석사학위논문

자동화 분석을 통한 악성코드 저자 식별 모델 연구

제주대학교 대학원

융합정보보안학협동과정

이 상 우

2022년 2월

자동화 분석을 통한
악성코드 저자 식별 모델 연구

지도교수 조 정 원

이 상 우

이 논문을 융합정보보안학협동과정 석사학위
논문으로 제출함

2021년 12월

이상우의 융합정보보안학협동과정 석사학위
논문을 인준함

심사위원장 변 영 철 ㊞

위 원 조 정 원 ㊞

위 원 박 남 제 ㊞

제주대학교 대학원

2021년 12월

목 차

표 목 차

그 림 목 차

<국문요약>

자동화 분석을 통한
악성코드 저자 식별 모델 연구

이 상 우

제주대학교 일반대학원 융합정보보안학협동과정

지도교수 조 정 원

IT기술의 발달로 인해 긍정적인 변화가 일어나는 반면 부정적인 변화도 같이 발생하고 있다. 모듈화와 표준화로 인한 악성코드의 대량생산이나, 취약점을 이용하여 지속적으로 공격이 이루어지는 APT공격 등이 자주 발생하고 있어, 기존의 보안시스템만으로 방어하고 대응하기에는 한계점이 있다. 최근 이러한 한계점을 해결하기 위해 인공지능 기술을 활용하는 연구가 많아지고 있고, 악성코드 저자 식별 연구도 다양해지고 있다.

악성코드 저자 식별 연구는 기존의 저자 식별 분야에서 확대된 연구 분야이며, 저자가 알려진 악성코드의 특징을 파악해 알려지지 않은 악성코드에 대입 후 악성코드의 저자를 유추하거나, 악성코드의 여부를 판별하는데 사용된다. 현재는 악성코드 저자 식별을 통해 APT공격과 같은 지속공격의 패턴을 파악하거나, 악성코드 포렌식 기반의 탐지 기법 중 하나로 활용되고 있다. 저자를 식별하는 분석 방법으로는 소스코드에서 특징을 추출하는 소스코드 기반 분석 방법과 바이너리에서 특징을 추출하는 바이너리 기반 분석 방법으로 이루어진다. 하지만 악성코드의 모듈화, 표준화로 인한 대량의 악성코드를 소스코드 기반 분석 방법과 바이너리 기반 분석 방법을 통해 특징을 추출하기에는 시간과 인력이 모두 부족하다.

그러므로 본 연구에서는 자동화 분석을 사용하여 빠르게 특징을 추출하고 분

석하여 악성코드의 저자를 식별하는 모델을 설계하였다. 자동화 분석은 Tool을 이용한 분석 방법이며, 전문 인력이 없이도 악성코드 파일이나 고유 해시값 등을 통해서 분석할 수 있고, 분석 시간도 다른 악성코드 분석 방법 중에서 가장 빠른 분석 방법이다. 실험은 6개의 악성코드 저자 그룹에 대해 다양한 머신러닝 분류 알고리즘을 적용하여 진행하였고, 저자 식별을 위한 특징으로는 자동화 분석에서 추출해 낼 수 있는 런타임 모듈과 Kernel32.dll API로 선정하였다. 또한 실험 결과를 바탕으로 기존의 연구와 비교를 진행한 결과 기존 연구보다 대체로 높은 정확도를 보여주었고, 자동화 분석을 사용함으로써 기존의 소스코드 기반 분석 방법과 바이너리 기반 분석 방법보다 빠르게 특징을 추출하여 악성코드의 저자를 식별할 수 있었다. 연구에서 제안한 자동화 분석을 통한 악성코드 저자 식별 모델을 통하여 대량 생산되는 악성코드와 APT공격에 대해 적용한다면 기존의 악성코드 저자 식별 방법보다 좋은 성능을 낼 것으로 기대한다.

I. 서 론

IT기술의 발달은 자율주행 자동차, 의약품 관리 시스템 등 사회 전반에 긍정적인 변화를 일으키고 있다. 하지만 긍정적인 변화뿐만 아니라 표준화된 악성코드 생성이나, 모듈화로 인한 악성코드 대량생산, APT(Advanced Persistent Threat)공격 등의 부정적인 변화도 일어나고 있다. 사이버 위협을 가하는 사이버 공격 또한 고도화, 지능화되고 있어 기존의 대응방법으로는 대응하기에 한계가 있다. 2020년 발표된 Verizon의 '데이터 침해 조사 보고서(DBIR)' 에 따르면 2016년 이후로 트로이목마와 같은 단일성 악성코드의 사용이 줄어든 대신 취약점을 찾고 오랜 기간 공격하는 지능형 지속 공격인 APT공격과 같은 효율적인 공격이 늘어났다고 하였다[1]. 또한 2021년 발표된 'CISCO 중소기업 사이버 보안: 아시아 태평양 디지털 방어 보고서' 에 따르면 한국의 중소기업 중 33%가 사이버 공격을 겪었다고 하였고, 그 중 85%가 악성코드에 의한 공격이라고 하였다[2]. 기존의 사이버공격 대응방법은 전문 인력에 의존한 보안관제시스템 구축이다. 이러한 방법은 모듈화로 인해 대량생산되거나, 변이된 사이버 공격, APT공격의 경우 대응하기 어렵고, 중소기업의 경우 보안관제시스템을 구축하기 어렵다는 한계점이 있다.

사이버 공격의 대부분은 악성코드로 이루어지기 때문에 공격자를 식별하기 위해서는 악성코드 저자 식별(Malware Authorship Attribution)이 필요하다. 악성코드 저자 식별 연구 분야는 기존의 알려진 악성코드를 분석하여 저자의 특징을 파악하고, 알려지지 않거나 새로운 악성코드에 대입하여 저자를 유추하고 대응방법을 찾는 연구 분야이다[3]. 또한 공격자의 공격 패턴을 찾아 APT공격과 같은 지속공격의 피해를 줄일 수 있고, 난독화 방법 등을 파악하여 악성코드 포렌식 분야의 중요한 정보를 제공할 수 있다. 악성코드 저자 식별은 소스코드 기반 분석과 바이너리 기반 분석으로 구분된다[4]. 소스코드 기반 분석은 악성코드의 소스코드를 수집한 후 사용된 변수 이름, 함수 이름, 문법 등을 분석하여 저자의 특징을 정의하여 저자를 식별하는 것이고, 바이너리 기반 분석은 악성코

드의 바이너리를 추출하고 분석하여 저자의 특징을 정의하여 식별하는 방법이다. 소스코드 기반 분석 방법은 정확도가 높지만 악성코드의 소스코드를 수집하기 어렵다는 한계점이 있고, 바이너리 기반 분석은 바이너리를 수집하기 쉽지만 정확도가 떨어지는 한계점이 있다[5]. 두 분석 방법 모두 소스코드, 바이너리로의 변환 후 특징을 추출해서 분석하기 때문에 악성코드의 표준화와 모듈화로 대량생산이 가능해진 현재 각각의 악성코드에 대한 저자를 식별하기에는 시간적인 측면에서의 어려움이 있고, 악성코드 변환과정이나, 특징 추출과정에서의 전문인력을 통해 진행함으로 인력적인 어려움이 있다.

자동화 분석은 악성코드의 모듈화로 인해 대량생산이 가능해진 현재 기존의 정적 분석, 동적 분석으로 악성코드를 분석하기에는 시간과 인력의 소비가 크기 때문에 발전되어진 분석 방법이다. 자동화 분석 툴(Tool)은 악성코드 파일, 해시값, 또는 웹사이트일 경우 URL을 입력하면 보고서 형식으로 레지스트리, PE(Portable Executable)정보 등을 제공해준다. 자동화 분석은 분석에 필요한 시간과 인력의 소비가 크지 않기 때문에 초기 분석단계에서 자주 사용되지만, 전문가에 의해서 분석되는 과정만큼 상세하거나 정확하지 않을 수 있다.

본 논문에서는 소스코드 분석 방법의 데이터 획득의 한계점, 바이너리 분석 방법의 특징 선정의 한계점, 그리고 두 가지 방법이 갖는 분석 및 특징추출에 소요되는 시간이 오래 걸린다는 한계점을 해소하기 위한 목적을 갖고 연구를 진행하였다. 연구의 주요 내용 및 방법은 다음과 같다.

첫째, APT1, APT10, APT29, Gorgon Group, Lazarus Group, Winnti 그룹의 악성코드 데이터를 수집하고 빠르게 분석할 수 있는 자동화 분석을 통해 그룹 식별을 위한 데이터인 런타임 모듈과 Kernel32.dll API를 추출한다.

둘째, 추출된 데이터를 바탕으로 일정 기준의 빈도분석을 진행하여 각각의 그룹에 대한 특징을 정의한다.

셋째, 정의된 특징으로 인공지능 모델을 생성하기 위해 다양한 머신러닝 분류 알고리즘을 적용한다.

넷째, 정확도, 정밀도, 재현율, F1 Score의 평가척도를 이용하여 가장 성능이 좋은 머신러닝 분류 알고리즘을 선택한다.

다섯째, 기존연구와 제안한 모델의 성능을 비교 분석한다.

위의 내용들을 통해 기존의 방법보다 빠르게 악성코드의 저자를 식별할 수 있는 모델을 제안하였다. 연구의 범위는 자동화 분석을 통해 런타임 모듈과 Kernel32.dll API를 추출하기 때문에 윈도우즈 기반 실행파일로 설정하였다.

본 논문의 구성은 2장에서 배경지식과 선행연구에 대해 설명하고 3장에서 자동화 분석 기반 악성코드 저자 식별 모델을 제안한다. 또한 4장에서 제안한 모델을 통한 실험 및 결과 분석을 진행하고, 5장에서 결론 및 향후 연구에 대해 서술한다.

II. 이론적 배경

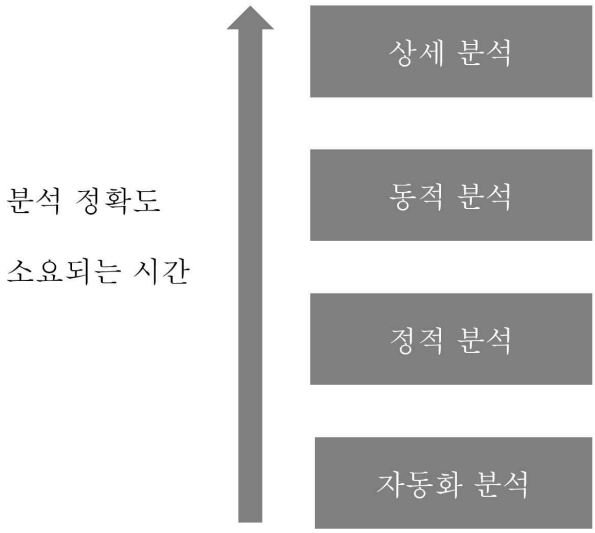
1. 악성코드(Malware)

악성코드는 Malicious Software의 줄임말이며, 기존 보안 인프라의 취약점을 악용하여 컴퓨터를 손상시키거나 시스템에서 정보를 훔치는 의도를 가지고 있다[6]. 악성코드의 주요 목표는 시스템 접근 권한 획득, 시스템 서비스 중단, 서비스 거부, 기밀 정보 도용 등이 있다. 2021년에 발표된 TitanFile의 '7 Type of Computer Malware and How to Prevent Them' 보고서에 따르면 악성코드는 여러 유형이 있으며 일반적으로 아래 <표Ⅱ-1>과 같이 악성코드를 구분할 수 있다[7]. 대부분의 악성코드는 이메일이나 웹 사이트, P2P 등의 네트워크 환경에서 전파된다.

<표 Ⅱ-1> 악성코드 분류

악성코드 종류	내용
Trojan Horse	• 정상 소프트웨어로 위장하여 악성 소프트웨어를 실행하는 악성코드
Spyware	• 은행 정보, 다양한 계정의 비밀번호나 개인정보 같은 민감한 정보를 훔치는 악성코드
Adware	• 컴퓨터 화면이나 모바일 장치에 원치 않는 광고를 표시하는 악성코드
Rootkits	• 공격자에게 장치에 대한 접근 및 제어 권한을 부여하도록 설계된 악성코드
Ransomware	• 파일을 암호화하고 일정 비용을 지불할 때 까지 파일에 대한 접근을 차단하는 악성코드
Worms	• 자신을 복제하여 네트워크를 통해 자동으로 확산되는 악성코드
Keyloggers	• 사용자의 키보드 입력 값을 로그에 기록하도록 설계된 악성코드

2. 악성코드 분석 기법



[그림 Ⅱ-1] 악성코드 분석 방법

악성코드 분석 기법은 [그림 Ⅱ-1]과 같이 자동화 분석, 정적 분석, 동적 분석, 상세 분석으로 구분할 수 있다. 분석 속도는 자동화 분석이 가장 빠르며, 정적 분석, 동적 분석, 상세 분석 순으로 분석 속도가 높아진다. 분석 정확도는 상황에 따라 다를 수도 있지만 평균적으로 상세 분석이 가장 정확도가 높으며, 추출할 수 있는 데이터의 양도 많고, 동적 분석, 정적 분석, 자동화 분석 순으로 정확도가 낮아진다. <표 Ⅱ-2>는 각각의 분석 방법의 장점과 단점, 추출할 수 있는 정보를 기술했다.

<표 Ⅱ-2> 악성코드 분석 방법

분석 방법	추출 정보	장점	단점
자동화 분석	• 통신 주소 • 레지스트리 수정 목록 • 런타임 모듈 • …	• 악성코드를 실행하지 않기 때문에 빠르고 안전하게 분석할 수 있다. • 파일이 없어도	• 정확도가 떨어진다. • Anti-VM기능이 있는 악성코드는 탐지가 어렵다.

		분석할 수 있다.	
정적 분석	• 문자열 정보 • 해시 값 • 리소스 정보 • …	• 악성코드를 실행하지 않기 때문에 빠르고 안전하게 분석할 수 있다. • 자동화 분석 대비 추출할 수 있는 정보가 많다.	• Anti-VM기능이 있는 악성코드는 탐지가 어렵다.
동적 분석	• 레지스트리 • 파일시스템 • 프로세스 • …	• 악성코드를 직접 실행시키며 분석하는 방법이기 때문에 추출할 수 있는 정보가 많다. • Anti-VM 기능이 있어도 탐지할 가능성이 높다.	• 전문적인 기술과 시간이 필요하다. • 악성코드를 직접 실행시키며 분석하기 때문에 위험성이 높다.
상세 분석	• 난독화-복호화 방법 • …	• 악성코드의 대부분의 정보를 추출할 수 있다.	• 전문적인 기술과 시간이 필요하다.

자동화 분석은 악성코드의 자동화, 모듈화로 인해 생산되는 시기가 빨라지면서 기존의 분석 방법으로 분석하기에는 시간과 인력 모두 부족하기 때문에 이를 해결하기 위해 개발된 분석 방법이다. 자동화 분석 툴(Tool)을 사용해 악성코드를 분석하는 방법이고, Cuckoo Sandbox, BSA(Buster sandbox analyzer), malware.com 등의 분석 툴이 있다. 분석 방법은 악성코드를 실행하지 않고, 악성코드 파일이나 md5, sha1 등의 해시 값을 분석 툴에 입력하면 자동적으로 분석이 이루어진다. 각각의 툴마다 제공되는 악성코드의 정보는 다르지만 일반적으로 통신 주소, Registry 수정, 런타임 모듈, PE정보 등을 제공한다. 하지만 난독화나 패킹처럼 Anti-VM기능을 가지고 있을 경우, 분석하기 힘들고, 정확도가 떨어진다는 단점이 있다.

