The Journal of The Institute of Internet, Broadcasting and Communication (IIBC)
Vol. 21, No. 4, pp.25-29, Aug. 31, 2021. pISSN 2289-0238, eISSN 2289-0246

https://doi.org/10.7236/JIIBC.2021.21.4.25
JIIBC 2021-4-4

스트립 바이너리에서 합성곱 신경망을 이용한 컴파일러 정보 추출 기법

Extracting Scheme of Compiler Information using Convolutional Neural Networks in Stripped Binaries

이정수*, 최현웅*, 허준영**

Jungsoo Lee*, Hyunwoong Choi*, Junyeong Heo**

요 약 스트립 바이너리는 디버그 심볼 정보가 삭제된 바이너리이며, 역공학 등의 기법을 통한 바이너리 분석이 어렵다. 기존의 바이너리 분석 툴은 디버그 심볼 정보에 의존하여 바이너리를 분석하기 때문에 이러한 스트립 바이너리의 특징이 적용된 악성코드를 감지하거나 분석하는데 어려움이 있다. 이러한 문제를 해결하기 위해 스트립 바이너리의 정보를 효과적으로 추출할 수 있는 기술의 필요성이 대두되었다. 본 논문에서는 바이너리 파일의 바이트 코드가 컴파일러 버전, 최적화 옵션 등에 따라 매우 상이하게 생성된다는 점에 착안하여 효과적인 컴파일러 버전 추출을 위해 스트립 바이너리 대상으로, 전체 바이트 코드를 읽어 이미지화 시킨 후 이를 합성곱 신경망에 적용, 정확도 93.5%을 달성하여 스트립 바이너리를 기존보다 더욱 효과적으로 분석할 수 있는 계기를 제공한다.

Abstract The strip binary is a binary from which debug symbol information has been deleted, and therefore it is difficult to analyze the binary through techniques such as reverse engineering. Traditional binary analysis tools rely on debug symbolic information to analyze binaries, making it difficult to detect or analyze malicious code with features of these strip binaries. In order to solve this problem, the need for a technology capable of effectively extracting the information of the strip binary has emerged.
In this paper, focusing on the fact that the byte code of the binary file is generated very differently depending on compiler version, optimazer level, etc. For effective compiler version extraction, the entire byte code is read and imaged as the target of the stripped binaries and this is applied to the convolution neural network. Finally, we achieve an accuracy of 93.5%, and we provide an opportunity to analyze stripped binary more effectively than before.

Key Words : Convolution Neural Network, Strip binary, Compiler information extraction

*학생회원, 한성대학교 컴퓨터공학부
*학생회원, 삼성서울병원 AI연구센터
**정회원, 한성대학교 컴퓨터공학부(교신저자)
접수일자 2021년 5월 17일, 수정완료 2021년 6월 27일
게재확정일자 2021년 8월 6일

Received: 17 May, 2021 / Revised: 27 June, 2021 /
Accepted: 6 August, 2021
*Corresponding Author: jyheo@hansung.ac.kr
Dept. Division of Computer Engineering, Hansung University, Korea

Ⅰ. 서 론

일반적인 바이너리 파일엔 디버깅 정보가 들어간 심볼 테이블이 존재하고 이를 통해 디버그와 관련된 모든 정보를 조회할 수 있어 파일 분석이 용이하다. 스트립 바이너리는 해당 바이너리 심볼 테이블의 내용을 삭제한 바이너리를 의미하며 역공학 및 디스어셈블 등의 바이너리 분석 기법으로 분석이 어렵다는 특징이 있다. 또한 이러한 특징으로 인해 기존의 바이너리 분석 도구들을 이용하여 스트립 바이너리를 분석할 경우, 기존 도구들은 피처 템플릿과 피처 순위에 의존하여 계산하기 때문에 연산 시간이 많이 소요되고 컴파일러 버전 등의 정보를 고려하지 않기 때문에 일반 바이너리에 비해 항상 의미 있는 정보를 제공하진 않는다는 단점이 있다. 스트립 바이너리에 대한 분석 정확도가 떨어진다는 특징을 이용하여, 악성코드 제작자는 스트립 바이너리를 이용하여 악성코드를 제작한다. 이에 대응하기 위해 효과적인 스트립 바이너리 분석에 대한 새로운 접근법이 필요하게 되었다.

