한국정보전자통신기술학회논문지(jkiiect)'23-6, Vol.16 No.3
논문 23-16-03-161 http://dx.doi.org/10.17661/jkiiect.2023.16.3.161

바이너리 코드 취약점 탐지를 위한 딥러닝 기반 동적 오염 탐지 기술

고광만*

Deep Learning based Dynamic Taint Detection Technique for Binary Code Vulnerability Detection

Kwang-Man Ko*

요 약 최근 바이너리 코드에 대한 신종·변종 해킹이 증가되고 있으며 소스 프로그램에서 악성코드를 탐지하고 공격에 대한 방어 기술의 한계점이 자주 노출되는 상황이다. 바이너리 코드에 대해 머신러닝, 딥러닝 기술을 활용하여 고도화된 소프트웨어 보안 취약점 탐지 기술과 공격에 대한 방어와 대처 능력이 필요하다. 본 논문에서는 바이너리 코드의 실행 경로를 추적(execution trace)하여 동적 오염 정보를 입력한 후 오염 정보를 따른 특징을 기반으로 멀웨어를 그룹핑하는 멀웨어 클러스터링 방법을 제안한다. 멀웨어 취약점 탐지는 3-계층으로 구성한 Few-shot 학습 모델에 적용하여 각 계층의 CPU, GPU에 대해 F1-score를 산출하였다. 학습 과정에서 97~98%의 성능과 테스트 과정에서 80~81% 정도의 탐지 성능을 얻었다.

Abstract In recent years, new and variant hacking of binary codes has increased, and the limitations of techniques for detecting malicious codes in source programs and defending against attacks are often exposed. Advanced software security vulnerability detection technology using machine learning and deep learning technology for binary code and defense and response capabilities against attacks are required. In this paper, we propose a malware clustering method that groups malware based on the characteristics of the taint information after entering dynamic taint information by tracing the execution path of binary code. Malware vulnerability detection was applied to a three-layered Few-shot learning model, and F1-scores were calculated for each layer's CPU and GPU. We obtained 97~98% performance in the learning process and 80~81% detection performance in the test process.

Key Words : Binary Code Analysis, Few-shot Learning, Deep Learning, Software Security

1. 서론

4차 산업혁명 시대에 다양한 분야에서 소프트웨어의 역할과 응용 분야가 넓어지면서 소프트웨어 보안의 중요성이 강조되고 있다. 신종·변종 해킹, 사이버 공격 등에 적극적으로 대처하기 위한 연구와 다양한 기술 개발이 국내외에서 활발하게 진행되고 있다[1]. 일반적으로 프로그램 분석은 소프트웨어의 결함 또는 버그를 찾아 신뢰성 높은 소프트웨어를 생산하기 위한 목적으로 활용되고 있다. 또한, 소프트웨어의 안전성이 매우 중요하게 요구되는 분야에서 발생 가능성은 낮지만 발생될 경우 치명적인 결함을 발생시키는 취약점을 찾을 목적으로 활용되고 있다. 최근에는 소프트웨어의 취약점을 분석하여 외부로부터 위조, 변조와 같은 공격 가능성을 차단할 목적으로 소스 코드 또는 바인너리 코드 수준에서 이루어지고 있다[2].

프로그램 분석은 목적과 환경에 따라 소스 코드 수

This research was supported by Sangji University Research Fund, 2020. And, This work was partially supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT) (No. 2021R1F1A1048903).
*Corresponding Author : Department of Computer Engineering, SangJi University (kkman@sangji.ac.kr)
Received June 15, 2023 Revised June 22, 2023 Accepted June 26, 2023

준과 바이너리 코드 수준에서 연구되어 왔다. 소스 코드 분석은 품질 향상, 치명적 결함 탐지, 보안 취약점 탐지 및 공격 가능성 차단 목적 등에 많은 한계점을 내포하고 있다. 바이너리 코드 분석은 소스 코드 분석에 비해 직접적이고 명확하게 분석 목적을 이룰 수 있다. 하지만 분석의 어려움과 기술의 한계에도 불구하고 최근에는 인공지능 기반의 도전 연구가 진행되고 있다. 또한, 소프트웨어 보안의 중요성이 증가되면서 실제로 실행되는 바이너리 코드 분석을 통해 보안 취약점 탐지, 공격 가능성에 대한 방어 목적으로 정확하고 고도화된 바이너리 코드 분석 프레임워크에 대한 수요와 필요성이 지속적으로 제기되고 있다[3].