정적 분석은 악성코드를 실행시키지 않고 파일을 어셈블리 또는 바이너리 파일로 변환하여, 악성코드여부를 확인하는 시그니처 기반 분석 방법이다. 파일을

실행하지 않고 분석하기 때문에 안전하다는 장점이 있지만 자동화 분석과 마찬가지로 난독화나 패킹처럼 Anti-VM기능을 가지고 있을 경우, 분석하기 힘들다. 하지만 자동화 분석과는 다르게 악성코드 파일을 직접 분석하는 것이기 때문에 자동화 분석보다는 식별 정확도가 높다. 정적 분석으로 추출할 수 있는 정보는 문자열 정보, 해시 값, 리소스 정보 등을 얻을 수 있다.

동적 분석은 동적 제어 환경에서 악성코드의 행위를 분석하는 행동 기반 분석 방법이다. 일반적으로 가상환경에서 악성코드를 실행하며 실행될 때 나타나는 변화를 모니터링 한다. 레지스트리, 파일시스템, 프로세스 등 악의적인 행위의 상세한 과정을 파악할 수 있으며, 악성코드를 직접 실행시킴으로 정확하고 자세한 정보를 얻을 수 있다. 또한 악성코드의 난독화나 패킹이 되어있어도, 악성코드를 분석할 수 있는 장점이 있다. 하지만 악성코드를 직접 실행하기 때문에 분석 작업에 위험성도 있고, 인력으로 실행시키는 것이기 때문에 분석에 소요되는 시간이 많다.

상세 분석은 분석 전문가가 자동화 분석, 정적 분석, 동적 분석을 통해 추가 분석이 필요할 경우 시행하는 분석 방법이다. 상세 분석에는 필요에 따라 다양한 분석 방법이 존재하는데, 어떤 방식으로 난독화가 되어 있고 복호화가 되는지, 또는 동적 분석 단계에서 어떤 방식으로 탐지를 회피했는지 등의 정보를 추출한다. 상세 분석의 경우 악성코드 파일에 대한 대부분의 정보를 추출할 수 있지만, 전문가의 영역이며 시간이 오래 소요된다는 단점이 있다.

3. 악성코드 저자 식별

기존의 저자 식별 연구는 19세기 Mendenhall의 The Plays of Shakespeare의 연구로부터 시작되었다. Shakespeare의 희곡 중에서 자주 사용되는 단어(“and”, “to” 등)의 빈도를 통계 분석하여 Shakespeare의 특징을 식별하고 저자가 알려져 있지 않던 작품에 대입해 저자가 Shakespeare인지 확인하는 연구였다[4]. 이를 악성코드에 접합시킨 것이 악성코드 저자 식별이다. 초반의 연구는 악성코드의 소스코드를 분석하여 저자를 식별했는데, 악성코드의 개발 속도

가 증가하고, 난독화 등의 문제로 인해 악성코드의 저자를 식별하는 방법도 발전하게 되었다. 악성코드의 저자를 식별하여 특정하는 이유는 표준화, 모듈화로 인해 대량생산되는 악성코드에 대한 프로필을 개발할 수 있고, 난독화 기술이 어떻게 적용되었는지 알 수 있기 때문이다. 또한 APT공격의 경우는 오랜 시간 지속적으로 이루어지기 때문에 공격의 패턴을 파악하여 피해를 줄일 수 있고, 대량생산되는 악성코드의 경우는 기존의 악성코드를 일부 수정한 변종이기 때문에 저자를 특정할 수 있다면 기존의 대응방법을 통해 피해를 줄일 수 있다. 현재 악성코드의 저자를 식별하는 방법은 소스코드 기반과 바이너리 기반이 있다. 각각에 대한 설명은 다음과 같다.

1) 소스코드 기반 저자 식별

기존의 저자 식별 연구에서 사용되던 방법을 소스코드에 대입한 분석 방법이다. 소스코드에서 저자의 제작 스타일에 대한 패턴을 찾아 이를 특징으로 이용한다. 특징에는 변수 명이나 함수 명, 문자열 길이 등 여러 가지가 있고, 일반적인 저자 식별과 비슷한 특징 때문에 정확도가 높다. 하지만 배포된 악성코드의 소스코드를 수집하기가 어렵고, 디컴파일러를 사용하여 소스코드를 수집한다고 해도, 변환하는 과정에서 시간이 오래 소요되고, 각각의 컴파일러마다 컴파일 과정에서 차이점이 존재하기 때문에 파일을 소스코드로 변환했을 때 함수 명이나 변수 명 등의 소스코드를 추출하기 어렵다는 한계점이 있다.

2) 바이너리 기반 저자 식별

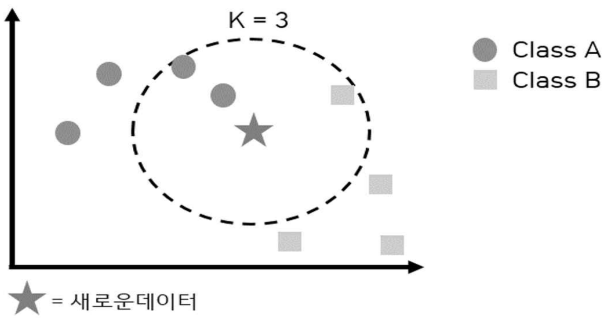
소스코드 기반 저자 식별에서 악성코드의 소스코드 획득에 대한 어려움을 해결하기 위해 제언된 방법이다. 바이너리 기반 저자 식별은 악성코드 파일을 바이너리로 변환한 후 바이너리에서 특징을 찾아 분석하는 방법이다. 바이너리에서 정의할 수 있는 특징은 바이너리 파일에서 찾을 수 있는 문자열, opcode 리스트 해싱값인 함수, API이름 등이 있다. 악성코드 파일을 바이너리로 변환하기 때문에 데이터를 획득하기 어려운 소스코드 기반 분석 방법의 한계점을 해결할 수

있지만, 바이너리 자체에서 특징을 추출할 때 전문 인력을 통해 추출하기 때문에 대량생산되는 악성코드에 적용하기 힘들고, 중복된 특징이 많기 때문에 특징을 정의하기 어렵다. 또한 저자가 다양해지면 정확도가 떨어진다는 한계점이 있다.

4. 머신러닝 분류 알고리즘

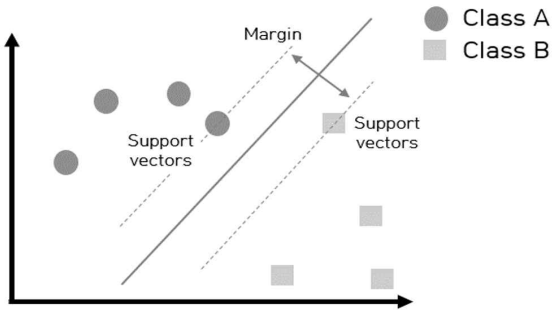
인공지능 지도학습의 일종으로 기존 데이터와 레이블을 통해 새로운 데이터의 레이블을 판별한다. 분류 알고리즘의 종류로는 k-NN(k-Nearest Neighbor), SVM(Support Vector Machine), Decision Tree, Naive Bayes와 같은 기본적인 분류 알고리즘이 있고, Adaptive Boosting, Gradient Boosting과 같은 Boosting 알고리즘이 있다.

Biau and Devroye(2015)에 따르면 k-NN은 입력된 데이터를 특정 값으로 분류하기 위해 기존 데이터의 분포 중에서 입력된 데이터로부터 인접한 k개의 데이터를 찾아 k개의 데이터 중 가장 많이 인접한 값으로 데이터를 분류하는 알고리즘이다[8]. k-NN은 파리미터 설정 값이 k값 하나로 간단하며, 거리에 기반한 분류 알고리즘이므로 숫자로 구성된 데이터 셋에 대해서 높은 정확도를 가진다. 하지만 비교할 속성이 많아질수록 분류속도가 느린 단점이 있다. [그림 Ⅱ-2]에서 k값이 3일 때 새로운 데이터가 사각형 데이터 셋보다 원형 데이터 셋에 가깝기 때문에 새로운 데이터를 원형 데이터로 분류한다.



[그림 Ⅱ-2] k-NN 동작예시

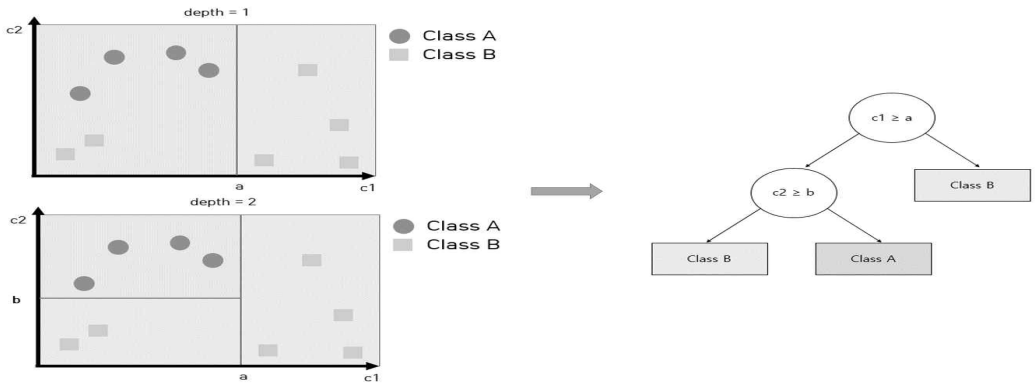
Vishwanathan(2002)에 따르면 SVM은 기존의 데이터를 통해 결정 경계선(Decision Boundary)을 구해 데이터의 구역을 나눈 후 새로운 데이터의 위치에 따라 분류를 해주는 분류알고리즘이다[9]. 특성이 다양한 데이터를 분류하는데 강점이 있고, 적은 데이터로도 높은 정확도를 기대할 수 있다는 장점이 있다. 모델의 정확도를 결정짓는 하이퍼 파라미터는 Cost와 Gamma가 있다. Cost는 마진의 너비를 조절하는 변수로서, 클수록 너비가 좁아지고, 작을수록 너비가 넓어진다. Gamma는 결정 경계의 곡률을 조절하는 변수이며, 클수록 곡률이 완만해지고, 작을수록 곡률이 급격해진다. [그림 Ⅱ-3]은 SVM의 동작 과정을 나타낸다.



[그림 Ⅱ-3] SVM 동작 과정

Anthony J(2004)에 따르면 Decision Tree는 데이터의 특징을 바탕으로 분리해 Tree를 생성하고, 입력 데이터를 Tree에 통과시켜 분류하는 알고리즘이다[10]. 모델의 분류과정을 시각화할 수 있어 이해하기 쉬우며, 높은 정확도를 가진다는 장점이 있지만 과대적합이 되기 쉽다는 단점이 있다. 과대적합을 피하기 위한 하이퍼 파라미터는 Max_Depth, Min_Samples_Split, Min_Sample_Leaf, Random_State가 있다. Max_Depth는 트리의 최대 깊이를 설정할 수 있는 변수이며, Min_Sample_Split은 자식 노드를 갖기 위한 최소한의 데이터의 개수를 설정할 수 있는 변수이다. Min_Sample_Leaf는 주변 노드의 최소 데이터의 개수를 설정할 수 있는 변수이며, Random_State는 동일한 정수를 입력했을 때 학습 결

과를 항상 같게 만들어주는 변수이다. [그림 Ⅱ-4]는 Decision Tree의 동작 과정을 나타낸다.



[그림 Ⅱ-4] Decision Tree 동작과정

Naive Bayes는 대표적인 분류 알고리즘이며, 데이터를 독립적인 사건으로 가정한 후 Bayes 정리에 대입시켜 가장 높은 확률의 레이블로 분류를 한다. 문서 분류에 높은 정확도를 보이며, 학습 속도가 빠르다는 장점이 있지만 문서 분류이외의 데이터에는 정확도가 낮아지는 단점이 있다.

Zhi-Hua Zhou(2019)에 따르면 Boosting 알고리즘은 앙상블 기법중 하나이며 동일한 알고리즘의 분류기를 통해 여러 개의 분류기를 만들어 가중 투표를 통해 예측 값을 얻는 기법이다[11]. Boosting의 특징은 순차적으로 학습하는 것과 가중 투표에 있다. Boosting 알고리즘 중Adaptive Boosting은 첫 번째로 약한 분류기를 통해 1차 분류를 수행한다. 그 이후 제대로 분류되지 못한 오류 데이터에 대해 가중치를 높여 지속적으로 정확도를 높이며 강한 분류기를 만들어내는 Boosting알고리즘이다. 하이퍼 파라미터는 base_estimators, n_estimators, learning_rate가 있다. base_estimators는 학습에 사용하는 알고리즘으로 분류기를 생성할 때 사용할 알고리즘을 선택하는 변수이다. n_estimators는 생성할 약한 분류기의 개수를 정하는 변수이다. 마지막으로 learning_rate는 학습을 진행할 때마다 적용하는 학습률을 정하는 변수이다.

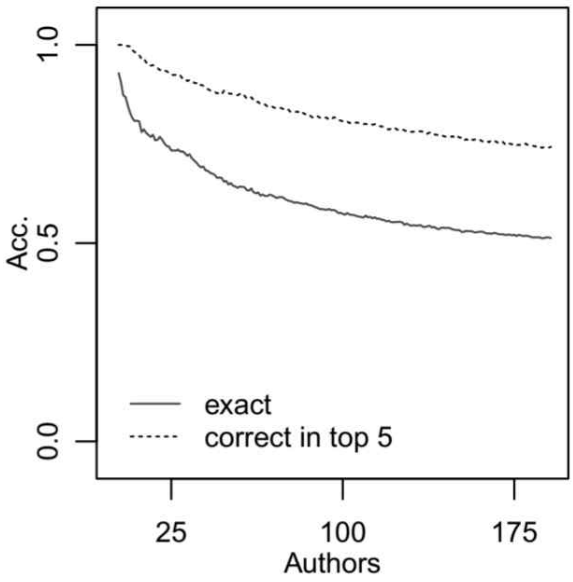
Gradient Boosting은 분류할 수 있는 예측 모형이며 X-Y로 되어 있는 데이터에 대해 높은 성능을 보이는 알고리즘이다. Gradient Boosting 알고리즘은 경

사하강법을 통해 약한 분류기가 만든 잔여 오차(Residual Error)에 새로운 분류기를 학습시키는 점에서 Adaptive Boosting과의 차이가 있다. 하이퍼 파라미터는 Tree에 관해서는 Decision Tree와 비슷한 파라미터를 가지고 있고, Boosting에 관해서는 loss, n_estimators, learning_rate, subsample이 있다. loss는 경사하강법에서 적용될 비용을 지정하는 변수이고, n_estimators는 생성할 트리의 개수를 지정하는 변수이다. learning_rate는 Adaptive Boosting과 같이 학습을 진행할 때마다 적용하는 학습률을 정하는 변수이고 subsample은 개별 트리가 학습에서 사용하는 데이터 샘플링의 비율을 정하는 변수이다.