또한, 바이너리 파일은 컴파일러 종류, 링크, 최적화 단계 등에 따라 같은 소스코드라 하더라도, 생성되는 바이너리의 정보는 상이하여 같은 소스코드에서 파생된 바이너리들에 대해서도 분석 결과가 정확하지 않을 수도 있다.[1] 본 논문은 보다 정확한 스트립 바이너리 분석을 위해, 바이트 정보를 추출하여 이를 이미지화 한 후, 합성곱 신경망을 통해 스트립 바이너리 파일의 컴파일러 버전을 추론하여 스트립 바이너리 함수 위치 시작점 추출 등의 연구에서 보다 높은 성능을 제공할 수 있는 방법을 제안한다.

Ⅱ. 관련 연구

바이너리 분석에 관한 연구는 대부분 일반 바이너리에 대한 분석이 주로 이루어져 있으며, 크게 특정 함수 시작점 위치 추출과 바이너리 분석 도구에 대한 관련 연구가 존재한다.

1. 함수 시작점 위치 추출

ByteWeight[2]는 스트립 바이너리 코드에서 함수를 추출하는 도구이다. 기존의 IDA, Dyninst 등의 디스어셈블러 툴은 스트립 바이너리에서 낮은 정확도를 보이며, 사용자가 직접 시그니처를 설정해야 한다는 단점이 있었다. ByteWeight는 해당 부분을 가중 접두사 트리를 사용한 머신러닝 방법을 적용함으로서 특정 함수에 대한 시작점이 0x55(PUSH) 등 특정 바이트 코드에 의해 작성되는 점을 착안하여 해당 바이트 코드를 1로 그 외 코드를 0으로 설정하는 One-Hot Encoding를 적용하여 함수의 시작점을 추출한다.

Recognizing Functions in Binaries with Neural Networks[3]에서는 순환 신경망을 사용하여 함수 시작점 탐지 및 함수 구분을 수행하였다. 해당 논문에서는 각 함수에 포함된 바이트 코드들의 순서를 순차적으로 라벨링하여 훈련 데이터로 주입하며, 훈련된 신경망은 입력된 바이트 코드들의 시작점과 종료점을 추론한 후, 두 지점을 사용해 함수를 구분한다. 앞서 ByteWeight와 동일한 데이터셋으로 실험한 결과, PE x86에서 ByteWeight가 94.57%의 F1 Score를 보인데 반해 해당 논문의 모델은 98.46%로 우세하였으며, ELF x86-64를 제외한 ELF x86, PE x86-64 데이터셋에서도 모두 본 모델의 성능이 더 뛰어났다.

2. 바이너리 분석 도구

바이너리 분석 도구를 구현하기 위해 우선 바이너리 코드를 Binary Lifting 등의 작업을 거쳐 어셈블 코드와 같이 중간 표현 언어(IR, Intermediate Language)로 변환하는 것이 분석을 위한 기조가 되므로 특히 중요하다.

BAP(Binary Analysis Platform)[4]는 바이너리 분석을 위해 명령어를 중간 표현 언어로 변환한 후, 이를 기반으로 제어 흐름 그래프를 생성하면서 바이너리를 분석하고, 전체 바이너리 중 일부를 대상으로 분석을 수행할 수 있다는 특징이 존재한다.

BitBlaze[5]는 악성 코드 분석 및 취약점 분석을 주 목적으로 한 바이너리 분석 도구로써, 동적 분석과 정적 분석을 결합함으로 각 기법의 단점을 보완했다는 특징이 존재한다. 해당 논문에서도 Vine이라는 도구를 이용하여 중간 표현 언어로 변환 후 정적 분석을 수행하고, TEMU라는 도구로 동적 분석을 수행한다. 또한, 각 기법 간의 특징을 결합하여 Rudder라는 도구에서 임의의 입력 값을 넣어 예상된 절차에 따라 수행 여부를 판단하는 테스트 도구도 존재한다.

마지막으로, 기계학습을 이용하여 스트립 바이너리에 대한 임의의 함수에 적절한 이름을 추론하는 DEBIN[6]은 ByteWeight와 BAP를 이용하여 IR 및 함수의 시작점을 추출한 후 각 IR에 대해 Extremely Ramdomized

Tree를 이용하여, 해당 요소가 예측할 가치가 있는지 없는지 이진 분류를 수행한다. 예측할 가치가 없는 요소는 상수, 동적 라이브러리 함수, 더미 함수 등이 이에 해당한다. 예측할 가치가 있는 요소끼리 특정 규칙에 의거하여 의존성 비방향 그래프를 생성한 후, 이를 기반으로 조건부 무작위장(Conditional Random Field)를 이용하여 각 요소의 이름을 추론한다.