최근에는 바이너리 코드에 대한 신종·변종 해킹 사례의 증가와 시큐어 코딩 등을 통해 소스 프로그램에서 악성코드를 탐지하고 공격에 대한 방어 기술에 대한 한계점이 자주 노출되는 상황이다. 소프트웨어 보안을 위해 바이너리 코드에 대해 머신러닝, 딥러닝 기술을 적용하여 고도화되고 지능화된 소프트웨어 보안 취약점 탐지, 사이버 공격 방어 또는 대처 능력 등이 필요한 상황이다[4].

본 논문에서는 바이너리 코드를 분석한 후 인공지능 기반으로 취약점을 탐지하는 연구가 시도되고 있는 상황에서, 바이너리 코드의 실행 경로를 추적(execution trace)하여 동적 오염 정보를 입력한 후 오염 정보를 따른 특징을 기반으로 멀웨어를 그룹핑하는 멀웨어 클러스터링 방법을 제안한다. 이 과정에서 최초 바이너리 데이터-셋(malware sample)으로부터 활성 학습과 점진 학습을 통해 지속적으로 신뢰할 수 있는 바이너리 코드 학습 데이터-셋의 구축은 전체적인 학습 모델을 구축하는 과정에서 매우 중요한 의미를 가지고 있다.

2. 연구배경 및 관련연구

2.1 바이너리 코드 취약점 탐지

바이너리 코드 분석은 리버스 엔지니어링을 통해 인간이 이해할 수 없는 실행파일을 저수준 언어로 변환한 후 정적 또는 동적 분석 방법으로 의미를 파악한다. 바이너리 코드에 대한 정적 분석은 실제 바이너리 실행 없이 바이너리가 가지고 있는 정보만을 이용하여 분석하기 위해 어셈블리어 또는 저수준 중간 언어로 변환한 후에 원래 프로그램의 의미를 분석하는 것으로 IDA Pro[5]가 국내외에서 대표적으로 활용되고 있다. 동적 분석은 바이너리 코드를 실제로 실행하면서 다양한 입력값을 사용하여 보다 구체적이고 정확한 분석 목적으로 활용되고 있지만 실행시간이라는 오버헤드를 가지고 있다.

최근에는 바이너리 파일에 대해 무작위 입력 값을 생성하여 실행해 보고 비정상적인 프로그램 동작을 발생시키는 입력을 찾아 프로그램의 동작을 분석하는 퍼징(fuzzing) 방법이 주목받고 있다. 퍼징은 소프트웨어 테스트 기법으로 프로그램에 대해 대용량의 데이터를 입력하여 프로그램의 충돌이 발생하는 취약점 위치를 찾아내는 방법으로, 소프트웨어나 컴퓨터 시스템들의 보안 취약점을 파악하고 정의되지 않을 영역을 검증하기 위해 사용되었다. 바이너리 프로그램 테스트에서 퍼징은 블랙박스, 그레이박스, 화이트 박스 퍼징으로 구분된다[6].

취약점 탐지를 위한 오염(taint) 분석은 특정 변수에 오염된 값을 할당한 후 이 값을 이용하는 부분을 오염된 값의 영향을 받은 부분으로 판단한다. 정적 분석 기반 오염 분석은 바이너리를 리버스 엔지니어링을 통해 저수준의 중간 코드로 변환하여 의미 정보를 기반으로 오염 분석을 수행한다. 동적 오염 분석은 바이너리 코드를 메모리에 로드한 후 실행을 통해 메모리상에서 오염된 값의 사용이 진행되는 경로를 추적하여 오염 분석을 진행한다[7].

2.2 관련 연구 및 연구 동기

바이너리 코드에 대한 분석 연구는 오픈소스 분석 플랫폼인 Angr[8], Ghirda[9]가 활용되고 있다. Kasperskey 랩과 Nvidia도 머신러닝 기반 학습모델을 기반으로 분석 기능을 보완하고 다양한 테스트를 통해 정적 및 동적 바이너리 프로그램 자동 분석 핵심 모듈을 개발하였다. 또한, 정상 코드와 악성 코드가 포함된 파일을 학습시킨 후 학습모델을 통해 악성 여부를 판단하는 검증 연구 성과를 제시하였다.

