논문 2021-2-5 http://dx.doi.org/10.29056/jsav.2021.12.05

스트립된 바이너리에서 LSTM을 이용한 함수정보 추출 기법

장두혁*, 김선민**, 허준영**†

Extraction Scheme of Function Information in Stripped Binaries using LSTM

Duhyeuk Chang*, Seon-Min Kim**, Junyoung Heo**†

요 약

악성코드를 분석하여 방어하기 위해, 함수 위치 정보 등을 분석 방식으로 리버스 엔지니어링을 활용한다. 하지만, 스트립 된 바이너리는 함수 심볼 정보가 제거되어 함수 위치 등의 정보를 찾기가 쉽지 않다. 이를 해결하기 위해, BAP, BitBlaze IDA Pro 등 다양한 바이너리 분석 도구가 존재하지만, 휴리스틱을 기반으로 하므로 일반적인 성능이 우수하진 못하다. 본 논문에서는 재귀 하강 방식으로 역 어셈블리어에 대응되는 바이너리를 데이터로 N-byte 기법의 알고리즘을 제시해 LSTM 기반 모델을 적용하여 함수정보를 추출하는 기법을 제안한다. 실험을 통해 제안 기법이 수행 시간과 정확도 면에서 기존 기법들보다 우수함을 보였다.

Abstract

To analyze and defend malware codes, reverse engineering is used as identify function location information. However, the stripped binary is not easy to find information such as function location because function symbol information is removed. To solve this problem, there are various binary analysis tools such as BAP and BitBlaze IDA Pro, but they are based on heuristics method, so they do not perform well in general. In this paper, we propose a technique to extract function information using LSTM-based models by applying algorithms of N-byte method that is extracted binaries corresponding to reverse assembling instruments in a recursive descent method. Through experiments, the proposed techniques were superior to the existing techniques in terms of time and accuracy.

한글키워드 : 양방향 순환신경망, 함수정보, 머신러닝, 순환신경망, 역어셈블 엔지니어링, 스트립 바이너리

keywords : Bidrectional RNN, Function information, Machine Learning, N-byte, RNN, Reverse Engineering, Stripped Binary

1. 서 론

다양한 IT 기기가 우리 생활 곳곳에서 사용되며, 이러한 장비들의 보안결함으로, 스마트폰 해

* 한성대학교 컴퓨터공학과
** 한성대학교 컴퓨터공학부
† 교신저자: 허준영(email: jyheo@hansung.ac.kr)
접수일자: 2021.10.21. 심사완료: 2021.12.14.
게재확정: 2021.12.20.

킹 및 사이버 테러의 위협으로 이어져 소프트웨어 보안의 중요성이 더욱 강조되고 있다[2]. 하지만 소프트웨어 보안에 있어 핵심 기술로 바이너리 분석은 국내적으로 뉴스에서 보도해줄 때만 잠시 대두되며, 분석연구가 활발히 일어나고 있진 않다. 또한 악성코드 관련 기존 악성코드와는 다른 다양한 언어의 악성코드 제작, 악성코드 탐지[1] 등 복잡해져, 이로 인해 분석의 어려움이 커지고 있다. 악성코드의 함수 위치나 정보 등을 분석하는 기법 중 스트립 바이너리를 리버스 엔지니어링[9] 하기 위해, 기존에 활용 되는 분석 도구인 BAP, BitBlaze, IDA Pro 등이 존재하지만 정확도가 비교적 떨어진다는 문제점이 있으며, 이를 보완하기 위해 사람의 주관적인 휴리스틱 알고리즘 기술로 정확성이 조금 떨어지는 한계점이 있다.

바이너리 역어셈블러 방식으로 바이너리 코드를 순차적으로 어셈블리 명령어로 하나씩 대응하는 선형쓸이(Linear Sweep)방식이 있으며, 바이너리 코드의 명령어 얼라인먼트(alignment)를 위한 패딩을 고려하지 않아 잘못 대응된 어셈블리 명령어로 치환하는 문제가 발생하게 된다. 이 점을 보완하여, 순차적으로 명령어에 대응하되, 바이너리 명령어 상의 레지스터 또는 메모리값 범위를 통해 바이너리를 추출하는 재귀 하강(Recursive Descent) 방식이 있다[2].