Ⅲ. 스트립 바이너리의 특징

스트립 바이너리란 컴파일러로부터 생성된 바이너리 정보, 예를 들면 함수 및 변수, 블록의 위치, 크기, 범위 등의 데이터를 삭제하고 명령어나 레지스터 용도와 같은 저수준의 정보만 가지고 있는 바이너리이다[7]. 스트립 바이너리를 사용하면 바이너리 분석이 어려워지고 크기가 축소되기 때문에 대부분의 상용 소프트웨어, 악성코드, 시스템 라이브러리 등은 스트립 바이너리 파일로 제작된다. 서론에서 언급한 바와 같이 대부분의 기존 바이너리 분석 도구들은 심볼릭 테이블 정보 기반 분석을 실시하고, 초기 단계에 함수들의 엔트리 포인트를 찾는데서부터 시작한다. 하지만, 이 테이블을 삭제하면 정적 분석을 통한 프로그램의 엔트리 포인트, 특정 함수의 시작점 위치 등의 내용 파악에 어려움을 겪는다.

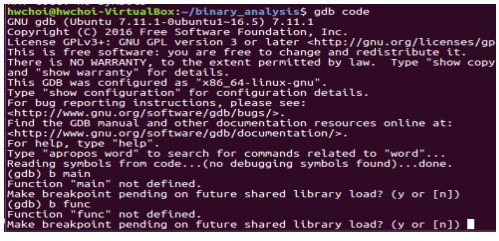


그림 1. 스트립 바이너리의 중단점 조회
Fig. 1. Breakpoint lookup on strip binary

그림 1에서 보이는 바와 같이, GDB 디버거를 이용하여 스트립 바이너리에 대한 각 함수에 대한 중단점을 조회한 결과, 중단점이 조회되지 않는 것을 확인할 수 있다. 특정 함수의 중단점은 심볼릭 테이블에 기록이 되어 있기 때문에, 스트립 바이너리의 중단점 조회를 위해서는 정적 분석이 아닌 동적 분석을 이용하여 GDB 디버거를 이용하여 스트립 바이너리를 실행 시킨 후 동적으로 생성되는 동적 심볼릭 테이블을 이용하여 특정 함수의 중단점을 파악할 수 있다.

하지만, 동적 분석을 실시할 경우 특정 트리거에 의해서 작동이 되는 함수나 인라인이 제거가 되는 함수 등과 같은 예외적인 경우에는 효과적으로 분석할 수 없다는 단점이 존재한다. 이러한 특징을 이용하여 악성코드 제작자들은 특정 트리거에 의해 작동되는 스트립 악성코드 파일을 생성하여 유포한다.

Ⅳ. 데이터셋 수집

1. 데이터셋 수집 및 전처리

스트립 바이너리 분석을 위해, x86 아키텍처의 ELF 포맷의 리눅스 유틸리티 패키지인 Binutils, Coreutils를 이용하여 gcc 컴파일러를 이용하여 버전은 gcc 3 ~ 9로, 최적화 옵션은 O0 ~ O3까지 설정하여 스트립 바이너리로 컴파일링을 진행하였다[8]. 결과적으로 표 1에서 보이는 바와 같이 각 버전별 스트립 바이너리 파일을 생성할 수 있었다.

표 1. GCC 버전별 수집된 스트립 바이너리 개수
Table 1. The number of Stripped Binaries collected for each GCC version

컴파일러 버전	데이터 개수
gcc3	464
gcc4	464
gcc5	407
gcc6	464
gcc7	349
gcc8	464
gcc9	464

하지만, gcc도 버전별로 컴파일러 구성이 다르기 때문에, gcc 5와 7 빌드 에러로 인해 다른 버전보다 데이터 확보가 덜 된 것을 확인할 수 있다.

그 다음으로는, 확보된 스트립 바이너리의 바이트 코드를 추출하여, 이미지화 시키는 작업을 진행하였다. 이 이미지화 기법은 특히, 악성코드 분류에서 전처리 작업으로 자주 사용되는 기법[9]이다. 스트립 바이너리에 대한 이미지화 로직은 다음 그림 2.와 같다.

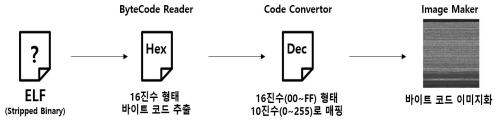


그림 2. 데이터 전처리 흐름도
Fig. 2. Data preprocessing flow chart

가장 먼저, ELF 포맷의 스트립 바이너리를 읽어 16진수 형태의 바이트 코드를 추출한 후, 해당 16진수를 기반으로 이미지를 생성하기 위해, 00 ~ FF의 코드를 0 ~ 255의 10진수로 변환한다. 마지막으로, 이미지를 생성하기 앞서 스트립 바이너리 파일 크기에 따라 생성되는 이미지의 너비를 다르게 조정하여 전체 데이터 셋의 균등화를 이룰 수 있었다. 생성된 이미지는 원활한 학습을 위해 픽셀 값을 255로 나누는 데이터 정규화를 수행하였다. 이미지화 로직에 의해 생성된 스트립 바이너리의 이미지는 다음 그림 3과 같다.