퍼징에서 무작위로 많은 테스트 데이터를 생성하는

진화 알고리즘을 개선하기 위해 인공지능 기술을 이용하여 입출력 테스트 데이터간의 관계를 파악하는 Neuzz[6] 연구, 퍼징에 효과적인 입력 데이터를 생성하기 위해 인공 신경망을 이용하는 연구[10]가 진행되었다.

본 논문은 바이너리 코드에서 동적 오염 분석 기술을 응용하여 유사한 오염 정보에 대해 비정상 행위의 특징을 그룹화한 후 학습 모델을 통해 멀웨어를 클러스터링하는 목적을 가지고 있다. 이러한 연구는 소스 코드 수준에서 오염을 탐지하고 오염 정보의 특징에 따라 클러스터링하는 기존 방법보다 멀웨어 탐지 정확도를 높이는데 활용할 수 있다.

3. 오염 분석 기반 멀웨어 클러스터링

3.1 전체 시스템 구성

본 논문에서는 그림 1과 같이 바이너리 코드를 분석한 후 오염 정보에 따라 비정상 행위 특징을 그룹화한 후 Few-shot [11] 학습 모델을 통해 멀웨어를 클러스터링(bucket-N)하는 전체 시스템을 구현하고 실험 결과를 제시한다.

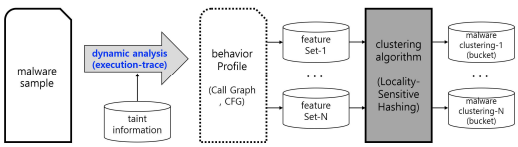

그림 1. 오염 탐지 기반 멀웨어 클러스터링 시스템
Fig. 1. Taint Analysis based Malware Clustering System

바이너리 코드는 실행 과정에서 실행 경로를 추적한 후 비정상 행위를 유발하는 오염 정보가 추가되며 동적 분석을 통해 프로파일 정보, 함수 호출 그래프, 제어 흐름 그래프 등의 실행 정보가 생성된다. 이 정보에서 추가되는 오염 정보에 기반하는 특징을 추출한다(feature Set-N). 특징의 종류가 지나치게 세분화되고 다양해지면 전체 시스템의 속도가 저하되므로 본 논문에서는 학습 모델을 고려하여 5개 이내의 특징을 제한하였다.

본 논문에서는 대용량 및 양질의 바이너리 데이터-셋 구축을 위해 적용하는 활성 학습(active learning)은 최소한 바이너리 샘플 데이터를 가지고 학습 모델이 최대한의 성능을 도출할 수 있도록 반복적으로 최초 바이너리 데이터를 정제한다. 특히, 바이너리 코드 분석의 경우는 많은 양의 학습 및 테스트 데이터 확보 어려움과 분석에 엄청난 분량의 경우의 수가 존재하므로 활성 학습을 적용한 연구는 중요한 의미를 가지고 있다. 또한, 점진 학습(incremental learning)은 학습은 바이너리 코드 분석 학습 모델이 분석을 요구하는 타겟 바이너리 코드를 입력으로 받아 분석 과정에서 생성되는 데이터를 학습 데이터-셋에 반영하는 것으로 학습 모델의 정확도 향상에 기여하고 있다.

3.2 오염 분석 기반 멀웨어 클러스터링

그림 2와 같은 소스 프로그램에 대해 실행 가능한 바이너리 프로그램을 생성한 후 그림 3과 같은 제어 흐름 그래프를 생성한다.

```
1  int main()
2  {
3      int vec1[6] = {1, 2, 3, 4, 5, 6};
4      int vec2[6] = {5, 4, 3, 2, 1, 0};
5      int sum[6];
6      for (int i = 0; i < 6; i++) {
7          sum[i] = vec1[i] + vec2[i];
8      }
9  ...
10 }
```

그림 2. 예제 프로그램
Fig. 2. Example Program

이 과정에서 바이너리 코드의 실행 경로를 파악할 수 있으며 적당한 위치에 오염 정보를 추가하였다.