본 논문에서는 GCC 컴파일러 대상으로 x86 아키텍처의 binutils, coreutils의 패키지에서 ELF(Executable and Linkable Format)의 실행파일을 분석하여 모델 학습에 제안 기법을 적용해 입력 데이터로 사용한다. 실행파일에 대해서, 재귀 하강 방식으로 바이너리를 추출하여, 함수 시작 과 나머지를 이진법으로 라벨링된 데이터를 문맥에 맞게 양끝을 잘라, 시퀀스별 입력 데이터로 양방향 순환 신경망 모델을 통해, 바이너리의 함수 위치 정보에 대한 학습을 진행하였다. 또한 데이터의 불균형 문제를 완화하기 위해 논문에서 제안한 N-Byte 전처리 방법을 사용하였고, 해당 방법을 통해 전체 데이터 대비 약 33%로 적은 데이터양으로 학습 시간에서의 효율과 97% 이상의 높은 성능 정확도를 얻을 수 있음을 확인하였다.

본 논문의 구성은 다음과 같다. 2장에서는 스트립 된 바이너리(Stripped Binary)의 구조와 데이터 추출 방법을 설명하며, 3장에서는 N-Byte 전처리 기법에 관해 설명한다. 4장에서는 사용된 양방향 순환 신경망 모델 구조와 학습 방법에 관해 설명하고, 5장에서는 GCC 컴파일러(6~9)들의 최적화 별 (O0~O3) 각각 학습된 모델의 성능을 비교하고, 최적화 구분 없이 통합하여 학습한 모델의 성능 비교를 진행하였다. 마지막으로 6장에서는 결론과 함께 향후 연구 방향을 제시하며 끝을 맺는다.

2. Stripped binary

2.1 Strip과 역 어셈블리 관계

Strip은 객체 파일에 심볼(Symbol)을 삭제하는 도구, GNU 유틸리티이다. 스트립 바이너리(Stripped Binary)[10] 는 파일의 디버깅 심볼 정보를 제거한 파일을 말한다. 역 어셈블리는 어셈블리를 거꾸로 하는 과정을 의미하며, 컴파일러가 어셈블리 코드를 바이너리 코드로 전환하는 과정을 의미한다. 예로 들어 “add rax, rbx >> 4803c3” 어셈블리 명령어 구성[6]마다 대응되는 바이너리 코드가 정해져 있다. 사진처럼 스트립되기 전에는 해당 부분(Section)과 함수 타입, 크기 등 다양한 심볼 정보를 알 수 있다. 하지만 스트립 과정 후, 디버그 심볼 정보가 없어, 정보를 조회하는데 어려움이 발생한다. 그러나 휴리스틱하게 리버스 어셈블리 작업을 할 수 있으나 정확

도가 낮다. 기존 기법들은 특징 템플릿과 순위에 의존하여 컴파일러 종류와 버전 파악에 시간이 오래 걸리고 계산이 복잡해진다. 또한 컴파일러 버전이나 최적화 레벨에 관한 정보는 추출하지 않고 일반 템플릿을 따르기 때문에 컴파일러에 대한 의미 있는 정보를 항상 제공하지 않는다.

2.2 추출한 데이터

바이너리[3]는 유닉스 시스템의 표준 바이너리 파일 형식을 사용했으며, 객체 파일과 링커를 거쳐서 나온 실행파일을 가지고 16진수로 바이너리를 추출했다. 실제로 프로그래밍한 코드 부분은 함수와 변수 등 다양한 타입으로 구성되어있다. 함수 타입은 Text 부분(Section)에 주로 분포되어있다. 바이너리를 분석하기 위해서, x86 아키텍처 버전 2.3.4의 binutils와 버전 8.3.2 coreutils 패키지를 사용했다. GCC 컴파일러 버전 6부터 9까지, 최적화 레벨 O0부터 O3까지 옵션으로 분류하여 추출하였다.