5. 악성코드 저자 식별 관련 연구

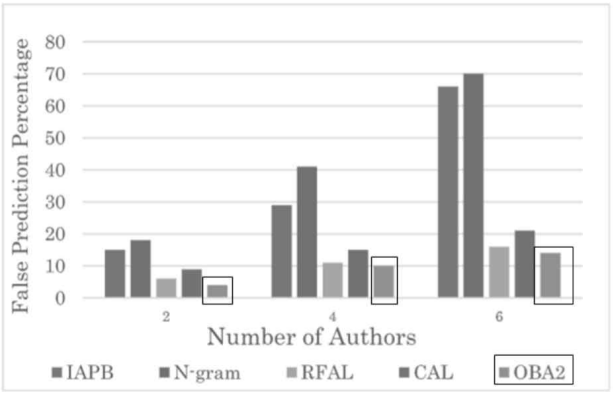
악성코드의 저자를 식별하는 연구는 악성코드 탐지나 대응에 관한 연구보다는 활성화 되어있진 않지만 저자별 악성코드의 난독화 방법이나, 악성코드 제작 특징 등의 중요한 악성코드 포렌식 정보를 제공할 수 있고, 저자의 특징을 파악해 저자가 알려지지 않은 악성코드에 대입하여 저자를 유추할 수 있는 연구 분야이기 때문에 지속적으로 연구되어왔다. 최근 대부분의 악성코드 저자 식별에 관한 연구는 소스코드 기반 분석과 바이너리 기반 분석으로 머신러닝을 사용하는 연구가 진행되고 있다.

Nathan Rosenblum et al(2011)은 소스코드를 바이너리코드로 변환하여 구문 분석을 진행하였고, Idioms, Graphlets, Supergraphlets, Call Graphlets등을 특징으로 선정하였다[12]. 선정된 특징을 바탕으로 머신러닝 알고리즘을 통해 소스코드의 저자를 군집화하고 분류하였다. [그림 Ⅱ-5]처럼 20명의 저자일 경우 77%의 정확도를 보였고, 상위 5명의 저자는 94%의 정확도를 보였다. 악성코드의 저자가 아닌, 일반 소스코드의 저자를 식별하는 연구였고, 또한 소스코드를 바이너리 코드로 변환할 때와 바이너리코드에서 특징을 추출할 때 시간소요가 크기 때문에 현재 대량생산되는 악성코드에 적용하기 힘들다.



[그림 Ⅱ-5] 저자 수에 따른 정확도

Saed Alrabaee et al(2014)은 바이너리 기반 분석을 기존의 IAPB(Identify the Author of Program Binaries)를 보완한 3가지 계층의 OBA2(An Onion approach to Binary code Authorship Attribution)방법론을 제시하였고, 그림 [Ⅱ-6]과 같이 부정확한 예측 비율을 낮춘 연구를 진행하였다[5].



[그림 Ⅱ-6] OBA2 방법론

OBA2는 기존의 저자 식별보다 높은 정확도를 보여줬지만, 바이너리 코드가 난독화 해제 되었을 때만 식별이 가능하고, 하나의 컴파일러에서의 결과 값만을

가지고 있다. 또한 저자가 증가함에 따라 정확도가 감소하고, 추출하는 특징이 많기 때문에 시간소요가 많이 소요된다는 한계점이 있다.

홍석진 외(2018)는 1,944건의 악성코드 샘플을 수집하고, 문자열, 함수, 네트워크, 레지스트리 등의 특징을 선정하였고, 총 111,144개의 데이터를 통해 실험을 진행하였다[13]. 실험 방법은 딥러닝을 통해 진행하였고, 은닉층과 정점 수, 학습률 등의 파라미터 조절을 통해 결과를 보였고 기존의 악성코드 저자 분류에서 좋은 성능을 발휘한 SVM 모델과 비교했다. 비교 결과는 <표 Ⅱ-3>과 같이 SVM을 사용할 때 보다 1.5%정도의 정확도 향상이 있었다. 연구는 다양한 특징들을 선정해 딥러닝을 통해 높은 정확도를 도출했으나, 다양한 특징 추출에는 전문 인력의 투입이 필요하고 오랜 시간이 소요되므로 현재의 대량의 악성코드에 전부 적용하기 어렵다는 한계점이 있다.

<표 Ⅱ-3> 연구 결과

	딥러닝	SVM
Accuracy	94.96%	93.42%
Precision	94.88%	93.32%
Recall	94.96%	93.42%
F-Measure	94.82%	93.12%

신건윤 외(2020)는 유전 알고리즘을 기반으로 한 악성코드 공격자 그룹의 특징을 추출하는 프레임워크를 제안하였다[14]. 이 프레임워크는 바이너리 코드, 어셈블리 코드, 소스코드를 수집한 후, 이를 Authorship Attribution 기반의 분석을 통해 데이터 셋을 구축하고, 유전 알고리즘을 적용하여 군집화 했다. 기존의 연구결과와의 비교 결과는 <표 Ⅱ-4>과 같다. 정확도가 향상된 것을 볼 수 있지만, 소스코드 기반 분석 방법과 바이너리 기반 분석 방법을 사용하여 특징을 추출하는 것이기 때문에 오랜 시간이 소요된다는 한계점이 있다.

<표 Ⅱ-4> 공격자 그룹 특징 추출 프레임워크 실험결과 비교

System	저자 수	정확도
Alrabaee et al(2014)	5+	84%
Rosenblum et al(2011)	50+	78%
공격자 그룹 특징 추출 프레임워크 실험결과(2020)	5	86%

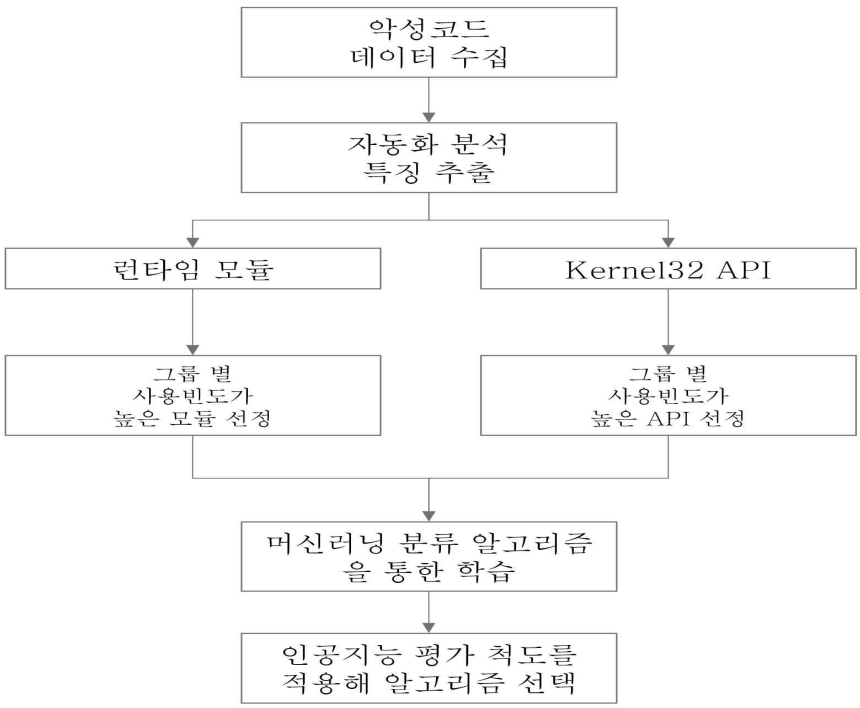
황철훈 외 3명(2020)은 바이너리 기반 분석 방법에서 저자가 많을 때 나타나는 특징의 중복을 해결하기 위해 서바이벌 네트워크 개념을 도입하였다[15]. 기존의 바이너리 기반 저자 식별 연구와 같은 방법을 사용하여 데이터 셋을 구축하였지만 서바이벌 네트워크 개념을 도입하여 저자 별 중복되는 특징을 제거함으로 정확도가 최대 5%가 증가하였다. 하지만 대량생산되는 악성코드를 전부 분석하여 특징을 추출하고, 중복되는 특징을 제거할 때 오랜 시간이 소요된다는 한계점이 있다.

III. 자동화 분석 기반 악성코드 저자 식별 모델

1. 제안 모델

본 연구에서는 기존의 악성코드 저자 식별에서 사용되었던 소스코드 기반 분석의 데이터 획득에 대한 한계점과 바이너리 기반 분석의 특징 추출의 한계점을 보완하고, 빠른 시간에 저자를 식별할 수 있는 자동화 분석 기반 악성코드 저자 식별 모델을 제안한다.

[그림 Ⅲ-1]은 제안한 모델의 생성과정을 나타낸 그림이다.



[그림 Ⅲ-1] 자동화 분석 기반 악성코드 저자 식별 모델 생성과정

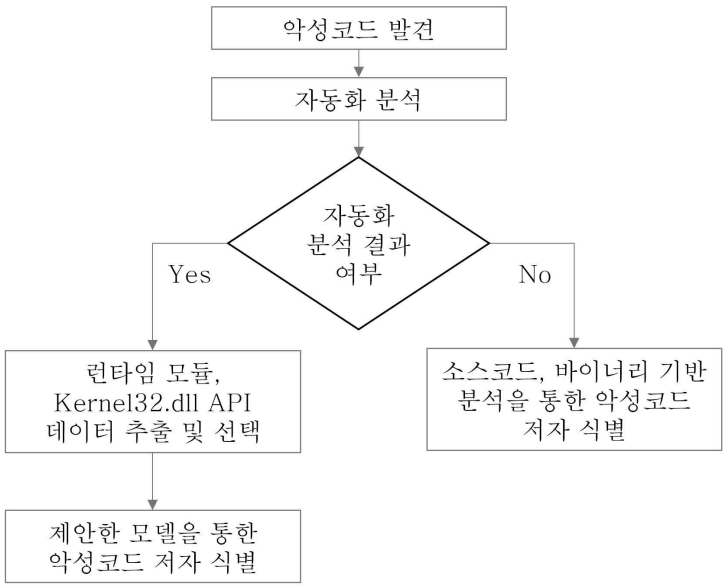
첫째. 학습을 위해 APT1, APT10, APT29, Gorgon Group, Lazarus Group, Winnti 그룹의 데이터를 수집한다.

둘째. 수집한 데이터를 자동화 분석을 통해 분석을 하고, 그 중 런타임 모듈과 Kernel32.dll API 데이터를 추출한다. 기존 연구에서는 디컴파일러를 사용하여 소스코드로 변환시키거나, 바이너리로 변환시킬 때의 시간이 오래 소요되었다. 이에 비해 자동화 분석은 소스코드나 바이너리로의 변환 과정이 없고, 특징추출을 자동으로 진행하므로 시간소요를 줄일 수 있다.

셋째. 런타임 모듈과 Kernel32.dll API에 대한 그룹별 빈도분석을 진행하고, 일정 기준이 충족된 데이터를 통해 데이터 셋을 구축한다.

넷째. 선별된 데이터를 머신러닝의 학습 정확도를 높이기 위해 One-Hot Encoding을 통한 전처리과정을 거친다.

다섯째. 머신러닝 분류 알고리즘을 통해 제안 모델을 학습하고, 인공지능 평가척도를 대입하여 적절한 알고리즘 선택하고 모델을 생성한다.



[그림 Ⅲ-2] 제안한 모델을 바탕으로 한 저자 식별 방법

[그림 Ⅲ-2]는 완성된 모델을 통해 악성코드의 저자를 식별하는 전체적인 과정이다.

첫째. 악성코드를 발견할 시, 자동화 분석을 진행한다.

둘째. 자동화 분석 결과 여부에 따라 제안한 모델을 사용할지 기존 방법을 사용할지 선택한다.

셋째. 자동화 분석 결과가 나오면 런타임 모듈과 Kernel32.dll API데이터를 추출하고 모델에 적용시킬 데이터를 선택한다.

마지막으로 선택한 데이터를 본 논문에서 제안한 모델에 적용시켜 악성코드의 저자를 식별한다. 만약 자동화 분석 결과가 나오지 않는 경우는 기존의 악성코드 저자 식별 방법인 소스코드, 바이너리 기반 분석을 진행한다.

2. 특징(Feature) 선정

악성코드 저자 식별 모델의 정확도를 높이기 위해서는 머신러닝에 학습시킬 특징을 선정하는 것이 중요하다. 본 연구에서는 그룹 분류를 위한 특징으로 런타임 모듈과 Kernel32.dll API를 선택하였다. 런타임 모듈은 악성코드 파일을 실행했을 때 런타임에 로드되는 동적 라이브러리 및 모듈이다. 또한 Kernel32.dll API은 메모리, 파일, 하드웨어 작업과 관련된 기능을 가지고 있는 API이고, 대부분의 악성코드는 메모리나 파일관련 작업을 처리하기 위해 Kernel32.dll API를 사용하기 때문에 식별 특징으로 선정했다. 런타임 모듈과 Kernel32.dll의 API에는 [그림 Ⅲ-3]과 같은 정보들이 있다.

런타임 모듈
Advapi32.dll
Cryptbase.dll
User32.dll
Oleaut32.dll
Dnsapi.dll
WS2_32.dll
Ntdll.dll
Wininet.dll
:
:

Kernel32.dll API
Sleep
CreateThread
CloseHandle
GetProcAddress
CreatePipe
ReadFile
LCMapStringW
IsBadCodePtr
:
:

[그림 Ⅲ-3] 런타임 모듈, Kernel32.dll API 종류

런타임 모듈의 대표적인 모듈과 각각의 기능은 <표 Ⅲ-1>와 같다.

<표 Ⅲ-1> 런타임 모듈의 대표적 모듈

런타임 모듈	기능
Advapi32.dll	서비스 관리자나 레지스트리의 관한 기능을 담고 있는 모듈
Cryptbase.dll	윈도우즈 디지털 서명에 사용되며 기본 암호화의 기능을 담고 있는 모듈
User32.dll	사용자 인터페이스 컴포넌트의 생성 및 동작에 관한 기능을 담고 있는 모듈.
Oleaut32.dll	RAM(Random Access Memory)의 작업을 담당하는 기능을 담고 있는 모듈
Dnsapi.dll	DNS(Domain Name Service) 클라이언트 API에서 사용하는 기능을 담고 있는 모듈
WS2_32	네트워크 통신의 기능을 담고 있는 모듈
Ntdll.dll	커널 인터페이스와 같은 NT시스템 기능을 담고 있는 모듈
Wininet.dll	상위 수준의 네트워크 함수와 같은 다양한 인터넷 기능을 담고 있는 모듈

런타임 모듈은 로드되는 모듈마다 기능이 다르다. 따라서 모듈의 기능을 통해 악성코드의 종류를 판별할 수 있으며 더 나아가 악성코드 저자마다 자주 사용하는 모듈을 파악할 수 있다면 악성코드의 저자 식별이 가능하다.

