머신 러닝 기반 코드 작성자 식별 기술에 대한 조망

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

A Survey on Machine Learning-Based Code Authorship Identification

Hyun-Jun Kim*, Sun-woo Ahn*, Seong-gwan Ahn*, Kevin Nam*, Yun-Heung Paek*
*Dept. of Electrical and Computer Engineering and Inter-University Semiconductor Research Center (ISRC), Seoul National University

요 약

본 논문에서는 특정 코드를 분석하여 해당 코드를 작성한 저자가 누구인지 식별할 수 있는 머신 러닝 기반 코드 저자 식별 기술에 대해 소개한다. 먼저 소스 코드를 분석하여 저자를 확인하는 기법들에 알아볼 것이다. 또한 저자를 식별할 수 있는 정보가 다소 소실된 바이너리 코드를 분석하여 저자를 확인하는 기법을 살펴본 다음, 저자 식별 기법의 향후 연구 방향에 대해 탐색하고자 한다.

1. 서론

최근 프로그래밍에 대한 부가가치가 높아지고 프로그래밍에 대한 진입 장벽이 낮아지게 되면서 일반인들도 손쉽게 프로그래밍을 접할 수 있게 되었다. 또한, 깃헙(github)과 같은 오픈 소스 플랫폼의 발달과 더불어 그들이 작성한 다양한 코드들이 인터넷상에 업로드되었으며, 이에 대한 액세스도 용이하게 되었다. 하지만 불특정 다수가 코드들을 액세스할 수 있게 되면서 다양한 보안 문제가 발생하게 되었다. 예를 들면 특정 작성자가 작성한 코드의 일부를 복사하여 사용한 다음 이를 자신이 작성한 것처럼 속일 수 있으며, 이를 상용으로 사용하여 부당한 이득을 챙길 수 있다.

코드 작성자 식별 기술을 사용하면 코드에 내장된 저자의 코딩 스타일을 인식하여 해당 코드가 다른 특정 코드 작성자에 의해 작성된 것을 알아낼 수 있다. 또한 해당 기술을 이용하여 공격에 사용된 악성 프로그램을 작성한 저자를 역으로 알아낼 수 있으므로 포렌식 분야에서도 유용하게 사용될 수 있다.

코드 내부에 내재된 코드 작성자의 특징을 추출해내는 다양한 연구들이 진행되었는데, 특히 데이터 내의 특정 패턴을 자동으로 추출하여 인식할 수 있는 머신 러닝을 활용한 기술들이 많이 개발되었다.

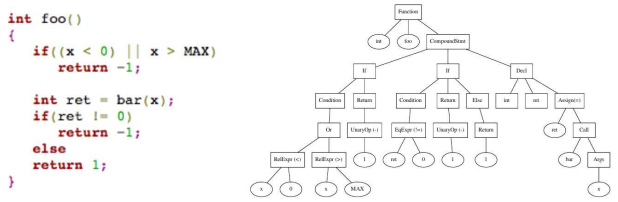
본 연구에서는 머신 러닝을 활용하여 특정 코드를 작성자를 식별해낼 수 있는 기법들에 대해 살펴보고, 이를 기반으로 향후 코드 작성자 식별 기법의 방향을 탐색해보려 한다.

2. 머신 러닝 기반 소스 코드 작성자 식별 기술

2-1. De-anonymizing Programmers via Code Stylometry [1]

해당 기법은 C/C++로 작성된 소스 코드로부터 작성자를 알아내는 기술이다. 해당 연구에서는 소스 코드에 내장된 코드 작성자의 스타일을 알아내기 위해 3가지의 특징(feature)을 추출하였다. 첫 번째는 어휘적 특징(lexical feature)이며 특정 키워드나 개체 식별자(identifier)에 대한 선호도, 함수 사용 빈도에 대한 통계값, 네스팅 깊이(nesting depth) 등을 식별하여 코드 작성자를 판별하는 데에 사용될 수 있다. 이 특징을 추출하기 위해 소스 코드를 분석하여 코드 내 단어 유니그램(unigram) TF(term frequency, 용어 빈도), 함수 당 사용 패러미터(parameter) 수의 평균 등 다양한 범주의 어휘적 특징을 나타내는 특징 벡터(feature vector)를 생성한다. 두 번째는 레이아웃 특징(layout feature)으로 코드 들여쓰기 등 포맷과 관련된 코드 작성자의 특성

을 나타낸다. 이 특징을 알아내기 위해 탭(Tab)과 스페이스(Space)의 선호도 등 화이트스페이스(whitespace) 문자들에 대한 통계값을 추출한다.