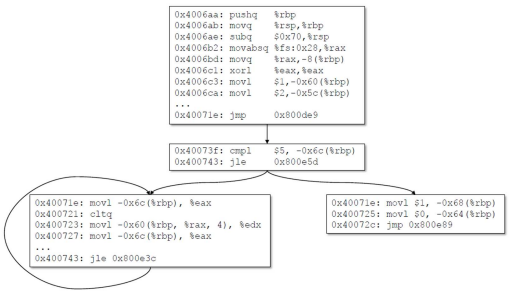

그림 3. 바이너리 코드 분석 기반 제어 흐름 그래프
Fig. 3. Binary Code Analysis based Control Flow Graph

특징이 유사한 비정상 행위를 기반으로 클러스터링을 위해 유사도 값을 쌍으로 찾아 버킷으로 구성하는 지역성 의존 해싱(locality sensitive hashing) 알고리즘을 적용하였다.

```
1:  for each row r do
2:      for each hash function h_i do
3:          Compute h_i(r);
4:      end for
5:      for each column c do
6:          if c has 1 in row r then
7:              for each hash function h_i do
8:                  if h_i(r) is smaller than M(i,c) then
9:                      M(i,c) := h_i(r);
10:                 end if
11:             end for
12:         end if
13:     end for
14: end for
```

그림 4. 지역성 의존 해싱 알고리즘
Fig. 4. Locality Sensitive Hashing Algorithm

3.3 실험 및 평가

본 논문에서 바이너리 코드를 분석한 후 오염 정보를 입력하고 멀웨어 클러스터링 학습과 테스트를 위한 모델을 위해 표 1과 같이 데이터-셋을 활용하였다.

표 1. 바이너리 코드 데이터-셋
Table 1. Binary Code Data-set

NVD, SARD	VulDeePecker, SySeVR
FFMPeg+Qemu	Devign
Chromium, Debian	Real-world projects
CVE	https://cve.mitre.org/

실험을 위한 학습 모델은 그림 5와 같이 바이너리 프로그램에 대한 분석 후에 오염 정보를 입력하기 위해 퍼저(fuzzer) 모듈을 개발하였으며 오염 정보의 종류에 따라 샘플(1 ~ N)을 구분하였다. 또한 바이너리 분석 도구인 Angr를 이용하여 정확한 오염 지점(sink)를 확인하였다. 학습 모델은 메타 러닝의 한 방식으로 기존의 데이터-셋을 단순히 레이블링하는 대신 구별하는 방법을 학습(learn to learn)하는 Few-shot 모델을 적용하였다. 특히, Few-shot learning은 기존의 지도학습과 다르게 지원 집합(support set)과 질의 집합(query set)을 이용하여 학습 집합에 없는 클래스를 맞추는 특징을 가지고 있어 본 논문에서 적정하게 활용할 수 있다. Hot-Byte의 생성을 통해 학습 모델에 절대적으로 영향을 주는 내용을 확인하였다. 또한, 학습과정에 샘플의 내용과 오염이 탐지된 내용을 비교하여 정확한 오염 위치와 원인을 확인하였다.

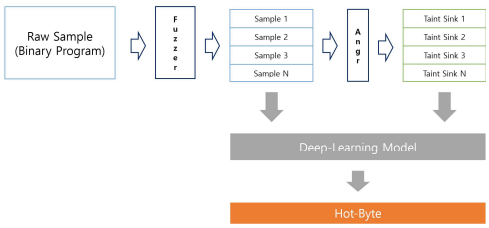

그림 5. 딥러닝 기반 학습 모델
Fig. 5. Deep Learning based Learning Model

실험 환경 및 학습 모델의 세부적인 구성은 표 2와 같다. 세부적인 실험과정에서 3의 계층으로 유지하는 것이 최적으로 분석되었으며 각 계층에서 CPU, GPU에 대해 학습을 진행하였다. 최종적인 학습과 테스트의 결과는 F1-score를 산출하여 학습 과정에서 97 ~ 98%의 성능을 확인하였으며 테스트 과정에서 80 ~

81% 정도의 탐지 성능을 얻었다.

Table 2. Taint Detection Results using Deep Learning Model with 3-hidden Layers
표 2. 학습 모델의 성능 평가 결과

Model		1 Hidden layer		2 Hidden layer		3 hidden layer	
Hidden Layer		DNN 4096 (1ea)		DNN 4096 (2ea)		DNN 4096 (3ea)	
Training time	h/w	CPU	GPU	CPU	GPU	CPU	GPU
	sec	1876.5	1072.3	2215.3	1197.5	3007.5	1660.7
	rela	x1.0	x 1.0	x1.2	x 1.1	1.6	x 1.5
weighted f1_score	train	97 %		98 %		98 %	
	test	80 %		81 %		81 %	

탐지 성능의 결과는 소스 프로그램을 분석한 후 정적 오염 분석 방법을 적용하는 실험에 비해 탐지 속도는 느리지만 정확한 오염 탐지 지점과 탐지율은 다소 높아진 결과를 확인하였다. 추가적으로 대용량의 바이너리 코드에 대해 학습 모델에 적용하면 탐지율은 상승할 것으로 판단된다.