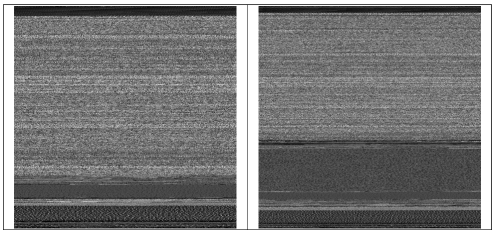


그림 3. 스트립 바이너리에 의해 생성된 이미지
Fig. 3. The images created by Stripped binaries

2. 모델 설계

다음으로, 이 이미지의 분류를 수행할 수 있도록 합성곱 신경망을 설계하였다. 해당 모델의 자세한 구성은 표 2와 같다.

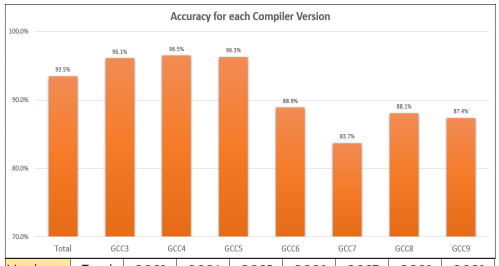
표 2. 합성곱 신경망 구조
Table 2. Structure of a convolutional neural network

Layer	Param
Input	shape = (160, 160, 1)
Conv2D-1	filter=64, relu
Conv2D-2	filter=64, relu
Maxpool2D-1	pool_size=(2, 2)
Conv2D-3	filter=128, relu
Conv2D-4	filter=128, relu
Maxpool2D-2	pool_size=(2, 2)
Conv2D-5	filter=256, relu
Conv2D-6	filter=256, relu
Maxpool2D-3	pool_size=(2, 2)
Conv2D-7	filter=512, relu
Conv2D-8	filter=512, relu
Maxpool2D-4	pool_size=(2, 2)
Flatten	
Dense-1	units=1024, relu
Dense-2	units=7, softmax
Training Setting	
Loss	categorical_crossentropy
Optimazer	adam
Learning late	0.001
Batch size / epochs	128 / 65

Input shape은 생성된 이미지의 평균 너비 값과 비슷한 (160, 160, 1)로 설정하였으며, 각 Maxpool2D 연산 후에 추가적으로 Dropout(late = 0.4)를 추가하여 과적합을 방지 하였다[10].

V. 실험 결과

스트립 바이너리의 컴파일러 정보 추출 실험은 8:2 비율로 학습 데이터와 평가 데이터를 나누어 진행하였으며, 과적합을 피하고 확실한 성능 평가를 위해 K-fold 교차 검증을 수행하였다. 실험 결과, 93.5%의 정확도로 컴파일러 버전을 올바르게 분류하였다.



Version	Total	GCC3	GCC4	GCC5	GCC6	GCC7	GCC8	GCC9
Accuracy	93.5%	96.1%	96.5%	96.3%	88.9%	83.7%	88.1%	87.4%

그림 4. 컴파일러 버전별 정확도
Fig. 4. Accuracy by Compiler Version

세부적으로는 그림 4.와 같이 gcc 버전 3, 4, 5의 구분 정확도가 높았고, gcc 버전 6, 7, 8, 9의 정확도가 상대적으로 낮았다. 이때 gcc6과 gcc8의 판별이 서로 혼동되는 경향이 있었고 gcc7과 gcc9도 동일한 경향을 보였다. 각 버전별 10개의 새로운 스트립 바이너리로 실험한 결과, gcc 3~5 버전에 대해서는 10개 모두 올바르게 구분하였고 gcc 6~9 버전에 대해서는 순서대로 각각 8개, 6개, 8개, 8개를 올바르게 구분하였으며 실제 실험에서 gcc 3~5 버전의 진양성(True Positive) 비율이 나머지보다 높았다.