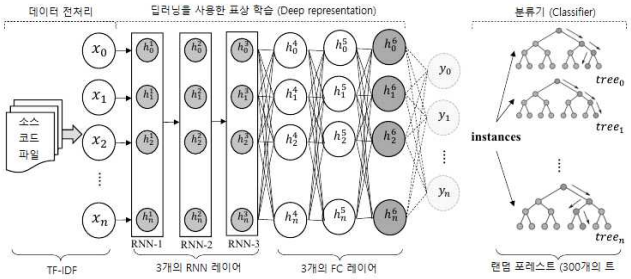
(그림 1) 소스 코드와 대응되는 추상 구문 트리.

마지막 특징인 문법적 특징(syntactic feature)은 본 논문에서 코드 작성자를 식별하는 데에 큰 기여를 한다고 주장한 특징으로, 코드로부터 얻은 추상 구문 트리(abstract syntax tree)를 분석하여 얻어낸다. 추상 구문 트리는 코드를 구성하는 키워드들을 노드로 하여 문법 특성에 맞게 트리를 구성한 것으로, 코드의 구조를 파악할 수 있게 해주는 구조체이며, 이에 코드 작성자의 스타일이 반영된다. 추상 구문 트리로부터는 노드의 최대 깊이, 노드 유니그램(unigram)과 엣지(edge) 정보를 나타내는 바이그램(bigram)의 TF(용어빈도)를 추출해내며, 노드 유니그램에 대해 드물게 사용되는 키워드들에 좀 더 높은 값을 배정하는 TF-IDF(용어빈도-역문서빈도)도 추출하여 코드 작성자 판별이 좀 더 용이하도록 특징 벡터를 구성하였다.

250명의 코드 작성자에 대해 각각 9개의 파일을 선별하여 위 특징들을 추출하면 120,000개의 특징들로 구성된 희소 벡터(sparse vector)가 나오게 된다. 이를 그대로 머신 러닝 모델에 넣으면 과적합(overfitting) 문제가 발생하므로, WEKA(Wikato Environment for Knowledge Analysis)란 자바 패키지에서 제공하는 정보 획득(information gain)을 활용한 차원 축소(dimension reduction) 기법을 사용하여 특징 벡터의 크기를 줄여주었다. 그리고 해당 벡터를 랜덤 포레스트 분류기(Random Forest Classifier)를 사용하여 코드 작성자를 구별하는 태스크(task)를 학습시켰고, 250명의 코드 작성자에 대해 약 98%의 정확도(accuracy)로 판별할 수 있었다. 해당 연구는 문법적 특징에 추가로 어휘적 특징과 레이아웃 특징을 조합하여 높은 정확도로 코드 작성자를 판별해내었다는 점에 의의가 있다.

2-2. Large-Scale and Language-Oblivious Code Authorship Identification [2]

해당 연구는 기존의 소스 코드 작성자 식별 기술을 좀 더 확장하여 딥러닝 모델을 활용하여 코드 작성자를 알아내었다.



(그림 2) Large-Scale and Language-Oblivious Code Authorship Identification 구조도.