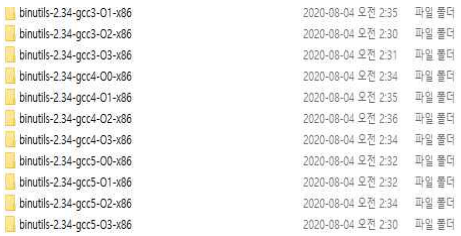


그림 1. GCC 컴파일 별, 최적화 옵션별로 추출한 binutils2.3.4 바이너리
Fig 1. Binutils2.3.4 Binaries extracted by GCC compiler and Optimization options

2.3 관련 연구

2.3.1 Binary Analysis Platform

바이너리 분석 도구로는 Binary Analysis Platform(BAP)[7] 카네기 멜런 대학 CYLab에서 개발한 오픈소스 바이너리 분석 플랫폼 있다. ARM, x86, x86-64, MIPS 아키텍처의 명령어 바이너리에 대해서 바이너리 수준의 명령코드에서 레벨을 한 단계 높여 추상화하여 표현기능인 바이너리 리프팅 기능 또한 제공한다. 리눅스에 포함된 objdump 명령어와 달리 프로그램의 흐름 그래프가 작성 가능하며, 실행파일 자체가 아닌 바이너리 일부 덩어리에 대해서도 분석할 수 있다.

2.3.2 BitBlaze

버클리 대학에서 개발한 오픈소스 바이너리 분석 도구로 악성코드 분석 및 취약점 분석을 주 목적으로 한다. 바이너리 프로그램에서 직접적으로 보안 관련 속성을 추출할 수 있으며, 동적 분석과 정적 분석을 결합하기 때문에 단점을 보완할 수 있다.

2.3.3 IDA Pro

가장 많이 사용되는 바이너리 분석 상용도구로 바인더 또는 DLL에 대한 디스어셈블 기능을 제공하며, 디버깅 기능도 제공한다 여러 가지 운영체제에서 실행 가능하며, 원격 디버깅 기능을 제공하여, 악성코드 분석을 더욱 안전하게 할 수가 있도록 한다. 대화식 인터페이스와 매크로 형 언어로 프로그래밍할 수 있도록 설계되었다.

3. N-byte 기법

3.1 기존 불균형 데이터 솔루션

N-byte 기법으로 명명한 알고리즘은 시계열 데이터의 불균형 문제(Imbalanced Problem)를 위한 솔루션으로 본 논문에서 제안되었다. 순환 신경망 기반 모델(Recurrent Neural Network)에서는 이전 시점 입력의 결과가 다음 시점 입력의 결과로 내보내는 셀의 역할로, 시계열 데이터의

시퀀스별 문맥(Context)이 중요하다. 기존의 불균형 데이터 솔루션으로는 주로 소수 클래스의 값들을 무분별하게 증가하는 오버 샘플링, 높은 비중을 차지하는 클래스의 값들을 임의로 제거하는 언더 샘플링 방식 그리고 모델을 훈련하는 동안 소수 클래스와 다수 클래스에 클래스의 비율에 대해 가중치를 달리 두는 방법으로 가중치 밸런싱(Weight Balancing) 방식이 있다. 하지만, 불균형 시계열 데이터에 기존 불균형 데이터 솔루션을 적용한 결과, 오버 샘플링(Over-Sampling)과 언더 샘플링(Under-Sampling) 방식은 시퀀스별로 무작위성을 가져야 하는데, 실험에 사용되는 바이너리별로 무작위성이 부여해 순서가 임의로 섞이게 되므로 순환 신경망 모델 학습에 대해 문맥 전후로 입력 결과를 다음 입력에 반영이 되는 문맥의 역할 무의미해진다. 가중치 밸런싱을 적용한 결과, 전체 바이너리 중 함수 시작이 아닌 정보로 99% 이상이 0으로 라벨링을 하고, 1% 미만인 함수 시작 정보로 1로 라벨링하여, 너무 극한 불균형 데이터이므로 효과가 미비했다.

3.2 N-byte 기법

N-byte 기법은 극한 불균형 시계열 데이터의 솔루션으로, 순환 신경망 모델에서, 소수 클래스 기준으로 시퀀스 길이 이외에 비중 높은 클래스를 잘라, 시퀀스 단위로 데이터를 재배열하여 모델 학습을 시키는 방식이다. N개 바이트씩 잘라내어 한 개의 시퀀스를 만든다. 그리고 소수 클래스 부분이 없는 시퀀스을 가지고, 소수 클래스 부분이 속해있는 시퀀스 크기만큼 같은 크기의 시퀀스를 만든다. 소수 클래스 부분의 유무 차이로 시퀀스을 만들어 각각 1 : 1 비율로 훈련 데이터 세트를 구성한다. 데이터의 무작위성을 부여하기 위해서, 전체 데이터를 시퀀스 단위로 셔플을 진행하여, 골고루 섞이게 한다.