<표 Ⅲ-2> Kernel32.dll의 대표적 API

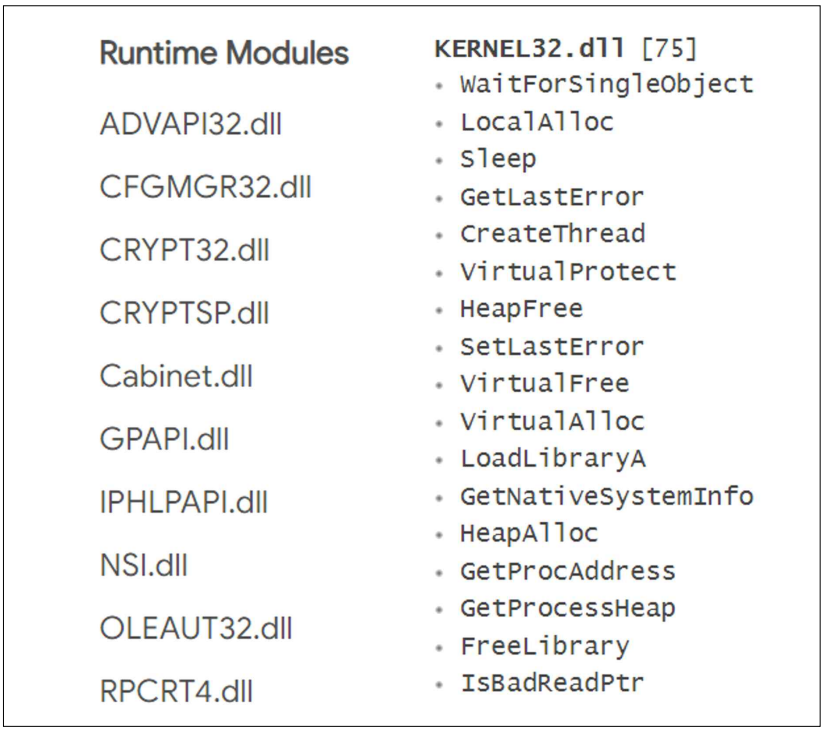
Kernel32.dll API	기능
Sleep	일정 시간동안 프로그램 실행을 일시 중단시키는 API
CreateThread	호출된 프로세스에서 실행할 스레드를 생성하는 API
CloseHandle	다른 API로부터 생성된 핸들을 메모리에서 해제하는 API
GetProcAddress	지정된 dll에서 내보낸 함수나 변수의 주소를 검색하는 API
CreatePipe	새로운 파이프를 만들고, 읽기 또는

	쓰기에 사용되는 핸들을 생성하고 반환하는 API
ReadFile	데이터를 읽는 기능을 하는 API
LCMapStringW	입력하는 문자열을 다른 문자열에 매핑하는 API
IsbadCodePtr	메모리에 대한 프로세스의 접근권한을 변경할 수 있는 API

Kernel32.dll의 API는 총 827개로 구성되어 있으며, 대표적인 API와 기능은 <표 Ⅲ-2>와 같다. 이를 통해 악성코드의 세부적인 동작을 파악할 수 있으며, 악성코드의 저자마다 자주 사용하는 API를 파악할 수 있으면 악성코드 저자 식별이 가능하다.

3. 데이터 수집 및 전처리

본 연구에서는 기존에 저자가 식별된 윈도우즈 기반의 실행 파일 악성코드의 해시 값과 파일 등을 수집하여 자동화 분석을 진행하였다. [그림 Ⅲ-4]는 Lazarus Group의 APT공격중 하나인 AppleJeus의 DorusioUpgrade.exe파일을 자동화 분석하여 런타임 모듈과 Kernel32.dll API를 추출하여 나타낸 것이다. Runtime Modules에서 사용된 모듈을 확인할 수 있고, PE Import에서 사용된 Kernel32.dll의 API 정보를 확인할 수 있다.



[그림 Ⅲ-4] DorusioUpgrade.exe에서 사용된
런타임 모듈과 Kernel32.dll API 목록

이와 같은 방법으로 데이터를 수집하고, 빈도분석을 통해 데이터 셋을 구축한다. <표 Ⅲ-3>수집된 런타임 모듈을 빈도분석 하여 데이터 셋에 포함여부를 확인하는 분석 결과 예시이다.

<표 Ⅲ-3> 빈도분석 결과 예시

런타임 모듈	APT1	APT10	APT29	Winnti	Gorgon Group	Lazarus Group	포함 여부
Advapi32.dll	90%	100%	94%	90%	100%	83%	X
Kernel32.dll	84%	93%	100%	84%	98%	93%	X
Ole32.dll	19%	100%	87%	48%	98%	60%	O
Msvcdrt.dll	9%	98%	15%	14%	35%	36%	O
Wininet.dll	60%	77%	30%	10%	64%	16%	O
Sspicli.dll	2%	43%	15%	2%	64%	43%	O
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

<표 Ⅲ-3>의 내용을 바탕으로 런타임 모듈에서의 식별 특징 추출 및 데이터 셋 구축 과정의 예시를 살펴보면 다음과 같다. Advapi32.dll과 Kernel32.dll의 경우는 6개의 그룹이 자주 사용하는 모듈이다. 이 모듈을 식별 특징으로 정할 경우 분류 알고리즘을 통한 분류가 어려울 수 있기 때문에 데이터 셋에 포함하지 않는다. Ole32.dll의 사용빈도는 APT1은 19%이고, APT10의 경우는 100%이다. 이와 같은 빈도의 차이를 통해 APT10은 악성코드를 제작할 때 Ole32.dll 모듈을 자주 사용하는 것으로 유추할 수 있기 때문에 데이터 셋에 포함한다. 마찬가지로 Msvcdrt.dll 모듈의 사용빈도는 APT1은 9%이고, APT10은 98%이다. 따라서 Msvcdrt.dll모듈은 APT10이 자주 사용하는 모듈이라고 유추할 수 있으므로 데이터 셋에 포함한다. 빈도분석을 통하여 데이터 셋으로 포함하는 기준은 다음과 같다.

1, 적어도 1개 그룹 이상에서 40%이상의 사용빈도

2, 타 그룹과의 30%이상의 빈도차이

런타임 모듈의 타 모듈과 Kernel32.dll API에 대해서도 이와 같은 방법을 적용하여 특징을 추출하고, 데이터 셋을 구축한다. 마지막으로 scikit-learn을 사용하여 분류알고리즘을 적용하는데, scikit-learn은 Python 의 대표적인 머신러닝 라이브러리이다. 하지만 제공하는 분류 알고리즘의 경우 Ole32.dll이나 PeekNamedPipe 등과 같은 문자열 데이터에 대해서는 분류가 불가능하기 때문에 문자열 데이터를 숫자로 변환하는 작업이 필요하며 이를 위해 One-Hot Encoding을 사용한다. [그림 Ⅲ-5]는 기존의 데이터 셋을 One-Hot Encoding 한 예시이다.

ID	Group	Module1	Module2	Module3	..
1	Lazarus Group	Advapi32.dll	Ole32.dll	Sspicli.dll	..
2	Lazarus Group	Advapi32.dll	Kernel32.dll	Ole32.dll	..
3	Lazarus Group	Advapi32.dll	Kernel32.dll	Msvcdrt.dll	..
4	APT1	Advapi32.dll	Kernel32.dll	Winine.dll	..
5	APT1	Advapi32.dll	Kernel32.dll	Winine.dll	..
:	:	:	:	:	

ID	Group	Advapi32.dll	Kernel32.dll	Ole32.dll	Sspicli.dll	..
1	Lazarus Group	1	0	1	1	..
2	Lazarus Group	1	1	1	0	..
3	Lazarus Group	1	1	0	0	..
4	APT1	1	1	0	0	..
5	APT1	1	1	0	0	..
:	:	:	:	:	:	

[그림 Ⅲ-5] One-Hot Encoding 예시

IV. 실험 및 결과분석

1. 실험 환경

본 연구의 실험 환경은 <표 Ⅳ-1>와 같다. CPU는 Intel(R) Core(TM) I7-10700 CPU @ 2.90GHz을 사용하였으며 NVIDIA GeForce RTX 3070의 GPU, 32.0GB 의 DDR4 메모리를 사용하였다. 운영체제는 Windows 10 Home 64bit 모델을 사용하였고, 실험은 Python 3.8.6 버전을 사용하여 진행하였다.

<표 Ⅳ-1> 실험 환경

CPU	Intel(R) Core(TM) I7-10700 CPU @ 2.90GHz
GPU	NVIDIA GeForce RTX 3070
RAM	32.0GB
OS	Windows 10 Home 64bit

2. 실험 방법

[그림 Ⅳ-1] 실험 방법

실험 방법은 다음과 같다.

첫째. 실험 데이터는 MITRE사의 ATT&CK 프레임워크와 CIBERSECURITY & INFRASTRUCTURE SECURITY AGENCY(CISA) 보고서, 악성코드 저자 식별 연구에 사용된 데이터를 활용하였다. 데이터는 윈도우즈 기반 실행파일이며, 악성코드 파일 및 해시 값을 수집하였다.

둘째. 수집된 데이터는 자동화 분석 툴인 Malware.com과 BSA(Buster Sandbox Analyzer), Virus Total을 활용해 분석을 실행하였다.

셋째. 런타임 모듈과 Kernel32.dll API를 추출하여 본 연구에서 제안한 기준을 가진 빈도분석을 진행 한 후 데이터 셋을 구성하였다. 구성된 데이터는 <표 Ⅳ-2>와 같다. 런타임 모듈의 경우는 총 266개의 데이터 셋을 가지며, Kernel32.dll API의 경우는 총 279개의 데이터 셋을 가진다. 데이터의 편향성을 해결하기 위해 런타임 모듈은 최소 11%부터 최대 20%의 데이터 비율을 설정하였고, Kernel32.dll API는 최소 12%부터 최대 20%의 데이터 비율을 설정

하였다.

<표 Ⅳ-2> 실험 데이터

런타임 모듈	데이터 셋 개수	비율	Kernel32.dll API	데이터 셋 개수	비율
APT 1	52	20%	APT 1	49	18%
APT 10	53	20%	APT 10	51	18%
Lazarus Group	30	11%	Lazarus Group	56	20%
Winnti	50	19%	Winnti	32	12%
Gorgon Group	48	18%	Gorgon Group	43	15%
APT29	33	12%	APT29	48	17%

넷째. 구성된 데이터 셋을 통해 One-Hot Encoding을 진행하였다.

다섯째. scikit-learn 라이브러리에서 제공하는 6개의 분류 알고리즘을 통해 분류를 진행한다. 실험에 사용된 분류 알고리즘은 <표 Ⅳ-3>과 같다.

<표 Ⅳ-3> 실험에 사용된 알고리즘

머신러닝 분류 알고리즘	설명
k-NN	기존 데이터의 분포 중에서 입력된 데이터로부터 인접한 k개의 데이터를 찾아 k개의 데이터 중 가장 많이 인접한 값으로 데이터를 분류하는 알고리즘
SVM	기존의 데이터를 이용하여 결정 경계선을 구하고, 데이터의 구역을 나눈 후 새로운 데이터의 위치에 따라 분류하는 알고리즘
Decision Tree	데이터의 특징을 바탕으로 분리해 트리를 생성하고, 입력 데이터를 트리에 통과시켜 분류하는 알고리즘
Naive Bayes	데이터를 독립적인 사건으로 가정한 후 Bayes 정리에 대입시켜 가장 높은 확률의 레이블로 분류하는 알고리즘
Adaptive Boosting	1차 분류를 수행하고, 오류 데이터에 대해 가중치를 높인 강한 분류기를 만들어내는 알고리즘
Gradient Boosting	경사하강법을 통해 약한 분류리가 만든 잔여 오차에 대해 새로운 분류기를 학습시키는 알고리즘

여섯째. 분류 알고리즘을 사용하여 정확도(Accuracy), 정밀도(Precision), 재현율(Recall), F1 Score를 구하여 각 머신러닝 모델의 성능을 평가하고 데이터에 맞는 모델을 선정한다. 전체적인 실험 방법은 [그림 Ⅳ-5]와 같다.

<표 Ⅳ-4>와 같이 분류 결과와 실제 정답으로 나뉘어져 있을 때, 정확도와 정밀도, 재현율, F1 Score를 구하는 공식은 <표 Ⅳ-4>의 True Positive(TP), False Positive(FP), False Negative(FN), True Negative(TN)를 통해 나타낼 수 있다.

<표 Ⅳ-4> 머신러닝 분류 결과

		실제 정답	
		True	False
분류 결과	True	True Positive	False Positive
	False	False Negative	True Negative

정확도(Accuracy)는 가장 간단하게 성능을 측정하는 방법으로써 정확도를 구하는 공식은 다음과 같다.

$$\text{정확도} = \frac{TP+TN}{TP+FP+FN+TN}$$

정밀도(Precision)는 모델이 True라고 예측한 결과 중에서 실제 True 데이터의 비율을 측정하는 방법으로써 정밀도를 구하는 공식은 다음과 같다.

$$\text{정밀도} = \frac{TP}{TP+FP}$$

재현율(Recall)은 실제 True인 데이터 중에서 모델이 True라고 예측한 비율을 측정하는 방법으로써 재현율을 구하는 공식은 다음과 같다.

$$\text{재현율} = \frac{TP}{TP+FN}$$

F1 Score는 정밀도와 재현율의 조화평균을 나타내는 값이며, F1 Score를 구하는 공식은 다음과 같다.

$$\text{F1 Score} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

3. 실험 결과

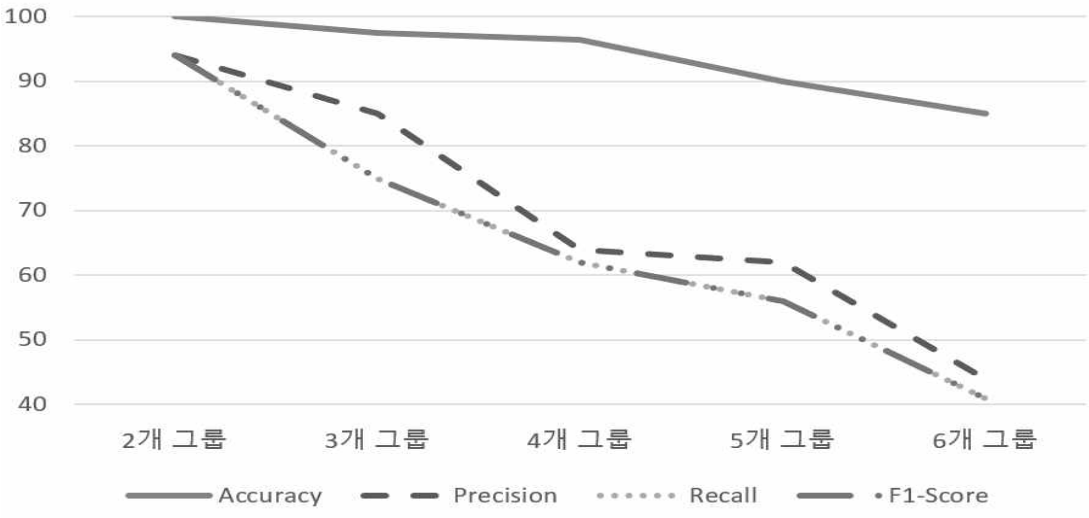
본 실험결과는 그룹이 2개일 때부터 6개일 때까지의 런타임 모듈과 Kernel32.dll API 데이터에 대해 6가지의 분류 알고리즘을 통해 정확도(Accuracy)와 정밀도(Precision), 재현율(Recall) F1 Score를 보여준다.