VI. 결 론

바이너리를 이미지화하여 생성된 데이터를 합성곱 신경망으로 학습하는 방법으로 93.5%의 정확도를 달성하였으며, 해당 버전의 컴파일러 버전 분류 및 바이너리 분석을 더욱 효과적으로 수행할 수 있는 계기를 마련하였다. 본 논문에서는 바이너리의 모든 섹션을 이미지화하

였지만 이후 연구에서 특정 섹션만 이미지화하거나, 기존 데이터의 손실을 더욱 줄일 수 있는 이미지화 기법을 사용한다면 구분 정확도 상승 및 분석이 더욱 용이해질 것으로 전망된다.

또한, 본 논문에서 제시한 방법을 이용하면 각 컴파일러 버전에 보다 최적화된 모델을 생성, 적용함으로써 스트립 바이너리의 함수 시작점 위치 탐색 등의 연구에서 효과적인 Front-end 작업을 수행할 수 있게 된다. 이는 기존에 컴파일러 버전 상이로 발생하던 문제들에 더욱 기민하게 대응할 수 있게 해주며 정확도 향상에 도움을 줄 수 있다.

References

[1] Harris, Laune C., and Barton P. Miller. "Practical analysis of stripped binary code." ACM SIGARCH Computer Architecture News 33.5: 63-68, 2015
DOI: https://doi.org/10.1145/1127577.1127590

[2] Bao, T., Burket, J., Woo, M., Turner, R., & Brumley, D. (2014). {BYTEWEIGHT}: Learning to recognize functions in binary code. In 23rd {USENIX} Security Symposium ({USENIX} Security 14) pp. 845-860, 2014.

[3] Eui Chul Richard Shin, Dawn Song, and Reza Moazzezi, "Recognizing Functions in Binaries with Neural Networks" Proceedings of the 23rd USENIX Security Symposium. August 20-22, 2014

[4] Brumley, D., Jager, I., Avgerinos, T., & Schwartz, E. J. "BAP: A binary analysis platform." International Conference on Computer Aided Verification. Springer, Berlin, Heidelberg, pp. 463-469, 2011.
DOI: 10.1007/978-3-642-22110-1_37

[5] Song, D., Brumley, D., Yin, H., Caballero, J., Jager, I., Kang, M. G. & Saxena, P. "BitBlaze: A new approach to computer security via binary analysis." International Conference on Information Systems Security. Springer, Berlin, Heidelberg, pp. 1-25, 2008.
DOI: 10.1007/978-3-540-89862-7_1

[6] He, J., Ivanov, P., Tsankov, P., Raychev, V., & Vechev, M. "Debin: Predicting debug information in stripped binaries." Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 1667-1680, 2018.
DOI: 10.1145/3243734.3243866

[7] L. C. Harris and B. P. Miller, ‘‘Practical analysis of stripped binary code,’’ ACM SIGARCH Comput. Archit. News, vol. 33, no. 5, pp. 63-68, Dec. 2005.
DOI: 10.1145/1127577.1127590

[8] Deok-Jo Jeon, Dong-Gue Park, "Real-time Linux Malware Detection Using Machine Learning" The Journal of KIIT, Vol. 17, No. 7, pp. 111-122, 2019.

[9] Kim, Se-Jin, et al. "A Study on Classification of CNN-based Linux Malware using Image Processing Techniques." Journal of the Korea Academia-Industrial cooperation Society, 21.9 : 634-642. 2020
DOI: https://doi.org/10.5762/KAIS.2020.21.9.634

[10] S. H. Ryu, “A Study on Random Selection of Pooling Operations for Regularization and Reduction of Cross Validation” Journal of the Korea Academia-Industrial cooperation Society(JKAIS), Vol. 19, No. 4, pp. 161-166, 2018.

저 자 소 개

이 정 수(학생회원)

- 2021년 : 한성대학교 컴퓨터공학부 졸업(학사)
- 2021년 ~ 현재 : 한성대학교 컴퓨터공학부 석사
- 관심분야 : 기계학습, 딥러닝, 컴퓨터 비전

최 현 웅(학생회원)

- 2021년 : 한성대학교 컴퓨터공학부 졸업.(학사)
- 2021년 ~ 현재 : 삼성서울병원 AI 연구센터 Engineer
- 관심분야 : 기계학습, 프로그램 개발

허 준 영(정회원)

- 1998년 : 서울대학교 컴퓨터공학과 졸업
- 2009년 : 서울대학교 컴퓨터공학과 졸업(박사)
- 2009년 ~ 현재 : 한성대학교 컴퓨터공학부 교수
- 관심분야 : 운영체제, 무선 센서 네트워크, 임베디드 시스템, 기계학습

※ 본 연구는 한성대학교 교내학술연구비 지원과제 임