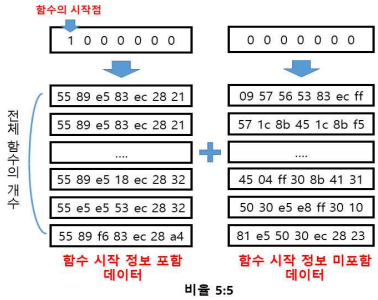


그림 2. 불균형 클래스 데이터를 N-byte로 적용하는 방식
Fig 2. How to apply imbalanced classes data as N-byte

3.3 시퀀스별 함수 정보 분석

함수 시작 정보를 기준으로 바이너리 시퀀스 안에 GCC 컴파일러별로 동일 최적화 옵션이더라도 규칙성을 가진 시퀀스 개수는 다르다. GCC 컴파일러별, 최적화 별 규칙을 가진 바이너리 시퀀스 개수는 다르나, 옵션 특징에 따라, 최적화 옵션이 O0, O1, O3로 갈수록 시퀀스 개수는 증가하며 O1, O3 과 달리 O2 옵션 일 때 인라인 함수에 대해서 최적화 작업을 하지 않기 때문에 최적화 O2일 때, 가장 많다.

표 1. 컴파일러별, 최적화 옵션별 함수 시작 정보 주변 분석
Table 1. Analysis surroundings of function start information by gcc compiler and Optimization Options

개수(개)	binutils				coreutils			
gcc별	O0	O1	O2	O3	O0	O1	O2	O3
gcc6	330	1961	4807	3981	569	1185	3115	2265
gcc7	22	1456	2596	2226	119	685	2327	1574
gcc8	330	1965	7478	6625	565	1162	3373	3623
gcc9	23	1452	2599	2408	570	1207	3520	3257

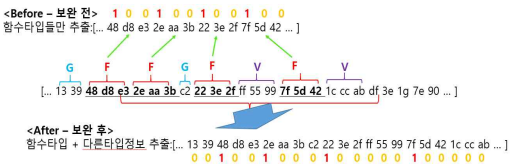
3.4 데이터 구성

실제로 실행파일을 분석할 때, 함수에 대응되는 바이너리만 추출하여 분석하는 것이 아니다. 함수에 대응되는 바이너리를 추출하여 모델 학습

을 통해, 실제로 테스트로 실행파일을 넣어 분석했을 때, 함수 시작 부분을 찾지 못하는 현저히 떨어지는 예측률을 보였다. 그리고 또한 바이너리 코드를 단순히 순차적으로 추출하여 어셈블리 명령어와 대응하면 부정확한 타입의 바이너리 코드를 얻어져 모델 설계 성능에 방해가 된다. 실행파일에는 함수 타입 바이너리뿐만 아니라 여러 가지 타입의 바이너리가 존재하기 때문에, 데이터 세트 추출 시에 실행파일 속 함수가 아닌 바이너리를 포함해서 훈련 데이터 세트를 재구성했다.

그림 3. 함수 타입이 아닌 바이너리를 반영하기 전후 데이터 구성

Fig 3. Configuring data before and after about reflecting - non function type binaries



3.4 컴파일러 버전별, 통합 버전 사용

기본적인 모델의 깊이는 같고, 파라미터 튜닝을 통해 컴파일러별 최적화 지점을 찾으려고 노력했다. 컴파일러 버전이 분류된 상황과 컴파일러별로 함수 시작 정보 시퀀스별로 특징이 달라서, 모델에 각각 로드(Load)했다. 아래 표를 통해서, 알 수 있듯이 가공 전 데이터를 원-최고 인코딩(One-hot Encoding) 과정과 N-byte 방식을 적용한 후, 원-핫 인코딩만 적용한 데이터와 비교했을 때 데이터 크기가 약 33%로 축소된 것을 확인할 수 있었다.