먼저 런타임 모듈에 k-NN알고리즘을 적용했을 때의 결과는 <표 Ⅳ-5>와 같다.

<표 Ⅳ-5> 런타임 모듈 데이터 k-NN알고리즘 적용 결과

	2개 그룹	3개 그룹	4개 그룹	5개 그룹	6개 그룹
하이퍼 파라미터	k=4	k=3	k=5	k=3	k=3
정확도	100%	97.5%	96.4%	90%	85%
정밀도	94%	85%	64%	62%	44%
재현율	94%	75%	62%	56%	41%
F1 Score	94%	75%	62%	56%	41%

식별하는 악성코드 저자그룹이 2개일 경우에는 k값이 4일 때 100%의 정확도를 보여주고, 정밀도, 재현율, F1 Score는 94%를 나타낸다. 저자 그룹이 많아질수록 정확도가 떨어지는데 6개의 저자 그룹일 경우에는 정확도가 85%, 정밀도, 재현율, F1 Score는 40%대로 예측된다. k-NN알고리즘의 경우 정확도 부분에서 기존의 연구 결과보다 높은 성능을 보이지만 정밀도, 재현율, F1 Score에서 낮은 성능을 보이기 때문에 종합적으로 봤을 때 본 연구에서 제시한 방법을 통한 악성코드 저자 식별에 대해 성능이 좋지 못하다.



[그림 Ⅳ-2] 런타임 모듈 데이터 k-NN 알고리즘 적용 결과 그래프

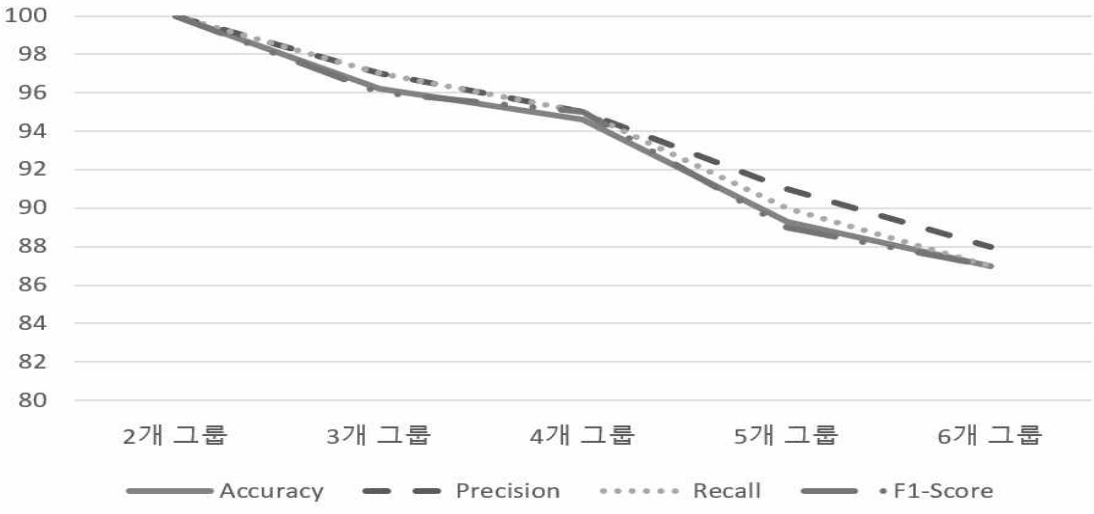
[그림 Ⅳ-2]는 k-NN 알고리즘을 적용하였을 때의 그룹의 수에 따른 정확도, 정밀도, 재현율, F1 Score를 그래프로 표현한 그림이다. 정확도와 정밀도, 재현율, F1 Score와의 괴리감이 있는 것을 볼 수 있다.

<표 Ⅳ-6> 런타임 모듈 데이터 SVM알고리즘 적용 결과

	2개 그룹	3개 그룹	4개 그룹	5개 그룹	6개 그룹
하이퍼 파라미터	C=1 gamma=0.1	C=10 gamma=0.1	C=10 gamma=0.1	C=1 gamma=1	C=1 gamma=0.1
정확도	100%	96.2%	94.6%	89.3%	87%
정밀도	100%	97%	95%	91%	88%
재현율	100%	97%	95%	90%	87%
F1 Score	100%	96%	95%	89%	87%

<표 Ⅳ-6>는 SVM 알고리즘에 대해 악성코드 저자 그룹의 수에 따라 정확도, 정밀도, 재현율, F1 Score를 나타낸 표이다. 하이퍼 파라미터는 Greedy 알고리즘을 사용하여 선정하였다. k-NN알고리즘과 비교했을 때, 저자 그룹의 수가 3, 4, 5개의 그룹일 때는 정확도가 낮지만, 6개의 그룹일 때는 정확도가 높은 것을

볼 수 있다. 그리고 정밀도, 재현율, F1 Score도 정확도와 비슷한 성능을 보여주고 있기 때문에 본 연구에서 제시한 방법을 통한 악성코드 저자 식별에 좋은 성능을 보인다.



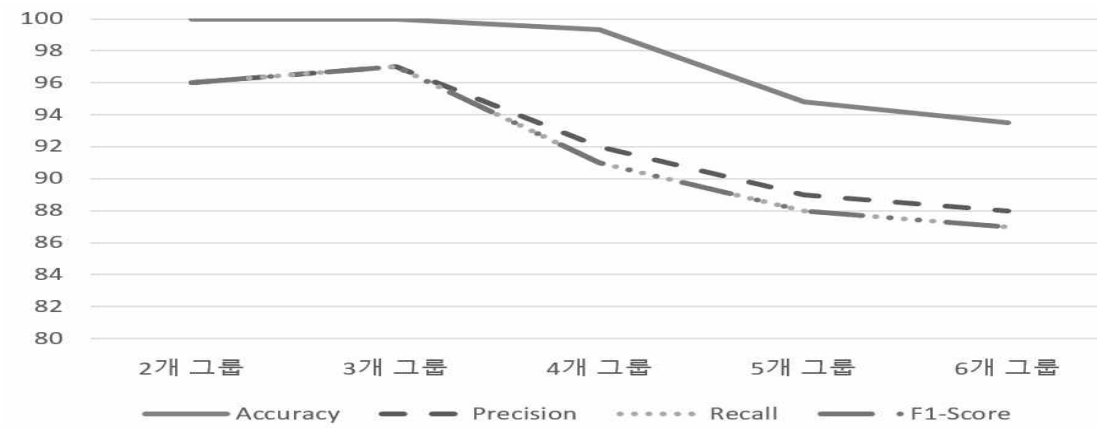
[그림 Ⅳ-3] 런타임 모듈 데이터 SVM 알고리즘 적용 결과 그래프

[그림 Ⅳ-3]은 SVM 알고리즘을 적용하였을 때의 그룹의 수에 따른 정확도, 정밀도, 재현율, F1 Score를 그래프로 표현한 그림이다. k-NN과는 다르게 정확도, 정밀도, 재현율, F1 Score가 비슷하게 분포되어 있는 것을 볼 수 있다.

<표 Ⅳ-7> 런타임 모듈 데이터 Decision Tree알고리즘 적용 결과

	2개 그룹	3개 그룹	4개 그룹	5개 그룹	6개 그룹
하이퍼 파라미터	Max_depth= default min_samples _split=2 min_samples _leaf=1 random_stat e=100	Max_depth= default min_samples _split=2 min_samples _leaf=1 random_stat e=100	Max_depth= default min_samples _split=2 min_samples _leaf=1 random_stat e=100	Max_depth= default min_samples _split=2 min_samples _leaf=1 random_stat e=100	Max_depth= default min_samples _split=2 min_samples _leaf=1 random_stat e=100
정확도	100%	100%	99.3%	94.8%	93.5%
정밀도	96%	97%	92%	89%	88%
재현율	96%	97%	91%	88%	87%
F1 Score	96%	97%	91%	88%	87%

<표 Ⅳ-7>은 Decision Tree 알고리즘을 적용한 결과이다. 하이퍼 파라미터에서 Max_depth의 값을 제한적으로 두는 것 보다 default값으로 사용하는 것이 정확도 측면에서 좋은 결과를 보였기 때문에 사용하였고, min_sample_split과 min_sample_leaf의 값을 제한을 두어서 과적합을 방지하였다. 전체적인 정확도는6개 그룹일 때에도 90%를 넘기는 준수한 성능을 보여주었지만, 정밀도, 재현율, F1 Score의 경우는 악성코드 저자 그룹의 수가 높아질수록 수치가 낮아지는 결과를 보여주었다.



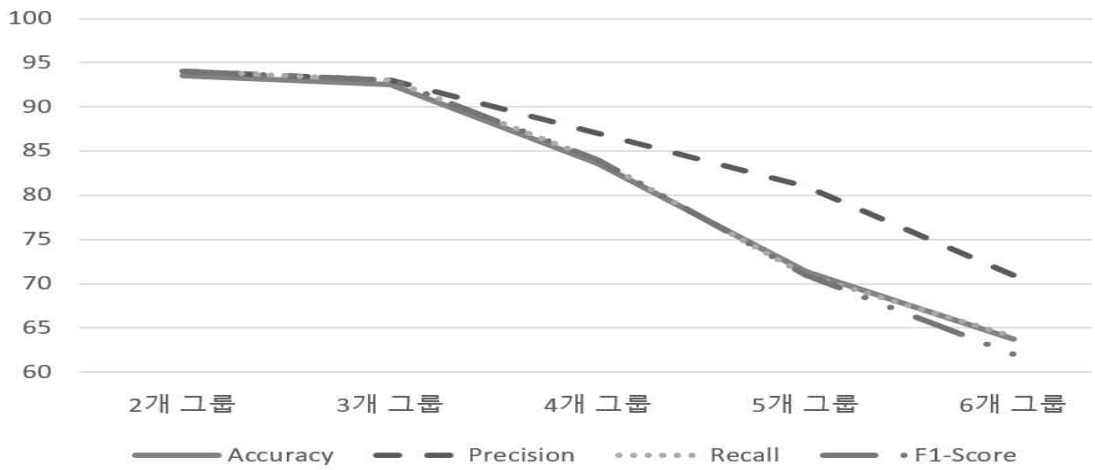
[그림 Ⅳ-4] 런타임 모듈 데이터 Decision Tree 알고리즘 적용 결과 그래프

[그림 Ⅳ-4]는 Decision Tree 알고리즘을 적용하였을 때 그룹의 수에 따른 정확도, 정밀도, 재현율, F1 Score를 그래프로 표현한 그림이다. 정확도는 높게 형성되지만 정밀도, 재현율, F1 Score는 정확도에 비해 낮게 형성된 것을 볼 수 있다.

<표 Ⅳ-8> 런타임 모듈 데이터 Naive Bayes알고리즘 적용 결과

	2개 그룹	3개 그룹	4개 그룹	5개 그룹	6개 그룹
하이퍼 파라미터	default	default	default	default	default
정확도	93.5%	92.5%	83.6%	71.4%	63.7%
정밀도	94%	93%	87%	81%	71%
재현율	94%	93%	84%	71%	64%
F1 Score	94%	93%	84%	71%	62%

<표 Ⅳ-8>는 Naive Bayes알고리즘을 적용한 결과이다. Naive Bayes알고리즘의 특성 상 Kerner32.dll이나 davapi.dll과 같은 자주 사용하는 런타임 모듈의 경우도 중요하고 독립적으로 가정하기 때문에 악성코드 저자 그룹이 2개에서 3개까지는 준수한 성능을 보이지만 4개 그룹 이상으로 가면서는 정확도가 많이 떨어지는 단점을 보인다.



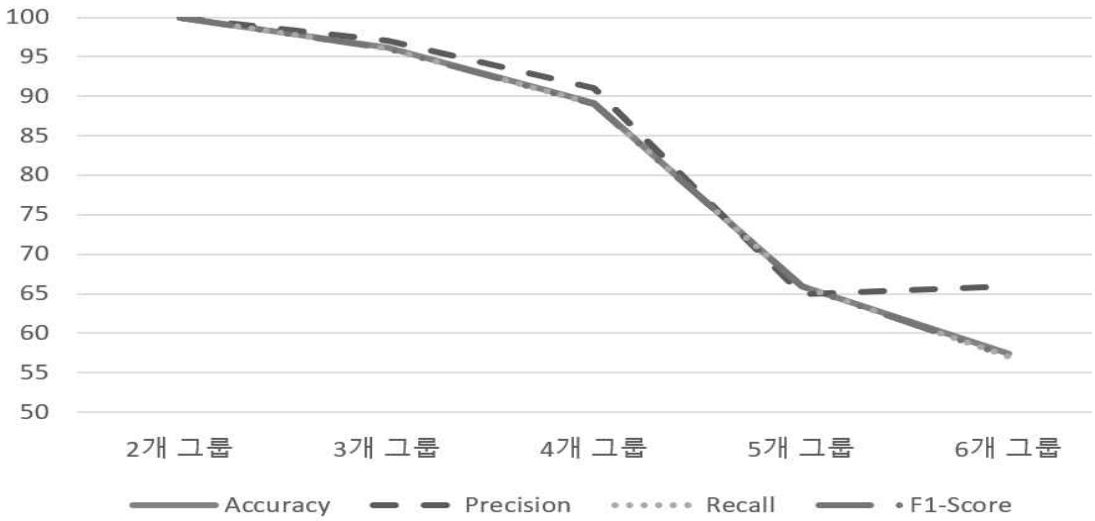
[그림 Ⅳ-5] 런타임 모듈 데이터 Naive Bayes알고리즘 적용 결과 그래프

[그림 Ⅳ-5]는 Naive Bayes 알고리즘을 적용하였을 때 그룹 수에 따른 정확도, 정밀도, 재현율, F1 Score를 그래프로 표현한 그림이다. 4개 그룹이상 일수록 정확도, 정밀도, 재현율, F1 Score가 낮아지는 것을 볼 수 있고, 정밀도와 정확도, 재현율, F1 Score의 차이가 발생하는 것을 볼 수 있다.

<표 Ⅳ-9> 런타임 모듈 데이터 Adaptive Boosting 알고리즘 적용 결과

	2개 그룹	3개 그룹	4개 그룹	5개 그룹	6개 그룹
하이퍼 파라미터	base_estimators = None n_estimators = 50 learning_rate = 1.0	base_estimators = None n_estimators = 100 learning_rate = 1.0	base_estimators = None n_estimators = 100 learning_rate = 0.5	base_estimators = None n_estimators = 100 learning_rate = 0.1	base_estimators = None n_estimators = 100 learning_rate = 0.1
정확도	100%	96.2%	89.2%	65.9%	57.4%
정밀도	100%	97%	91%	65%	66%
재현율	100%	96%	89%	66%	57%
F1 Score	100%	96%	89%	66%	57%

<표 Ⅳ-9>은 Adaptive Boosting알고리즘을 적용한 결과이다. base_estimators = None는 학습에 사용하는 알고리즘을 Decision Tree(Max_depth=1)를 적용하기 때문에 None으로 설정하였고, n_estimators와 learning_rate는 Greedy 알고리즘을 통해 최적의 값을 설정하였다. 정확도, 정밀도, 재현율, F1 Score는 Naive Bayes 알고리즘과 마찬가지로 4개 그룹 이상부터 정확도가 낮아지기 시작하고, 6개 그룹일 경우는 본 연구에서 사용한 분류 알고리즘 중에서 가장 낮은 성능을 보여주었다.