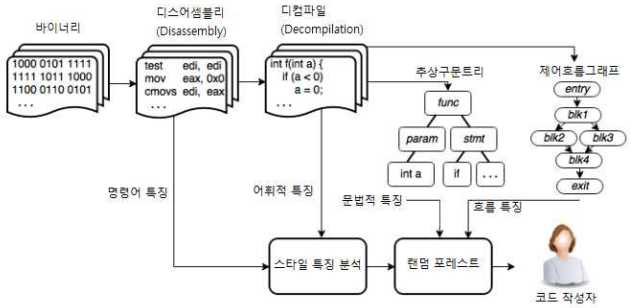
해당 연구에서는 복잡한 추상 구문 트리를 사용하지 않았고, 대신 코드에 사용된 토큰들의 유니그램, 바이그램, 트라이그램(trigram)에 대해 TF-IDF를 사용하여 특징 벡터를 추출하였다. 그 다음에 가장 수치가 높은 k개의 특징을 추출하여 특징 벡터의 차원을 감소시킴으로써 과적합 문제를 방지하였다. 최종적으로 얻은 특징 벡터에 대해 3개의 순환신경망(RNN, Recurrent Neural Network) 레이어와 3개의 완전 연결 계층(FC, Fully Connected Layer) 레이어로 구성된 딥러닝 모델에 넣어서 특징 벡터의 패턴을 잘 나타내는 deep representation(심층 표상)을 학습한다. 코드 작성자의 스타일이정보가 집약되어 있는 deep representation에 대해 랜덤 포레스트 분류기를 사용하여 코드 작성자 판별을 수행하였다. 결과를 연구 [1]과 비교했을 때, 더 많은 500명의 C++ 코드 작성자에 대해서 동일하게 약 98%의 정확도로 판별할 수 있음을 보였다. 그리고 타겟으로 하는 코드 작성자 수를 늘려도 정확도가 많이 감소하지 않음을 보였다. 또한 C/C++ 뿐만 아니라 자바, 파이썬으로 작성된 코드에 대해서 동일한 방법을 사용하여 성공적으로 코드 작성자를 판별해낼 수 있음을 증명하였다. 해당 연구는 딥러닝 모델을 활용하여 간단한 전처리만을 통해 코드 작성자 판별이 가능함을 보였으며, 확장성(scalability) 문제를 완화하였고 다양한 프로그래밍 언어에 해당 기법을 적용 가능함을 보였다.

3. 머신 러닝 기반 바이너리 코드 작성자 식별 기술

바이너리 코드는 소스 코드와는 달리 코드 작성자의 스타일을 알아내는 데에 필요한 정보가 많이 소실된다. 특히 심볼 테이블(Symbol Table)과 같은 정보가 제거된 'stripped binary'에 대해서는 심볼의 선

호도와 같은 정보를 활용할 수 없다. 그래서 기존의 바이너리 연구[4]에서는 기본 블록(Basic Block)으로 구성된 제어 흐름 그래프(Control Flow Graph)를 활용하여 흐름 특징을 추출하여 코드 작성자 판별을 시도하였지만 낮은 성능을 보였다.

3-1. When Coding Style Survives Compilation De-anonymizing Programmers from Executable Binaries [3]



(그림 3) When Coding Style Survives Compilation De-anonymizing Programmers from Executable Binaries 구조도.