표 2. N-byte 방식 적용과 미적용 학습데이터 비교

Table 2. Compared Train-dataset applied N-byte and Raw data

npy기준(GB)	Raw 데이터					N-byte 적용 전처리				
binutils + coreutils	Data Size					Data Size				
	O0	O1	O2	O3	통합	O0	O1	O2	O3	통합
gcc6	7.4	6.79	7	8.17	27.14	2.37	1.83	1.78	1.66	7.64
gcc7	9.79	7.82	7.93	9.2	31.68	4.32	3.28	3.34	3.62	13.73
gcc8	7.4	6.8	6.8	7.82	27.14	2.37	1.76	1.77	1.69	7.75
gcc9	9.69	7.93	8.17	9.02	31.89	4.88	3.77	3.81	3.71	14.76

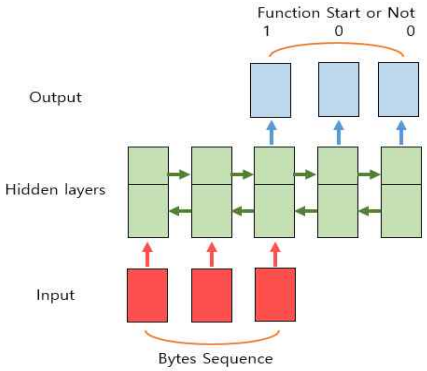
4. 모델링

4.1 양방향 순환 신경망 모델(Bi-directional LSTM Model)[4]

순환신경망 네트워크(Recurrent Neural Network)[5]는 셀(Cell)를 활용한 시계열 데이터 처리 모델로, 입력 또는 출력을 시퀀스 단위로 처리하며, 출력값에 대한 정보가 다음 입력값에 반영되어 시퀀스별로 순환되는 구조 방식이다. 함수정보 또한 함수의 시작점에서의 개행과 동시에 순방향으로 진행될수록 함수의 정보들이 반영된다. Input Sequence 크기로 입력으로, many-to-many의 구조이다.

그림 4. 제안된 양방향 LSTM 모델

Fig 4. Proposed Bi-directional LSTM Model



4.2 모델 학습설정 옵션과 하드웨어 환경

표 3. 하드웨어 설정 & 모델 구동 환경
Table 3. Hardware setting & model environment

목록	성능
윈도우	Windows10
CPU	AMD Ryzen7 2700X
RAM	16G
GPU	NVIDIA GeForce RTX2070

옵션	구성
딥러닝 라이브러리	Keras
Batch_Size	32
CallBacks	patience =10
Validation_split	0.3

5. 결 과

5.1 실험 결과

모델의 입력 시퀀스의 길이와 히든 유닛(Hidden Unit) 메모리 유닛을 컴파일러 GCC 버전별(ver.6~9)로 GCC 6, 8 학습모델에는 (96, 144) 일 때, GCC 7, 9 학습모델엔 (192, 288) 일 때가 최적의 파라미터였다. GCC 6, 8 경우, F1-Score를 기준으로 GCC 6 컴파일러 모델은 O0는 0.987 O1은 0.9788 O2는 0.9726 O3는 0.9783으로 평가지표가 가장 높았다. GCC 8 모델은 O0는 0.9865 O1은 0.9769 O2는 0.9779 'O3는 0.9765로 평가지표가 가장 높았다.

표 4. GCC 6 모델 평가지표의 통계
Table 4. About GCC 6 Statistics of Model Evaluation Indicators