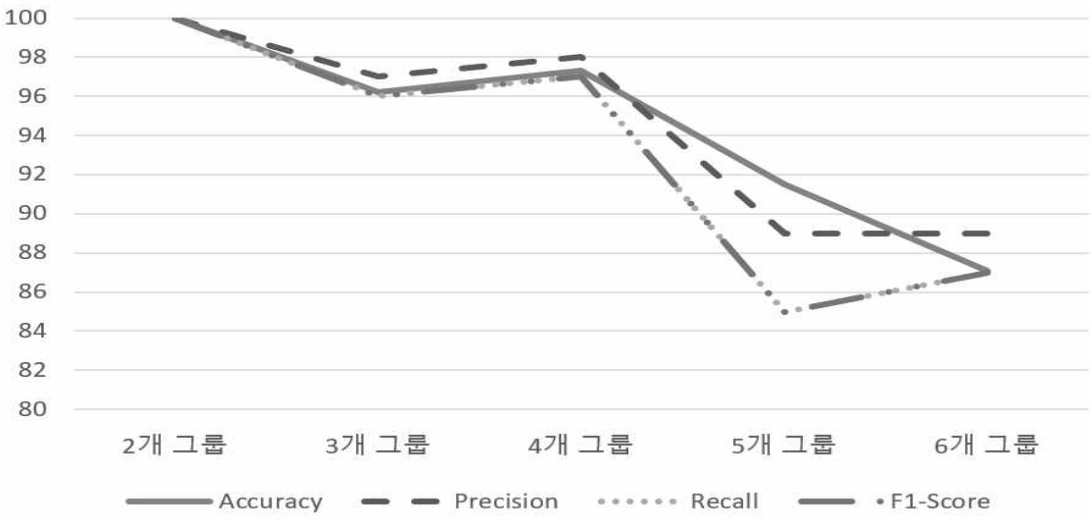
[그림 Ⅳ-6] 런타임 모듈 데이터 Adaptive Boosting 알고리즘 적용 결과 그래프

[그림 Ⅳ-6]은 Adaptive Boosting 알고리즘을 적용하였을 때 그룹 수에 따른 정확도, 정밀도, 재현율, F1 Score를 그래프로 표현한 그림이다. 정확도, 정밀도, 재현율, F1 Score의 수치가 비슷하다가 6개 그룹일 때 차이가 발생하는 것을 볼 수 있다.

<표 Ⅳ-10> 런타임 모듈 데이터 Gradient Boosting알고리즘 적용 결과

	2개 그룹	3개 그룹	4개 그룹	5개 그룹	6개 그룹
하이퍼 파라미터	learning_rate=0.1 n_estimators=200	learning_rate=0.05 n_estimators=200	learning_rate=0.1 n_estimators=200	learning_rate=0.05 n_estimators=200	learning_rate=0.05 n_estimators=100
정확도	100%	96.2%	97.3%	91.5%	87.1%
정밀도	100%	97%	98%	89%	89%
재현율	100%	96%	97%	85%	87%
F1 Score	100%	96%	97%	85%	87%

<표 Ⅳ-10>는 Gradient Boosting을 적용한 결과이다. Adaptive Boosting과 마찬가지로 Greedy 알고리즘을 사용하여 하이퍼 파라미터 값을 설정하였고, Tree에 대한 하이퍼 파라미터는 Default로 설정하였다. 정확도 측면에서는 Dicision Tree보다 낮지만 정밀도, 재현율, F1 Score와 정확도와의 차이가 많이 나지 않기 때문에 본 연구에서 제시한 방법을 통한 악성코드 저자 그룹 식별에 좋은 성능을 보인다.

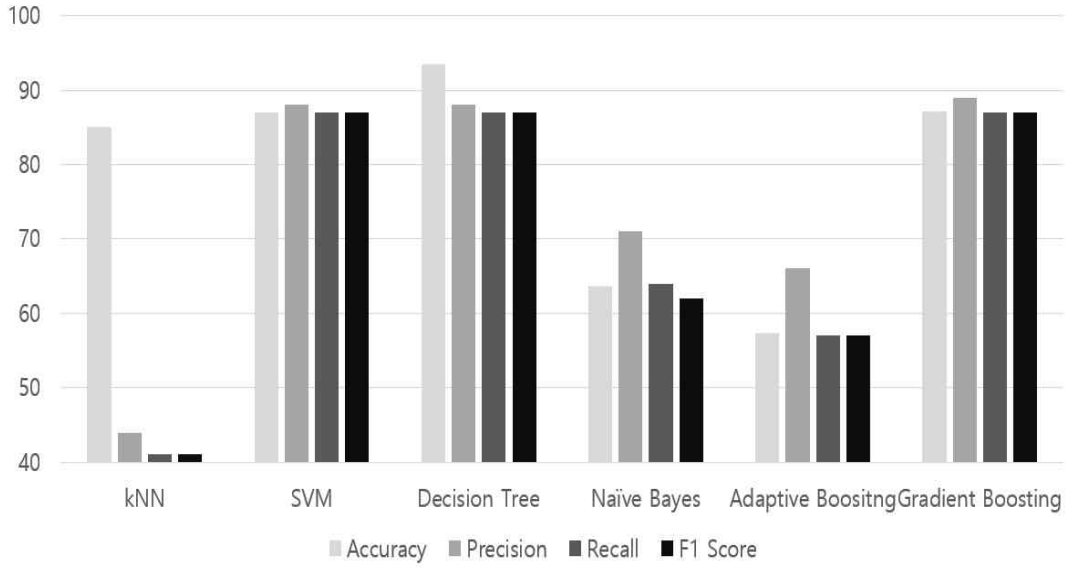


[그림 Ⅳ-7] 런타임 모듈 데이터 Gradient Boosting알고리즘 적용 결과 그래프

[그림 Ⅳ-7]은 Gradient Boosting 알고리즘을 적용하였을 때 그룹 수에 따른 정확도, 정밀도, 재현율, F1 Score를 그래프로 표현한 그림이다. 정확도, 정밀도, 재현율, F1 Score의 값이 차이가 많이 나지 않는 것을 볼 수 있다.

[그림 Ⅳ-8]악성코드 저자그룹의 수가 6개 일 때의 정확도, 정밀도, 재현율, F1 Score를 비교한 그래프이다. k-NN알고리즘의 경우는 정확도는 높지만 정밀도, 재현율, F1 Score에서 낮은 수치를 보이고, Naive Bayes, Adaptive Boositng알고리즘은 정확도가 낮기 때문에 성능이 좋지 못하다. Decision Tree 알고리즘은 전체적인 성능은 높게 나오지만 정확도와 정밀도, 재현율, F1 Score와의 차이가 발생하고 있지만 SVM, Gradient Boosting 알고리즘의 경우는 준

수한 정확도를 가지면서도 나머지 수치와의 차이가 거의 없기 때문에 SVM알고리즘과 Gradient Boositng알고리즘이 본 연구에서 제안한 방법을 통해 악성코드 저자 그룹 식별에 좋은 알고리즘이다.



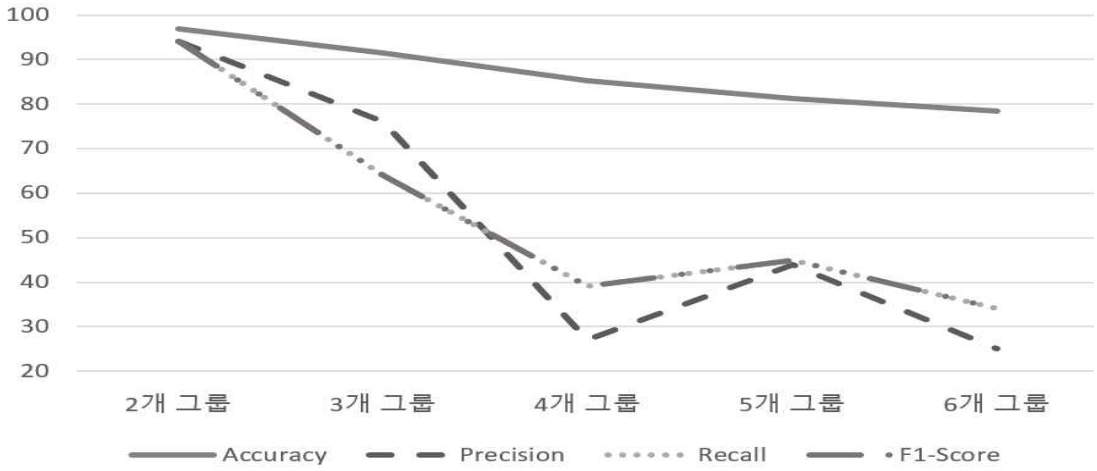
[그림 Ⅳ-8] 런타임 모듈 데이터를 6명의 저자일 때 적용한 알고리즘에 따른 결과

런타임 모듈은 위와 같은 결과가 나왔다. 다음으로는 Kerner32.dll API에 대한 실험 결과이다. 실험 결과는 런타임 모듈과 같이 정확도, 재현율, 정밀도 F1 Score에 대해 그룹의 개수만큼 6개의 분류 알고리즘을 적용시킨다.

<표 Ⅳ-11> Kernel32.dll API 데이터 k-NN알고리즘 적용 결과

	2개 그룹	3개 그룹	4개 그룹	5개 그룹	6개 그룹
하이퍼 파라미터	k=3	k=3	k=5	k=5	k=8
정확도	96.9%	91.5%	85.2%	81.3%	78.4%
정밀도	94%	76%	27%	44%	25%
재현율	94%	64%	39%	45%	34%
F1 Score	94%	64%	39%	45%	34%

<표 Ⅳ-11>은 Kernel32.dll API 데이터에 k-NN알고리즘을 적용한 결과이다. 2개 그룹일 때 96.9%부터 6개 그룹일 때 78.4%의 정확도를 보인다. 런타임 모듈 데이터와 비교할 때 정확도가 다소 낮아진 것을 볼 수 있다. 3개 그룹 이상일 때의 정밀도와 재현율, F1 Score가 정확도와 많은 차이가 발생한다.



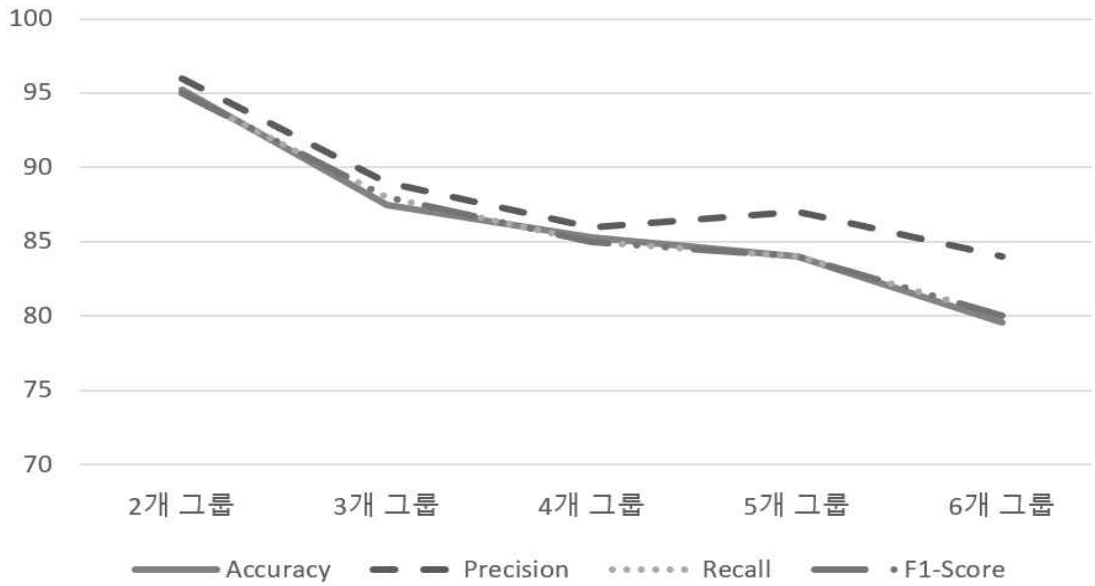
[그림 Ⅳ-9] Kernel32.dll API 데이터 k-NN알고리즘 적용 결과 그래프

[그림 Ⅳ-9]는 k-NN 알고리즘을 적용하였을 때 그룹 수에 따른 정확도, 정밀도, 재현율, F1 Score를 그래프로 표현한 그림이다. 런타임 모듈과 마찬가지로 각 수치마다 차이가 많이 발생하는 것을 볼 수 있다.

<표 Ⅳ-12> Kernel32.dll API 데이터 SVM알고리즘 적용 결과

	2개 그룹	3개 그룹	4개 그룹	5개 그룹	6개 그룹
하이퍼 파라미터	C=10 gamma=0.1	C=100 gamma=0.01	C=1000 gamma=0.01	C=100 gamma=0.1	C=1 gamma=1
정확도	95.2%	87.5%	85.3%	84%	79.6%
정밀도	96%	89%	86%	87%	84%
재현율	95%	88%	85%	84%	80%
F1 Score	95%	88%	85%	84%	80%

<표 Ⅳ-12>은 SVM알고리즘을 적용한 결과이다. 하이퍼 파라미터는 Greedy 알고리즘을 사용하여 적용하였고, 정확도는 2개 그룹일 때 95.2%부터 6개 그룹일 때79.6%로 보인다. 런타임 모듈 데이터와 비교하였을 때 낮은 정확도를 보인다. 하지만 k-NN알고리즘과 비교하였을 경우는 정확도뿐만 아니라 정밀도, 재현율, F1 Score에서도 좋은 결과를 가진다.