4. 결론 및 향후 연구

소프트웨어 보안을 위해 바이너리 코드에 대해 머신러닝, 딥러닝 기술을 적용하여 고도화되고 지능화된 소프트웨어 보안 취약점 탐지, 사이버 공격 방어 또는 대처 능력 등이 필요한 상황이다. 본 논문에서는 바이너리 코드를 분석한 후 인공지능 기반으로 취약점을 탐지하기 위해 바이너리 코드의 실행 경로를 추적한 후 동적 오염 정보를 입력하여 오염 정보를 따른 특징을 기반으로 취약점을 탐지하고 멀웨어를 그룹화하는 멀웨어 클러스터링 방법을 제안하고 실험하였다.

현재까지 실험된 결과는 오염 정보를 탐지하는 탐지 정확도는 소스 코드 수준 분석 방법에 비해 우수하지만, 분석 대상 소프트웨어 단위가 증가하면 실행 경로 추적에 많은 시간이 소비되는 것을 확인하였다. 현재 소스 코드와 바이너리 코드 분석 수준에서 오염 탐지 정확도와 탐지율을 보다 세부적으로 비교하는 실험을 보완중이며, 앞으로 다양한 바이너리 코드 데이터-셋 구축을 통해 학습 및 테스트를 확장할 예정이고 새로운 멀웨어 클러스터링 알고리즘을 추가적으로 개발할 예정이다.

REFERENCES

[1] Seoksu Lee, Wonchan Oh, Sunnyeo Park, Eun-Sun Cho, In Sung Baek, "Binary Vulnerability Analysis Framework Combining Static and Dynamic Analyses," *Journal of KIISE*, Vol.45, No.12, pp.1217-1226, 2018. 12.

[2] Sang-Gil Cha, "Software Security and Binary Analysis Technique," *Communications of the Korean Institute of Information Scientists and Engineers*, Vol. 36, No.3, pp.11-16, 2018.

[3] Huozhu Wang; Ziyuan Zhu; Zhongkai Tong; Xiang Yin; Yusi Feng; Shi; Dan Meng, "An Effective Approach for Malware Detection and Explanation via Deep Learning Analysis," *International Joint Conference on Neural Networks (IJCNN)*, 2021.

[4] HONGFA XUE, SHAOWEN SUN, GURU VENKATARAMANI, TIAN LAN, "Machine Learning-Based Analysis of Program Binaries: A Comprehensive Study," IEEE Access, Vol.7, pp.65889-65912, 2019.

[5] IDA Pro, https://hex-rays.com/ida-pro/

[6] Dongdong She, Yizheng Chen, Abhishek Shah, Baishakhi Ray, Suman Jana, "Neutaint: Efficient Dynamic Taint Analysis with Neural Networks," In Proceedings of the 41st IEEE Symposium on Security and Privacy. (S&P20), Volume: 1, Pages: 1527-1543. 2020.

[7] Dongdong She, Kexin Pei, Dave Epstein, Junfeng Yang, Baishakhi Ray, Suman Jana. "Neuzz: Efficient fuzzing with neural program learning" In 2019 IEEE Symposium on Security and Privacy(S&P). IEEE, 2019.

[8] "Angr," https://angr.io/

[9] "Ghidra," https://ghidra-sre.org/

[10] P. Godefroid, H. Peleg, and R. Singh, "Learn&fuzz: Machine learning for input fuzzing," CoRR, vol. abs/1701.07232, 2017.

[11] Archit Parnami, Minwoo Lee, "Learning from Few Examples: A Summary of Approaches to Few-Shot Learning," https://doi.org/10.48550/arXiv.2203.04291, 2022.

저자약력

고 광 만(Kwang-Man Ko)

- 1991년 2월 : 원광대학교 컴퓨터공학과(공학사)
- 1993년 2월 : 동국대학교 컴퓨터공학과(공학석사)
- 1998년 2월 : 동국대학교 컴퓨터공학과(공학박사)
- 1998년 3월~현재 : 상지대학교 컴퓨터공학과 교수

〈관심분야〉 컴파일러, 소프트웨어 보안