predict			input 16 / unit 24			
gcc	evaluate	utils	O0	O1	O2	O3
gcc6	Accuracy	bin6+core6	0.9998	0.9994	0.9993	0.9996
	Recall		0.9841	0.8927	0.8991	0.8576
	Precision		0.9494	0.7621	0.6956	0.8267
	F1-score		0.9664	0.8222	0.7844	0.8419
			input 32 / unit 48			
gcc	evaluate	utils	O0	O1	O2	O3
gcc6	Accuracy	bin6+core6	0.9998	0.9997	0.9997	0.9998
	Recall		0.9541	0.9308	0.9222	0.9517
	Precision		0.9749	0.8803	0.8695	0.8929
	F1-score		0.9644	0.9049	0.8951	0.9214
			input 48 / unit 72			
gcc	evaluate	utils	O0	O1	O2	O3
gcc6	Accuracy	bin6+core6	0.9999	0.9998	0.9997	0.9999
	Recall		0.982	0.9606	0.9586	0.967
	Precision		0.9757	0.9337	0.8736	0.949
	F1-score		0.9788	0.947	0.9141	0.9579
			input 96 / unit 144			
gcc	evaluate	utils	O0	O1	O2	O3
gcc6	Accuracy	bin6+core6	0.9999	0.9999	0.9999	0.9999
	Recall		0.9926	0.9812	0.9777	0.9891
	Precision		0.9814	0.9763	0.9669	0.9676
	F1-score		0.987	0.9788	0.9726	0.9783
		학습시간	1005.9	714.2	769.3	868.3

표 5. GCC 8 모델 평가지표의 통계
Table 5. About GCC8 Statistics of Model Evaluation Indicators

predict			input 16 / unit 24			
gcc	evaluate	utils	O0	O1	O2	O3
gcc8	Accuracy	bin8+core8	0.9998	0.9994	0.9994	0.9994
	Recall		0.9518	0.8758	0.8058	0.823
	Precision		0.9617	0.745	0.7765	0.7309
	F1-score		0.9567	0.8051	0.7909	0.7742
			input 32 / unit 48			
gcc	evaluate	utils	O0	O1	O2	O3
gcc8	Accuracy	bin8+core8	0.9999	0.9997	0.9997	0.9997
	Recall		0.9715	0.9378	0.981	0.9129
	Precision		0.9711	0.8761	0.8824	0.9004
	F1-score		0.9713	0.9059	0.9013	0.9066
			input 48 / unit 72			
gcc	evaluate	utils	O0	O1	O2	O3
gcc8	Accuracy	bin8+core8	0.9999	0.9998	0.9998	0.9998
	Recall		0.9798	0.9619	0.9555	0.9595
	Precision		0.9759	0.9136	0.9495	0.9251
	F1-score		0.9778	0.9371	0.9525	0.942
			input 96 / unit 144			
gcc	evaluate	utils	O0	O1	O2	O3
gcc8	Accuracy	bin8+core8	0.9999	0.9999	0.9999	0.9999
	Recall		0.9876	0.9851	0.9839	0.9792
	Precision		0.9854	0.9689	0.972	0.9738
	F1-score		0.9865	0.9769	0.9779	0.9765
		학습시간	939.7	719.5	813.6	750.4

표 6. GCC 7 모델 평가지표의 통계
Table 6. About GCC 7 Statistics of Model Evaluation Indicators

			input 32 / unit 48			
gcc	evaluate	utils	O0	O1	O2	O3
gcc7	Accuracy	bin7+core7	0.9999	0.9988	0.9987	0.9986
	Recall		0.973	0.7964	0.7114	0.5518
	Precision		0.9214	0.5209	0.4658	0.3683
	F1-score		0.9465	0.6298	0.563	0.4417
			input 48 / unit 72			
gcc	evaluate	utils	O0	O1	O2	O3
gcc7	Accuracy	bin7+core7	0.9999	0.9996	0.9996	0.9997
	Recall		0.9698	0.8775	0.8808	0.8461
	Precision		0.9532	0.8455	0.8013	0.8175
	F1-score		0.9614	0.8612	0.8392	0.8315
			input 96 / unit 144			
gcc	evaluate	utils	O0	O1	O2	O3
gcc7	Accuracy	bin7+core7	0.9999	0.9998	0.9998	0.9998
	Recall		0.979	0.9567	0.9471	0.9345
	Precision		0.9574	0.9068	0.9038	0.91
	F1-score		0.9681	0.9311	0.9249	0.9221
			input 192 / unit 288			
gcc	evaluate	utils	O0	O1	O2	O3
gcc7	Accuracy	bin7+core7	0.9999	0.9999	0.9999	0.9999
	Recall		0.9674	0.9813	0.9688	0.9525
	Precision		0.9817	0.9586	0.9626	0.9553
	F1-score		0.9745	0.9698	0.9657	0.9539
		학습시간	1951	1614.6	1583.8	1713.8