본 연구에서는 미리 컴파일이 된 바이너리에 디스어셈블리(disassembly)와 디컴파일(decompile)을 수행하여 얻은 코드들로부터 필요한 특징들을 추출하여 코드 작성자 판별을 수행하였다. 먼저 디스어셈블리를 하여 얻은 어셈블리 코드에서 저레벨 명령어 특징을 추출한다. 이 과정에서 두 개의 다른 어셈블러를 사용하여 서로 잡아내지 못하는 특징들을 보완하였으며 디스어셈블리한 코드에서 유니그램, 바이그램, 트라이그램, 2줄에 걸쳐서 나타나는 6-그램에 대해 TF 값을 추출하였다. 디컴파일한 코드는 심볼 정보가 대부분 소실되므로 단어(word) 유니그램만 추출하여 어휘적 특징으로 사용한다. 그리고 디컴파일한 코드는 고레벨 언어로 작성되어 있기 때문에 소스 코드에 대한 코드 작성자 판별 기법들을 그대로 사용 가능하다. 따라서 추상 구문 트리와 제어 흐름 그래프를 생성하여 문법적 특징과 흐름 특징을 추출해낼 수 있다. 추출된 모든 특징들에 대해 이전 연구들에서 사용했던 차원 감소 기법과 랜덤 포레스트 분류기를 사용하여 코드 작성자를 식별하였다. 기존 바이너리 연구 [4]는 코드 작성자 100명에 대해 61%만 식별하는 낮은 정확도를 보였지만, 본 연구에서는 96%의 정확도를 보이며 성능을 큰 폭으로 향상시켰다. 그리고 추가로 컴파일 후 디컴파일을 하는 과정에서 문법적 특징이 약 80% 정도가 남아있음을 보여 디컴파일한 코드의 추상구문트리를 사용하는 방식이 유효함을 증명하였다. 본 연구는 디스어셈블리, 디컴파일을 통해 소스 코드 작성자 식별 기법들을 응용할 수 있게 하여 효과적으로 바이너리에 대해서도 작성자 식별을 가능하게끔 만들었다는 점에서 그 의의가 있다.

4. 차후 연구 방향

소스 코드에 대해 코드 작성자를 식별해내는 연구들은 충분히 많이 진행되었지만, 바이너리 코드에 대한 연구는 아직 충분하지 않은 상황이다. 많은 수에 저자에 대해서도 높은 정확도로 판별할 수 있는 기법에 대한 연구를 더 진행할 수 있을 것이다.

또한 해당 분야에 사용되는 머신러닝/딥러닝 모델을 속이기 위한 적대적 예제(Adversarial Examples) 연구들도 최근 많이 진행되고 있다. 적대적 예제 기법으로 변조한 코드를 코드 작성자 식별 모델에 넣으면 자신의 코드를 다른 코드 작성자가 작성한 것처럼 속일 수 있을 것이다. 그리고 조건부 적대신경망(Conditional GAN)과 같은 생성 모델을 사용하면 코드 작성자가 원하는 스타일을 강제로 코드에 내포시켜 특정 코드 작성자가 작성한 것처럼 속이는 것이 가능할 것이다. 하지만 이러한 기법이 제대로 적용되려면 코드 수준에서의 변형이 전처리를 거친 다음에 생성된 특징 벡터에 어떻게 영향을 끼치는가에 대한 연구가 추가로 필요하다. 만약 내가 원하는 특징 벡터를 생성하게 만들면서 코드 동작에 차이점을 만들지 않도록 원본 코드를 변형할 수 있으면 해당 기술을 유효하게 사용할 수 있을 것이다.

5. 결론

본 논문에서는 머신 러닝을 활용한 코드 작성자 식별 기술을 다룬 3개의 연구를 살펴보고, 각 논문에서 어떤 특징들과 머신 러닝 모델을 활용하여 코드 작성자를 식별해내는지 알아보았다. 아직 해당 분야에서 성능을 개선할 여지가 남아있으며, 다른 모델을 사용하여 다양한 목적의 연구를 추가로 진행할 수 있을 것이다.

6. Acknowledgement

이 논문은 2021년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원(No.2020-0-01840,스마트폰의 내부데이터 접근 및 보호 기술 분석)과 한국연구재단의 지원을 받아 수행된 연구이며(NRF-2020R1A2B5B03095204), 2021

년도 BK21 FOUR 정보기술 미래인재 교육연구단에 의하여 지원된 연구임을 밝힙니다.

참고문헌

[1] Caliskan-Islam, Aylin, et al., “De-anonymizing programmers via code stylometry.”, USENIX Security Symposium, Washington D.C., 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, New York, 2018.

[3] Aylin, Fabian, et al., “When Coding Style Survives Compilation: De-anonymizing Programmers from Executable Binaries.“, San Diego, NDSS, 2018.

[4] Rosenblum, Nathan, Xiaojin Zhu, and Barton P. Miller., “Who wrote this code? identifying the authors of program binaries.”, European Symposium on Research in Computer Security, Berlin, 2011.