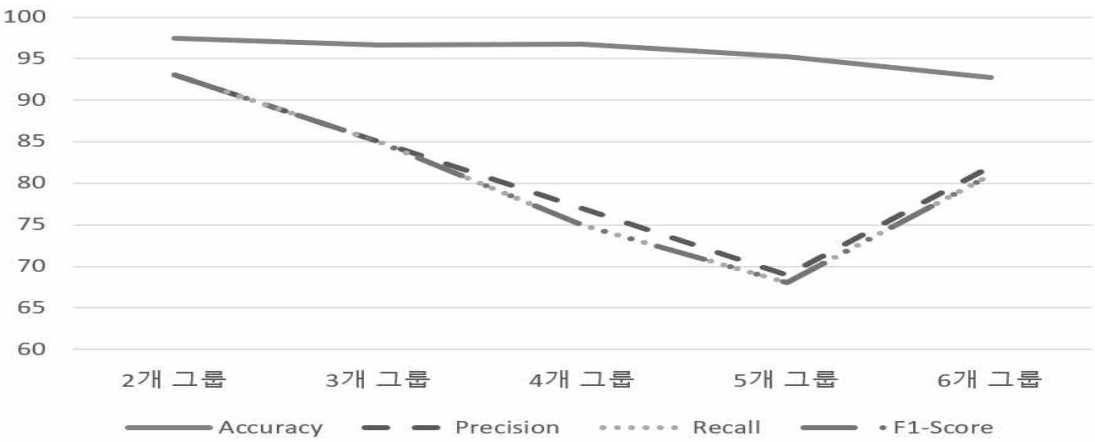
[그림 Ⅳ-10] Kernel32.dll API 데이터 SVM알고리즘 적용 결과 그래프

[그림 Ⅳ-10]은 SVM 알고리즘을 적용하였을 때 그룹 수에 따른 정확도, 정밀도, 재현율, F1 Score를 그래프로 표현한 그림이다. 각 수치별로 차이가 많이 발생하지 않는 것을 볼 수 있다.

<표 Ⅳ-13> Kernel32.dll API 데이터 Decision Tree알고리즘 적용 결과

	2개 그룹	3개 그룹	4개 그룹	5개 그룹	6개 그룹
하이퍼 파라미터	Max_depth= default min_samples _split=2 min_samples _leaf=1 random_stat e=100	Max_depth= default min_samples _split=2 min_samples _leaf=1 random_stat e=100	Max_depth= default min_samples _split=2 min_samples _leaf=1 random_stat e=100	Max_depth= default min_samples _split=2 min_samples _leaf=1 random_stat e=100	Max_depth= default min_samples _split=2 min_samples _leaf=1 random_stat e=100
정확도	97.4%	96.6%	96.7%	95.2%	92.7%
정밀도	93%	85%	77%	69%	82%
재현율	93%	85%	75%	68%	81%
F1 Score	93%	85%	75%	68%	81%

<표 Ⅳ-13>은 Decision Tree알고리즘을 적용한 결과이다. 하이퍼 파라미터는 런타임 모듈과 같이 정확도를 높이기 위해 Max_depth는 default로 설정하였고, 과적합을 막기 위해 min_sample_split와 min_sample_leaf에 제한을 두었다. Kernel32.dll API를 적용한 다른 알고리즘에 비해 6개 그룹일 때도 92%의 정확도를 보여주는 준수한 성능을 가졌지만 4개 그룹이상일 때 정밀도, 재현율, F1 Score가 정확도와의 차이가 많이 발생한다.



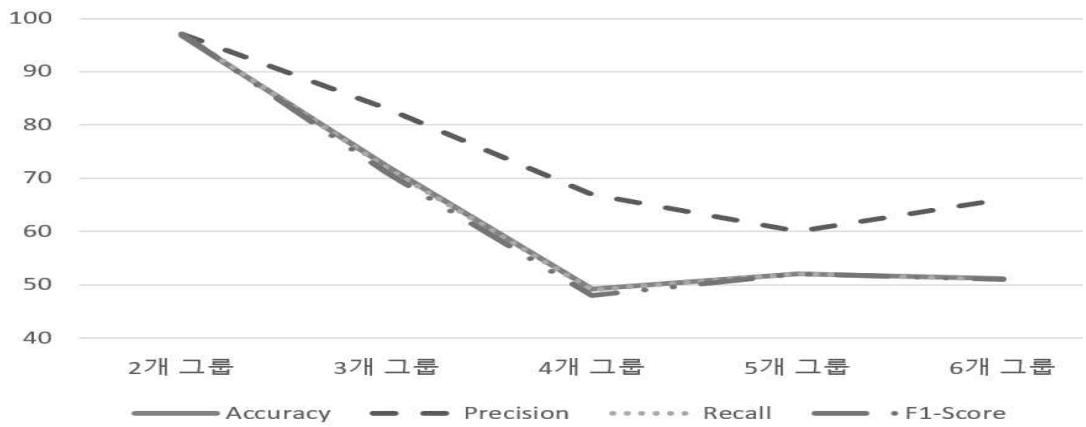
[그림 Ⅳ-11] Kernel32.dll API 데이터 Decision Tree알고리즘 적용 결과 그래프

[그림 Ⅳ-11]은 Decision Tree 알고리즘을 적용하였을 때 그룹 수에 따른 정확도, 정밀도, 재현율, F1 Score를 그래프로 표현한 그림이다. 정확도는 높지만, 각 수치별 차이가 많이 발생하는 것을 볼 수 있다.

<표 Ⅳ-14> Kernel32.dll API 데이터 Naive Bayes 알고리즘 적용 결과

	2개 그룹	3개 그룹	4개 그룹	5개 그룹	6개 그룹
하이퍼 파라미터	default	default	default	default	default
정확도	96.8%	72.3%	49.2%	52%	51.2%
정밀도	97%	83%	67%	60%	66%
재현율	97%	82%	49%	52%	51%
F1 Score	97%	81%	48%	52%	51%

<표 Ⅳ-14>는 Naive Bayes를 적용한 결과이다. 런타임 모듈 데이터를 적용하였을 때보다 데이터의 분류 기준이 많기 때문에 정확도가 많이 낮아진 것을 볼 수 있다. 특이점은 모든 부분에서 정확도보다 정밀도와 재현율, F1 Score가 높지만 유의미하지 않은 수치이다.



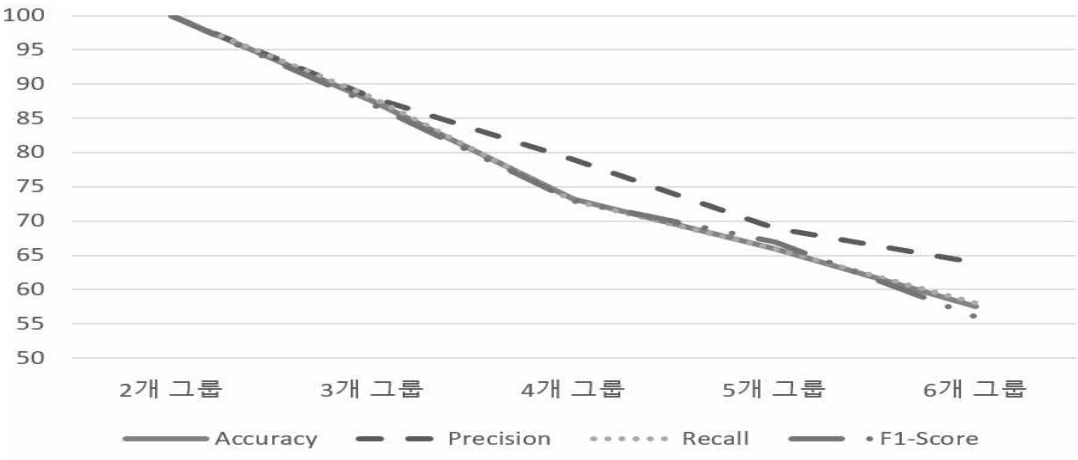
[그림 Ⅳ-12] Kernel32.dll API 데이터 Naive Bayes 알고리즘 적용 결과 그래프

[그림 Ⅳ-12]는 Naive Bayes 알고리즘을 적용하였을 때 그룹 수에 따른 정확도, 정밀도, 재현율, F1 Score를 그래프로 표현한 그림이다. 4개 그룹 이상부터 모든 수치가 낮아지는 것을 볼 수 있다.

<표 Ⅳ-15> Kernel32.dll API 데이터 Adaptive Boosting알고리즘 적용 결과

	2개 그룹	3개 그룹	4개 그룹	5개 그룹	6개 그룹
하이퍼 파라미터	base_estimators = None n_estimators = 50 learning_rate = 1.0	base_estimators = None n_estimators = 50 learning_rate = 0.5	base_estimators = None n_estimators = 100 learning_rate = 0.01	base_estimators = None n_estimators = 50 learning_rate = 0.1	base_estimators = None n_estimators = 100 learning_rate = 0.01
정확도	100%	87.5%	73.2%	66%	57.6%
정밀도	100%	88%	79%	69%	64%
재현율	100%	88%	73%	66%	58%
F1 Score	100%	87%	73%	67%	56%

<표 Ⅳ-15>은 Adaptive Boosting를 적용한 결과이다. base_estimators는 None로 설정하여 Decision Tree(Max_depth=1)를 적용하였고, n_estimators와 learning_rate는 Greedy 알고리즘을 적용하여 설정하였다. 정확도는 2개 그룹일 때 100%와 6개 그룹일 때 57.6%로 저자 그룹의 수에 따라 차이가 많이 발생했다.



[그림 Ⅳ-13] Kernel32.dll API 데이터 Adaptive Boosting알고리즘 적용 결과 그래프

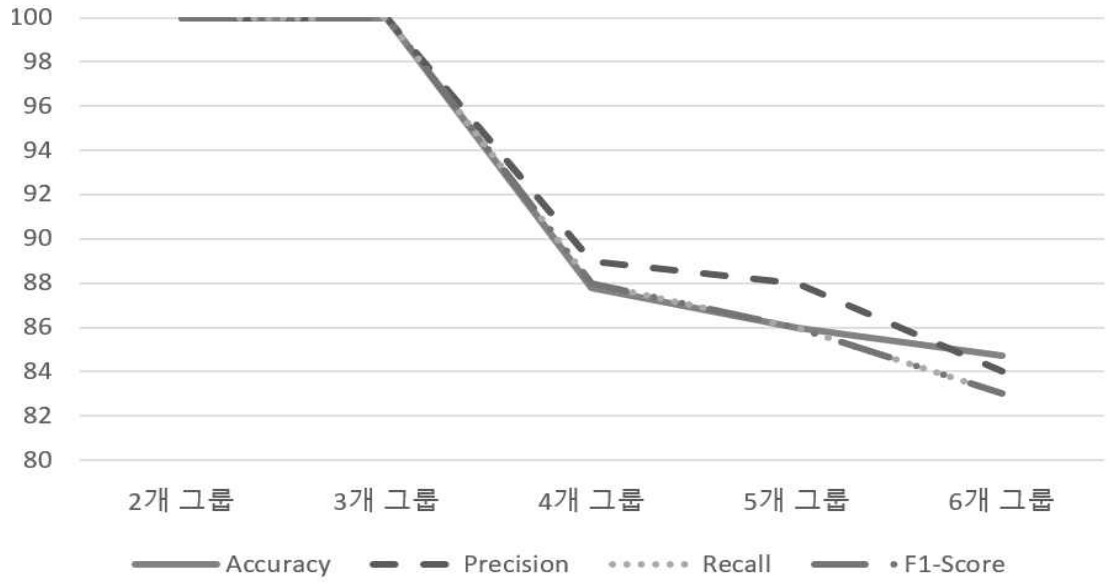
[그림 Ⅳ-13]은 Adaptive Boosting 알고리즘을 적용하였을 때 그룹 수에 따른 정확도, 정밀도, 재현율, F1 Score를 그래프로 표현한 그림이다. 각 수치별 차이는 많이 발생하지 않지만, 그룹의 수가 많아질수록 수치가 많이 낮아지는 것을 볼 수 있다.

<표 Ⅳ-16> Kernel32.dll API 데이터 Gradient Boosting알고리즘 적용 결과

	2개 그룹	3개 그룹	4개 그룹	5개 그룹	6개 그룹
하이퍼 파라미터	learning_rate=0.1 n_estimators=100	learning_rate=0.05 n_estimators=100	learning_rate=0.05 n_estimators=100	learning_rate=0.05 n_estimators=100	learning_rate=0.05 n_estimators=100
정확도	100%	100%	87.8%	86%	84.7%
정밀도	100%	100%	89%	88%	84%
재현율	100%	100%	88%	86%	83%
F1 Score	100%	100%	88%	86%	83%

<표 Ⅳ-16>는 Gradient Boosting을 적용한 결과이다. 하이퍼 파라미터는

Greedy 알고리즘을 적용하여 설정하였고, 2개 그룹일 때 100%의 정확도와 6개 그룹일 때 84.7%의 정확도를 가지며, 정확도, 정밀도, 재현율, F1 Score의 차이가 많이 발생하지 않았다.

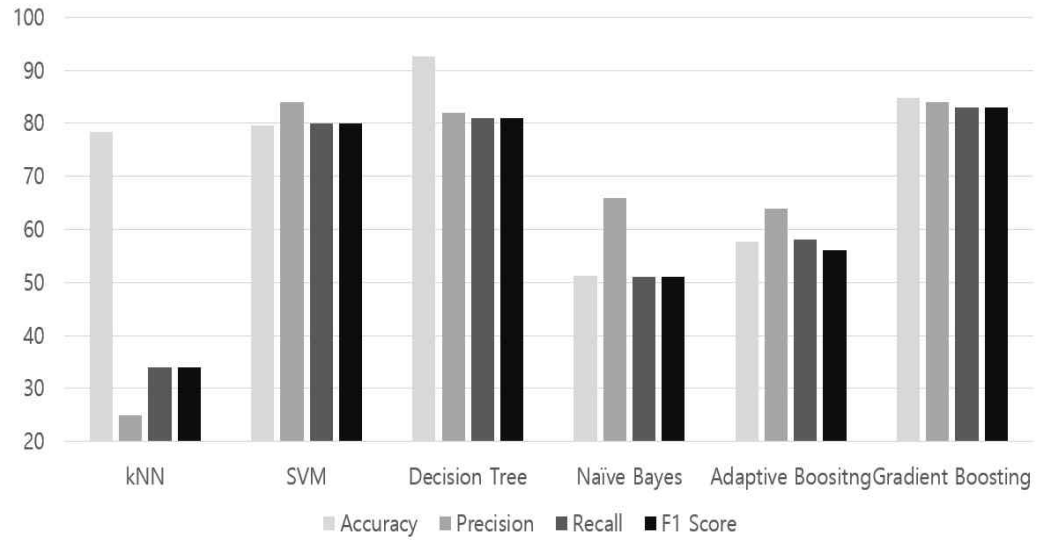


[그림 Ⅳ-14] Kernel32.dll API 데이터 Gradient Boosting알고리즘 적용 결과 그래프

[그림 Ⅳ-14]는 Gradient Boosting 알고리즘을 적용하였을 때 그룹 수에 따른 정확도, 정밀도, 재현율, F1 Score를 그래프로 표현한 그림이다. 각 수치별 차이가 많이 발생하지 않은 것을 볼 수 있다.

[그림 Ⅳ-15]는 6명의 악성코드 저자 그룹일 때의 각 분류 알고리즘을 적용한 결과이다. k-NN알고리즘의 경우 정확도는 80%에 근접하지만, 정밀도, 재현율, F1 Score가 20~30%이므로 좋지 못한 성능을 보이고, Naive Bayes알고리즘과 Adaptive Boosting알고리즘은 정확도와 정밀도, 재현율, F1 Score는 비슷한 %를 보이고 있지만, 낮기 때문에 성능이 좋지 못하다. SVM알고리즘과 Decision알고리즘, Gradient Boosting알고리즘의 경우는 정확도가 높게 나타나는데, Decision Tree의 경우는 정확도와 정밀도, 재현율, F1 Score의 차이가 발생하기에 런타임 모듈과 마찬가지로 SVM알고리즘과 Gradient Boositng알고

리즘이 본 연구에서 제안한 방법으로 악성코드 저자 그룹 식별에 좋은 성능을 보인다.