GCC 6, 8의 학습 시간의 전체적 성향은 비슷하다, 최적화 O1에서 가장 학습 시간이 적게 걸린다. GCC 7 모델은 O0는 0.9745 O1은 0.9698 O2는 0.9657 O3는 0.9539로 평가지표가 가장 높고, GCC 9 모델은 O0는 0.98 O1은 0.9724 O2는 0.9657 O3는 0.9582로 평가지표가 가장 높다. 최적화 옵션별로 학습데이터양은 O0, O3, O1, O2 순으로 높아서, 데이터양에 소요 시간은 비례한다. GCC 6, 8의 모델과 다른 점은 최적화 O2만

비교했을 때, 컴파일러 GCC 9 버전이 데이터 학습량과 함수 개수가 GCC 7보다 많지만, 학습 소요 시간은 GCC 7보다 훨씬 적게 소요된다.

표 7. GCC 9 모델 평가지표의 통계
Table 7. About GCC 9 Statistics of Model Evaluation Indicators

			input 32 / unit 48			
gcc	evaluate	utils	O0	O1	O2	O3
gcc9	Accuracy	bin+core9	0.9999	0.9994	0.9993	0.9993
	Recall		0.9505	0.8581	0.7979	0.8498
	Precision		0.9447	0.7112	0.6929	0.6358
	F1-score		0.9476	0.7778	0.7417	0.7274
			input 48 / unit 72			
gcc	evaluate	utils	O0	O1	O2	O3
gcc9	Accuracy	bin+core9	0.9999	0.9997	0.9995	0.9997
	Recall		0.9757	0.9238	0.761	0.8834
	Precision		0.9552	0.822	0.8333	0.8127
	F1-score		0.9653	0.8699	0.7955	0.8466
			input 96 / unit 144			
gcc	evaluate	utils	O0	O1	O2	O3
gcc9	Accuracy	bin+core9	0.9999	0.9998	0.9998	0.9998
	Recall		0.9717	0.955	0.9039	0.8829
	Precision		0.9678	0.9184	0.9261	0.9212
	F1-score		0.9698	0.9364	0.9149	0.9016
			input 192 / unit 288			
gcc	evaluate	utils	O0	O1	O2	O3
gcc9	Accuracy	bin+core9	0.9999	0.9999	0.9999	0.9999
	Recall		0.9849	0.9695	0.9771	0.9652
	Precision		0.975	0.9753	0.9546	0.9514
	F1-score		0.98	0.9724	0.9657	0.9582
		학습시간	2577.2	1723.1	1314.1	1721.5

학습데이터를 함수 타입뿐만 아니라 주변의 다른 타입 데이터의 학습 결과가 더 좋음을 확인함으로써, 순환신경망 모델(RNN)은 구별해야 하는 표적 데이터뿐만 아니라, 주변 데이터까지 필요하다. N-byte 기법을 활용하여 함수 시작 정보 기준으로 입력 시퀀스만큼 자르기 때문에, 실제 평균 함수 길이인 300바이트보다 작아진다.

N-byte 방식을 적용하여, 가공 전 데이터를 재배열 후 학습을 진행할 수 있다. 이로 인해, 가공 전 데이터 크기 대비 약 33%로 축소된다. 하지만, 모델학습 후 평가지표는 미적용 모델[8]과 비슷한 높은 성능을 보인다. N-byte 방식을 통해 데이터양이 약 33%로 감소했으므로, 모델 학습할 때 컴퓨터 메모리나 그래픽카드에 로드되는 메모리 크기가 낮아지고, 학습 시간 또한 50% 이상 감소한다. 최적화 옵션별로 데이터 학습한 결과보다 최적화 옵션을 모두 통합한 데이터 학습 결과가 최적화 옵션별로 평균 1 ~ 2% 이상 향상된 안정적인 결과를 보였다.

6. 향후 연구

binutils, coreutils 외 다른 패키지 추가로 findutils등 다양한 패키지에서 추출한 실행파일을 활용하면, 함수 시작 정보의 종류 수로 더욱 향상된 성능의 일반성을 가진 모델 설계로 이어질 수 있다고 생각한다. 또한 gcc 컴파일러에는 최적화 옵션 O0 ~ O3 이 있지만, 인라인 패스를 제거하거나, 디버그 출력을 활성화 또는 동적 객체를 분석하여 특징으로 활용해, 더욱 다양한 옵션으로 확장된 모델 제시와 옵션들을 분류할 수 있다고 예상한다.

