3119, 2013.

[13] O. Levy & Y. Goldberg, "Dependency-Based Word Embeddings," in ACL, vol. 2, pp. 302-308, 2014.