스트립 바이너리에서 저자 식별에 관한 연구

안성관*, 안선우*, 김현준*, 하회리*, 백윤흥*
*서울대학교 전기정보공학부, 서울대학교 반도체공동연구소
sgahn@sor.snu.ac.kr, swahn@sor.snu.ac.kr, hjkim@sor.snu.ac.kr, wrha@sor.snu.ac.kr, ypaek@snu.ac.kr

A Study on Authorship Identification in Strip Binary

Seonggwan Ahn*, Sunwoo Ahn*, Hyunjun Kim*, Whoi Ree Ha*, Yunheung Paek*
*Dept. of Electrical and Computer Engineering and Inter-University Semiconductor Research Center (ISRC), Seoul National University

요 약

최근 익명성이 보장되는 네트워크와 인터넷이 생기며 이를 이용한 악성코드가 증가하고 있다. 이를 막기 위한 방안 중 하나로 코드의 저자를 밝혀내는 연구인 코드 저자 식별이 있다. 이에 관해 최근 연구들은 소스 코드와 바이너리에서 높은 정확도로 저자를 식별해낼 수 있다는 것을 밝혀냈다. 하지만 스트립 바이너리와 관련해서는 연구가 많이 이루어지지 않았다. 이에 본 연구에서는 최근 연구에 사용되는 방법을 스트립 바이너리에 적용하여 실험을 진행하여 그 결과가 좋지 않음을 보였다. 그리고 이를 바탕으로 스트립 바이너리에서 저자 식별이 어려운 이유를 분석하였다.

1. 서론

코드의 저자 식별은 코드에서 코딩 스타일과 같은 저자만의 특징을 뽑아내어 해당 코드를 작성한 저자를 식별해내는 것이다. 이는 privacy 와 anonymity 측면에서 중요하다. 코드 저작권 분쟁 사례에서 판단할 때와 익명의 악성 코드 배포자를 찾아내는 데에 도움이 될 수 있다[1].

연구의 분석 대상인 코드는 소스 코드, 바이너리 코드로 나뉘어서 볼 수 있다. 최근의 연구[4]에서 소스 코드의 attribution 은 99.5%의 높은 정확도로 저자를 식별할 수 있는 것을 밝혀냈다. 높은 성능의 이유 중 하나로 소스 코드에는 변수 이름, 함수 이름과 같이 저자를 확연하게 식별할 수 있는 특징들이 그대로 존재하기 때문이다. 그러나 바이너리 코드는 소스 코드를 컴파일 과정에서 최적화 레벨에 따라 프로그램의 구조도 변경된다. 이로 인해 바이너리 코드는 소스 코드보다 저자 식별이 더욱 어렵다.

특히 바이너리 코드에서 함수 이름, 변수 이름이 적혀 있는 symbol table 을 제거한 스트립 바이너리 코드는 컴파일 과정에서 함수 이름, 변수 이름과 같은 저자의 특징들이 사라진다.

이에 본 논문에서는 코드 저자 식별 연구에서 주로 사용되는 저자 특징 추출 방법을 살펴보고, 스트립 바이너리 코드에서 해당 방법들을 그대로 적용하기 어려운 점을 구체적으로 살펴보고자 한다.

2. 저자 특징 추출 및 선택

최근의 코드 식별 저자 연구 [1]~[4]는 저자의 특징을 추출하기 위해 TF-IDF(Term-Frequency-Inverse-Document-Frequency)를 사용한다. TF-IDF 란 여러 문서로 이루어진 corpus 가 있을 때 어떤 단어가 특정 문서 내에서 얼마나 중요한 것인지를 나타내는 통계적 수치이다. TF-IDF 는 TF 와 IDF 를 곱한 값을 의미한다. 문서를 d, 단어를 t, 총 문서의 개수를 n 이라고 할 때, 각 값은 다음과 같이 계산된다.

$$\text{TF-IDF}(d,t) = \text{TF}(d,t) * \text{IDF}(d,t)$$
$$\text{TF}(d,t) = \text{frequency of t in d}$$
$$\text{IDF}(d,t) = \log\left(\frac{n}{1+df(t)}\right)$$

이를 통해 다른 저자와 비교하여 특징이 되는 단어에 더 높은 가중치 값을 줄 수 있고, 한 저자가 어떤 단어를 자주 사용하더라도 다른 저자들이 모두 해당 단어를 자주 사용하는 경우에는 낮은 가중치 값을 줄 수 있다. 이를 통해 각 저자의 특징적인 단어 또는 단어 조합을 얻을 수 있다.

3. 스트립 바이너리에서 저자 식별의 어려움

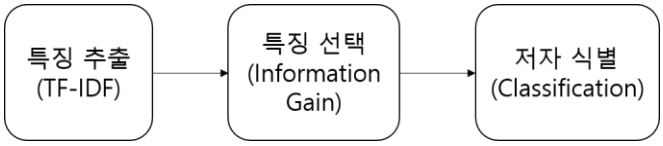
우선 스트립 바이너리에서 특징 추출을 하기 위해서는 스트립 바이너리를 Disassemble, Decompile 해야 한다. 그 결과로 각각 assembly code, decompiled code 이 생성된다. 두 가지 코드에서 저자의 특징 추출을 하기 위해 assembly code 의 instruction 과 decompiled code 의 각 토큰을 이용한다.