※ 본 연구는 한성대학교 교내학술연구비 지원과제 임

참 고 문 헌

[1] Dong-Hyeok Park, Eui-Jung Myeong, Joobeom Yun. “Efficient Detection of Android Mutant Malwares Using the DEX file”, Journal of The Korea Institute of Information Security & Cryptology, 26 (4), pp.895-902. 2016. DOI: https://doi.org/10.13089/ JKIISC.2016.26.4.895

[2] CHA, Sang-Gil, “Software Security and Binary Analysis”, Communications of the Korean Institute of Information Scientists and Engineers, 2018, 36.3: 11-16. URL: https://www.koreascience.or.kr/article/JAKO201811553400494.page

[3] Taeeun Kim, et al., “A Study on Hybrid Fuzzing using Dynamic Analysis for Automatic Binary Vulnerability Detection”, Journal of the Korea Academia- Industrial cooperation Society, 2019, 20.6: 541-547. DOI: https://doi.org/10.5762/KAIS.2019.20.6.541

[4] Ho Cheul Jung, Young Ghyu Sun, Donggu Lee, Soo Hyun Kim, Yu Min Hwang, Issac Sim, Sang Keun Oh, Seung-Ho Song, Jin Young Kim, “Prediction for Energy Demand Using 1D-CNN and Bidirectional LSTM in Internet of Energy”, Institute of Korean Electrical and ElectronicsEngineers, (2019) 23(1), 134-142. DOI: http://dx.doi.org/10.7471/ikeee.2019.23.1.134

[5] Hwang, Seong Oun, “A Methodology for Security Vulnerability Assessment Process on Binary Code”, The Journal of The Institute of Internet, Broadcasting and Communication, vol. 12, no. 5, pp.237-242, Oct. 2012. DOI: https://doi.org/10.7236/JIWIT.2012.12.5.237

[6] Jianming Fu, Rui Jin, Yan Lin, Baihe Jiang, Zhengwei Guo, “Function Risk Assessment Under Memory Leakage”, Networking and Network Applications (NaNA) 2018 International Conference on, pp.284-291, 2018. DOI: https://doi.org/10.1109/NANA.2018.8648754

[7] BRUMLEY, David, et al., “BAP: A binary analysis platform”, International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg, pp.463-469. 2011. DOI: https://doi.org/10.1007/978-3-642-22110-1_37

[8] SHIN, Eui Chul Richard; SONG, Dawn; MOAZZEZI, Reza., “Recognizing functions in binaries with neural networks”, 24th USENIX Security Symposium (USENIX Security 15). pp.611-626, 2015. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/shin

[9] DAVID, Yaniv; ALON, Uri; YAHAV, Eran, “Neural reverse engineering of stripped binaries using augmented control flow graphs”, Proceedings of the ACM on Programming Languages, 2020, 4.OOPSLA: 1-28. DOI: https://doi.org/10.1145/3428293

[10] Laune C. Harris and Barton P. Miller, “Practical analysis of stripped binary code”, SIGARCH Comput. Archit. News 33, 5, , pp.63 - 68, December 2005. DOI: https://doi.org/10.1145/1127577.1127590

저 자 소 개

장두혁(Duhyeuk Chang)

2020.2 한성대학교 컴퓨터공학부 졸업
2020.3-현재 : 한성대학교 컴퓨터공학과 석사과정
<주관심분야> 임베디드 시스템, 기계학습 경량화, 운영체제

김선민(Seon-Min Kim)

2021.2 한성대학교 컴퓨터공학부 졸업
2021.3-현재 : 뉴럴웍스 AI Engineer
<주관심분야> 기계학습, 딥러닝

허준영(Junyoung Heo)

1998.2 서울대학교 컴퓨터공학과 졸업
2009.2 서울대학교 컴퓨터공학과 박사
2009.9-현재 : 한성대학교 교수
<주관심분야> 운영체제, 무선 센서 네트워크, 임베디드 시스템, 기계학습