[그림 Ⅳ-15] Kernel32.dll API 데이터를 6명의 저자일 때 적용한 알고리즘에 따른 결과

4. 기존 연구와의 비교

위의 실험결과를 바탕으로 SVM알고리즘과 Gradient Boosting알고리즘의 정확도를 기존연구와 비교를 진행하였다. 비교 대상 연구로는 저자 식별 연구 분야의 대표 연구인 Saed Alrabaee et al(2014)의 OBA2방법론과 최근 연구인 신건윤 외(2020)의 유전 알고리즘을 기반으로 한 악성코드 공격자 그룹의 특징을 추출하는 프레임워크이다. 먼저 OBA2 방법론은 저자가 2명일 때 95%의 정확도, 4명일 때 90%의 정확도, 6명일 때 84%의 정확도를 보인다. 특징 선택 및 추출 단계에서 소스코드 기반 분석 방법과 바이너리 기반 분석 방법을 사용하였다. 본 연구에서는 런타임 모듈 데이터를 통한 결과에서는 SVM알고리즘은 저자가 2그룹일 때 100%의 정확도, 4그룹일 때 94.6%의 정확도, 6그룹일 때 87%의 정확도를 보이고, Gradient Boosting알고리즘은 저자가 2그룹일 때 100%의

정확도, 4그룹일 때 97.3%의 정확도, 6그룹일 때 87.1%의 정확도를 보이며 런타임 모듈 데이터 기준 OBA2방법론에서 제시한 방법보다 최대 7.3%p높은 정확도를 보인다. 시간적인 측면에서도 자동화 분석을 사용하여 특징을 추출하기 때문에 OBA2방법론에서 제시한 방법보다 시간소요가 적다. <표 Ⅳ-15>는 비교결과를 표로 나타낸 것이다.

<표 Ⅳ-17> OBA2 방법론과의 정확도 비교

저자 수	OBA2	Proposed Model (런타임 모듈)		Proposed Model (Kernel32.dll API)	
		SVM	Gradient Boosting	SVM	Gradient Boosting
2	95%	100%	100%	95.2%	100%
4	90%	94.6%	97.3%	85.3%	87.8%
6	84%	87%	87.1%	79.6%	84.7%

유전 알고리즘을 기반으로 한 악성코드 공격자 그룹의 특징을 추출하는 프레임워크를 통해 추출한 특징을 바탕으로 악성코드의 저자를 식별하는 연구를 진행하였을 때 5명의 저자일 때 86%의 정확도를 보였고, 저자 식별을 위한 특징 선택 및 추출 단계에서 소스코드 기반 분석 방법과 바이너리 기반 분석 방법을 사용하였다. 본 연구에서 제안한 방법을 통해 악성코드의 저자를 식별하는 연구를 진행했을 때, 5명의 저자 기준으로 런타임 모듈 데이터를 SVM알고리즘을 적용하였을 때 89.3%의 정확도를 보였고, Gradient Boosting알고리즘을 적용하였을 때는 91.5%의 정확도를 보였다. 또한 Kernel32.dll API 데이터를 SVM알고리즘을 적용하였을 때 84%의 정확도를 보였고, Gradient Boosting알고리즘을 적용하였을 때는 86%의 정확도를 보였다. Kernel32.dll API데이터를 통한 연구결과는 기존 연구보다 정확도가 낮거나 같은 결과를 보였지만, 런타임 모듈 데이터를 통한 연구 결과는 기존 연구보다 3.3%, 5.5%의 정확도가 높은 것을 볼 수 있다. <표 Ⅳ-16>은 비교결과를 표로 나타낸 것이다. 또한 자동화 분석을 사용

하여 특징을 선택하고 추출하였기 때문에 유전 알고리즘을 기반으로 한 악성코드 공격자 그룹의 특징을 추출하는 프레임워크보다 시간 소요가 적다.

<표 Ⅳ-18> 유전 알고리즘 기반 프레임워크와의 정확도 비교

저자 수	유전 알고리즘기반 공격자 그룹 특징 추출 프레임워크	Proposed Model (런타임 모듈)		Proposed Model (Kernel32.dll API)	
		SVM	Gradient Boosting	SVM	Gradient Boosting
5	86%	89.3%	91.5%	84%	86%

V. 결론 및 향후 연구

IT기술의 발달로 인해 나타나는 부정적인 변화 중에서 모듈화로 인한 악성코드의 대량생산과 APT공격의 발달이 있다. 대량 생산되는 악성코드의 경우 기존의 악성코드에서 일부만 수정한 변종이 많으며, 이는 원래의 악성코드의 특징을 파악하여 저자를 식별하게 된다면 새롭게 생성되는 악성코드를 식별하기 수월할 것이고, 장기간 지속 공격을 하는 APT공격 같은 경우는 저자 식별을 통해 악성코드들의 패턴을 파악하여 피해를 낮출 수 있다. 그러므로 악성코드 저자 식별 연구가 활성화 될 필요가 있다. 하지만 악성코드 저자 식별 연구는 모듈화로 인해 대량생산되는 악성코드에 대해 데이터를 획득하고 소스코드 및 바이너리로 변환할 때와, 저자 식별을 위한 특징을 선정하는 과정에서 시간적, 전문 인력의 필요성 등의 한계점이 존재한다.

이러한 기존의 악성코드 저자 식별 연구의 한계점을 해결하기 위해 본 연구에서는 자동화 분석 기반 악성코드 저자 식별 모델을 제안하였다. 자동화 분석은 악성코드의 파일이 없어도 해시값 만으로 분석이 가능하며 파일을 바이너리나 소스코드로 변환하지 않아도 분석결과를 추출할 수 있다. 또한 전문 인력을 통해 특징을 선정하지 않아도 필요한 정보를 추출할 수 있으므로 빠르게 분석하고 특징을 추출할 수 있는 장점이 있다. 자동화 분석을 통해 나오는 결과 값 중에서 본 연구에서는 런타임 모듈과 Kernel32.dll API를 식별특징으로 선정하였다. 또한 기존의 악성코드 저자 식별 연구보다 정확도를 높이기 위해 다양한 머신러닝 알고리즘을 사용하였다. 실험 결과 SVM 알고리즘과 Gradient Boosting 알고리즘이 기존 연구보다 최대 7.3%p 정확도가 높게 나오는 것을 볼 수 있었다. 그러므로 본 연구는 대량 생산되는 악성코드에 대해 비교적 높은 정확도로 빠르게 저자를 식별할 수 있다는 의미에서 기존연구와 차별화 되어 있으며, 새롭고 빠르게 생성되는 악성코드에 대해 저자를 식별하여 피해를 줄일 수 있을 것이라 기대한다.

하지만 악성코드 분석 방법 중 자동화 분석의 특성상 모든 분석이 정확한 것이 아니기 때문에 정확도가 낮게 나오거나, 자동화 분석에서의 결과 값을 도출하지 못할 경우에는 소스코드 기반 분석 방법과 바이너리 기반 분석 방법으로 상세 분석을 진행해야 하는 한계점이 있고, 데이터의 한계로 인해 다양한 그룹의 연구를 진행하지 못하였다.

향후 연구로는 난독화나 패킹이 되어있어 분석 결과가 도출되지 않는 자동화 분석의 한계점을 해결하기 위한 연구를 진행하여, 본 연구에서 제안하는 방법을 통해 모든 악성코드에 대해 저자 식별이 가능한 모델을 개발할 것이다. 또한 윈도우즈 기반 실행파일에만 국한되지 않고, 전체 악성코드 파일에 적용할 수 있도록 추가 식별 특징을 선정하고, 시각화 기준 마련 후 딥러닝 알고리즘을 활용하여 연구를 진행할 것이다.

참 고 문 헌

[1] Verizon, Gabriel Bassett, C. David Hylender, Philippe Langlois, Alexandre Pinto, Suzanne Widup(2020). Data Breach Investigations Report[Research Report]. Retrieved From https://www.verizon.com/business/resources/reports/dbir/2020

[2] CISCO, Kerry Singleton, Michiko Kamata, Bidhan Roy(2021). Cybersecurity for SMBs:Asia Pacific Businesses Prepare for Digital Defense[Research Report]. Retrieved From https://www.cisco.com/c/en_sg/products/security/cybersecurity-for-smbs-in-asia-pacific/index.html

[3] Saed Alrabaee, Paria Shirani, Mourad Debbabi, Lingyu Wang(2016). On the Feasibility of Malware Authorship Attribution. *International Symposium on Foundations and Practice of Security*, 256-272.

[4] Efstathios Stamatatos(2009). A survey of modern authorship attribution methods. *Journal of the American Society for information Science and Technology, 60*(3), 538-556.

[5] Saed Alrabaee, Noman Saleem, Stere Preda, Lingyu Wang, Mourad Debbabi(2014). OBA2:An Onion approach to Binary code Authorship Attribution. *Digital Investigation, Vol.11,* 94-103.

[6] Simon Kramer, Julian C. Bradfield(2010). A general definition of malware. *Journal in Computer Virology, Vol. 6,* 105-114.

[7] TitanFil(2021.05.14.). 7 Types of Computer Malware and How to Prevent Them[Research Report]. Retrieved From https://www.titanfile.com/blog/types-of-computer-malware/

[8] Gerard Biau, Luc Devroye(2015). *Lectures on the Nearest Neighbor Method.* German : Springer Science+Business Media

[9] Sundar Vishwanathan, M. Narasimha Murty(2002). SSVM: a simple SVM algorithm. *Proceedings of the 2002 International Joint Conference on Neural Networks*. IJCNN'02 (Cat. No.02CH37290)

[10] Anthony J. Myles, Rovert N. Feudale, Yand Liu, Nathaniel A. Woody, Steven D. Brown(2004). An introduction to decision tree modeling. *Jouranal of Chemometrics, 18*(6), 275-285

[11] ZH Zhou(2019). Ensemble methods: foundations and algorithms. United Kingdom: A CHAPMAN & HALL BOOK

[12] Nathan Rosenblum, Xiaojin Zhu, Barton P. Miller(2011). Who Wrote This Code? Identifying the Authors of Program Binaries. *Computer Security-ESORICS 2011,* 172-189.

[13] 홍석진, 홍지원, 김상욱, 김동필, 김원호(2018). 딥 러닝 기반 분류 모델을 이용한 악성코드 제작자 그룹 분류. *한국정보과학회, 데이터베이스연구 34*(2), 34-45.

[14] 신건윤, 김동욱, 한명묵(2020). 공격자 그룹 특징 추출 프레임워크: 악성코드저자 그룹 식별을 위한 유전 알고리즘 기반 저자 클러스터링. *인터넷정보학회논문지, 21*(2), 1-8.

[15] 황철훈, 신건윤, 김동욱, 한명묵(2020). 서바이벌 네트워크 개념을 이용한 저자 식별 프레임워크: 의미론적 특징과 특징 허용 범위. *정보보호학회논문지, 30*(6), 1013-1021.

[16] Muhammad Ijaz, Muhammad Hanif Durad, Maliha ismail(2019). Static and Dynamic Malware Analysis Using Machine Learning. *Proceedings of 2019 16th International Bhurban Conference on Applied Sciences & Technology (IBCAST)*. Islamabad, Pakistan, 8th - 12th January, 2019

[17] Daniele Ucci, Leonardo Aniello, Roberto Baldoni(2019). Survey of machine learning techniques for malware analysis. *ELSEVIER COMPUTERS & SECURITY, Vol. 81,* 123-147.

[18] Vaibhavi Kalgutkar, Ratinder Kaur, Hugo Gonzalez, Natalia

Stakhanova(2019). Code Authorship Attribution: Methods and Challenges. *ACM Computing Surveys, 52*(1), 1-36.

[19] 홍지원, 박상현, 김상욱(2018). 악성코드 작성자 그룹 분류를 위한 특징 선택. *한국정보과학회, 데이터베이스연구 34*(1), 14-24.

[20] 김태근, 임을규(2014). 악성코드 변종 탐지를 위한 코드 재사용 분석 기법. *정보보호학회지, 24*(1), 32-38.

[21] 허민석(2020). 나의 첫 머신러닝/딥러닝. 경기도 파주시: 위키북스

[22] Infineon(2019.10). Cybersecurity basics: Everything you should know[Research Report]. Retrieved From https://www.infineon.com/cms/en/discoveries/cybersecurity-basics/

[23] accenture(2019.03.06.). Ninth Annual Cost of Cybercrime Study [Research Report]. Retrieved From https://www.accenture.com/us-en/insights/security/cost-cybercrime-study

[24] Julia Sowells(2019.04.25.). Static Malware Analysis Vs Dynamic Malware Analysis. *HACKER COMBAT COMMUNITY News*. Retrieved From https://hackercombat.com/static-malware-analysis-vs-dynamic -malware-analysis/

[25] CYBERSECURITY&INFRASTRUCTURE SECURITY AGENCY(2021.04.15.).AppleJeus: Analysis of North Korea' s Cryptocurrency Malware[Research Report]. Retrieved Fromhttps://us-cert.cisa.gov/ncas/alerts/aa21-048a

[ABSTRACT]

A Study on Malware Authorship Attribution Model through Automated Analysis

Sang Woo Lee
(Supervised by Professor Jungwon Cho)

Department of Convergence Information Security
The Graduate School of Jeju National Unieversity

The advancement in IT technology can have both positive and negative effects. With the present security system, there is a limit to defend and respond to the mass production of malware due to modularization and standardization or APT attacks targeting vulnerabilities of the system. To solve these problems, recent researches have studied on malware authorship attribution using artificial intelligence technology.

Malware authorship attribution study is an expanded research field from the existing authorship attribution field. To infer the author of the malware code or determine whether it is malicious or not, the researcher identifies the characteristics of well-known malware and then assigns these to the unknown one. Currently, it is being used as one of the detection techniques based on malware forensics or identifying patterns of continuous attacks such as APT attacks. The analysis methods to identify the author are as follows. One is a source code-based analysis method that extracts features from the source code and the other is a binary-based analysis method that extracts features from the binary. However, to handle the modularization and the increasing amount of malicious code with these methods, both time and

manpower are insufficient to figure out the characteristics of the malware.

Therefore, in this study, I have designed a model to malware authorship attribution by rapidly extracting and analyzing features using automated analysis. Automated analysis is an analysis method using a tool, and can be analyzed through a file of malware and the specific hash values without experts. Furthermore, it is the fastest to figure out among other malware analysis methods. The experiment was conducted by applying various machine learning classification algorithms to six malware author groups, and Runtime Modules and Kernel32.dll API that can be extracted from automated analysis were selected as features for author identification. In addition, as a result of comparison with existing studies based on the experimental results, it showed generally higher accuracy than existing ones. By using automated analysis, it extracts features of malware faster than existing source code-based and binary-based analysis methods. If the malware authorship attribution model through the automated analysis proposed in this study is applied to mass-produced malicious code and APT attacks, it is expected to perform better than the existing malware authorship attribution method.