Disassemble 과 Decompile 의 결과를 이용한 저자 식별 실험을 위한 데이터셋은 <표 1>에 정리되어 있다. 최적화 레벨 O0 으로 설정한 이유는 최적화 레벨이 높을수록 프로그램의 구조가 바뀌기 때문에, 이를 최소화한 상태에서 스트립 바이너리의 저자 식별의 어려움을 보기 위해서이다. 그리고 저자별 파일 개수 9 개에서 8 개는 학습, 1 개는 추론을 위해 나누었다. 바이너리 파일의 Disassembler, Decompiler 는 Hex-Rays[5] 를 사용하였다.

저자 수	저자별 파일 개수	컴파일 최적화 레벨	프로그래밍 언어
100	9	O0	C++

<표 1> 실험 데이터셋

실험의 과정은 <그림 1>에서 설명했으며, 실험 결과는 <표 2>에 정리했다. 저자 식별 과정에서 저자일 가능성이 가장 높은 한 명만 예측하는 것이 아니라, 저자일 가능성이 높은 5 명까지 예측하도록 하였다.



(그림 1) 스트립 바이너리 코드 저자 식별 실험 과정

Top-K Accuracy	Instruction	Decompile	Instruction & Decompile
Top 1	0.34	0.37	0.36
Top 5	0.50	0.70	0.66

<표 2> 실험 결과

Top 1 을 기준으로 보면, Instruction, Decompile, Instruction & Decompile 모두 정확도가 30% 중반으로 나왔으며, Top 1 과 Top 5 의 정확도 차이가 크게 난다. 이는 스트립 바이너리에서 저자의 특징을 완전하게 구분하기가 어렵다는 것을 의미한다. 그 이유를 Decompiled code 를 기반으로 분석해볼 수 있다. Assembly code 는 원본 파일과 비교하며 설명하기 어려우므로, 이에 대한 설명은 생략한다. <그림 2>와 <그림 3>은 모두 동일한 문법이지만, 변수 이름에서 차이가 난다. 저자의 원본 코드에서 사용하는 변수의 이름 'n'과 배열의 이름 'cx'는 저자의 특징이 될 수 있지만, Decompiled code 에서는 그 특징이 사라지고, Decompiler 의 규칙에 따라 변수 이름이 정해졌다. 이 규칙은 다른 저자의 코드에도 적용되어, 저자의 특징을 완전하게 파악하는 데에 방해가 되었다. 이처럼 스트립 바이너리에서는 저자가 지은 변수 이름과 함수 이름이 사라지므로, 저자 식별이 더욱 어려워진다.

```
for (i = 0; i < n; i++) {
    cx[i] = 0;
}
```

(그림 2) 원본 소스 코드 일부

```
for (i = 0; i < v10; i++) {
    dword_80742C0[i] = 0;
}
```

(그림 3) <그림 2>와 대응되는 Decompiled 코드

4. 결론

본 연구에서는 스트립 바이너리에서 저자 식별의 어려움과 그 이유를 살펴보았다. 어려움의 이유는 스트립 바이너리에서 변수 이름과 함수 이름이 사라지기 때문이다. 이에 스트립 바이너리에서 저자 식별을 위해 코드의 모습에 의존하기보다 그 의미까지 이해하는 연구가 필요해 보인다.

ACKNOWLEDGEMENT

이 논문은 2021 년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원 (No.2020-0-01840, 스마트폰의 내부데이터 접근 및 보호 기술 분석), (No.2020-0-00325, 클라우드 엣지 전주기 데이터 안정성을 위한 추적성 보장 기술 개발)과 2021 년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받았으며 (NRF-2020R1A2B5B03095204), 2021 년도 BK21 FOUR 정보기술 미래인재 교육연구단에 의하여 지원되었음.

참고문헌

[1] Caliskan, Aylin, et al. "When coding style survives compilation: De-anonymizing programmers from executable binaries." arXiv preprint arXiv:1512.08546 (2015).

[2] Abuhamad, Mohammed, et al. "Large-scale and language-oblivious code authorship identification." Proceedings of

the 2018 ACM SIGSAC Conference on Computer and Communications Security. 2018.
[3] Caliskan-Islam, Aylin, et al. "De-anonymizing programmers via code stylometry." 24th {USENIX} Security Symposium ({USENIX} Security 15). 2015.
[4] Abuhamad, Mohammed, et al. "Code authorship identification using convolutional neural networks." Future Generation Computer Systems 95 (2019): 104-115.
[5] “Hex-rays decompiler,” November 2015. [Online]. Available: https://www.hex-rays.com/