Pusan National University
Computer Science and Engineering
Technical Report 2022-09

NLP 기반 Powershell 악성 스크립트 탐지 툴

201845108 김서영
201924481 박지연
201714537 최연재

지도교수 : 최윤호 교수님

<제목 차례>

1. 서론

1.1. 연구 배경

PowerShell은 시스템 관리자를 위해 특별히 설계된 Windows 명령줄 셸이다. 이는 운영 체제와 프로세스를 관리하는 기능의 대부분을 제어할 수 있는 API와 단일 함수 명령줄 도구(cmdlet, Command-Let)를 가지고 있다. 다양한 장점에도 불구하고 비교적 짧은 소스코드로 효과적인 성능을 낼 수 있다는 점으로 인해 사이버 공격자의 공격 도구로 사용된다는 취약성이 있다.

2014년 이후 'poweliks'나 'kovter'와 같은 악성코드에서 PowerShell이 공격도구로서 활용되기 시작하였다. 2017년에는 PowerShell을 이용한 악성코드가 전년대비 432%가 증가하였고, 그 중 4분기에서만 267%가 증가하였다는 것을 알 수 있다.

PowerShell은 정상적인 프로그램이므로 완전한 실행을 차단할 수 없어 악성 코드를 방지하기 위해서는 전체 코드 안에서 실행되는 악성 명령을 차단해야 한다. 그러나 PowerShell의 유연한 기능들은 스크립트가 다양한 난독화를 할 수 있게 하고 이는 기존 안티바이러스의 시그니처 기반 탐지를 어렵게 한다. 또한, 사이버 공격자들은 지속적으로 탐지 기술을 우회할 수 있는 방안을 찾고 있기 때문에 기존의 방식으로는 사이버 범죄 행위를 발견하기 어렵다. 그리하여 해당 연구에서는 난독화된 파워셀 기반 악성스크립트를 비난독화하고, 머신러닝 모델을 활용하여 악성 스크립트를 탐지하고자 한다.

1.2. 연구 목표

난독화된 스크립트 분석을 위한 비난독화 모듈을 개발하고 머신 러닝 기법을 활용한 악성 스크립트 분류 모듈을 개발한다. 그리고 해당 프로그램의 사용을 위해 GUI 시각화 도구 개발을 진행한다.

비난독화는 Base64와 16진수를 이용한 방식을 이용하여 개발을 진행한다. 그리고 PoswerShell 스크립트의 명령어들을 추출하여 미리 학습시킨 앙상블 모델을 이용하여 악성 유무를 판별하는 프로그램을 개발한다. 앙상블 모델은 Boostring 기법을 이용한 AdaBoost, GBM(Gradient Boosting Method), XGBoost(Extreme Gradient boostring)를 Voting 방식으로 결합하여 사용한다. 전체적인 구성도는 아래와 같다.

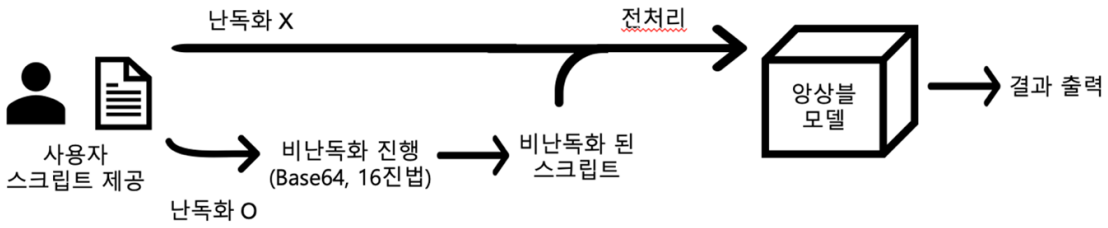

그림 1. 전체적인 구성도

2. 연구 내용

2.1. 비난독화 모듈 개발

사이버 공격자들은 난독화 기법을 사용하여 안티바이러스 프로그램의 탐지를 회피하거나 악성 코드 분석가들이 해당 코드를 분석하기 어렵게 만든다. 이를 방지하기 위하여 본 프로젝트에서는 'Base64'와 '16진수'을 이용하여 인코딩된 파일을 비난독화 해주는 모듈을 개발하였다.

Base64 인코딩이란 Binary Data를 Text로 바꾸어주는 인코딩 방법중에 하나이다. 이는 Binary Data를 문자 코드에 영향을 받지 않는 공통 ASCII 영역의 문자들로만 이루어진 일렬의 문자열로 바꾸어준다. 16진수 인코딩이란 각 바이트를 2개의 문자(0~9사이의 숫자 및 A~F 사이의 문자)로 변환하는 방식이다.

Base64 디코딩을 위해서 'base64'라는 모듈을 사용하였다. 난독화된 파일을 각 라인별로 읽은 후, base64모듈 안의 'b64decode'라는 함수를 이용하여 각각의 라인을 base64 방식으로 비난독화 해주었다.

16진수 디코딩을 위해서는 파이썬의 내장 함수인 'bytearray'를 사용하였다. 이것 역시 난독화된 파일을 각 라인별로 읽은 후, 내장 함수안의 'fromhex'라는 함수를 이용하여 16진수를 이용하여 비난독화 하였다.

2.2. 데이터 수집

모델 학습을 위한 데이터를 위해서 정상 데이터는 'Github', 'PowershellGallery', 'PoshCode'에서, 악성 데이터는 'Github', 'Malwarebazaar', 'Any.Run'에서 수집하였다. 수집된 정상 데이터들은 파워쉘 스크립트 파일과 모듈 파일로 이루어져 있으며 악성 데이터 중 비난독화할 수 없는 데이터들은 데이터 세트에서 제외하였다. 데이터 수집 관련해서 악성 데이터가 정상 데이터에 비해 상대적으로 많이 부족하여 악성 데이터는 개발을 진행하면서 지속적으로 수집을 진행하였다.

2.3. 데이터 전처리 및 분석

우선 Powershell 스크립트를 명령어만 뽑아 내기 위해 'textPsParser'의 Tokenize 메소드를 사용하여 토큰을 가져오는 함수'와 '파일을 읽고 txt 파일에 결과를 저장하는 함수'를 파워쉘 스크립트로 구현하였다. 논문에 따르면 'Command', 'CommandArgument', 'CommandParameter', 'Keyword', 'Variable' 5가지의 명령어 조합이 가장 높은 성능을 보여주는 것을 참고하여 5가지의 명령어 종류만 이용했다. [2]

변수의 목적을 나타내는데 이름이 사용되는 경우 공통 변수 이름이 여러 스크립트에 나

타날 수 있다. 조건문이나 분기문에서 특정 변수 상태를 확인하는 경우가 있기 때문에 Keyword와 Variable 유형을 함께 확인하며 분석하기 위해 사용했다. Command, CommandParameter, CommandArgument는 명령어가 실행될 때 같이 쓰이기 때문에 밀접한 관계이기에 사용했고, Number와 String 유형은 노이즈로 작용할 수 있기 때문에 이 두 가지는 이용하지 않았다.

Combination	Token types
1	Command + CommandParameter + CommandArgument
2	Command + CommandParameter + CommandArgument + Keyword
3	Command + CommandParameter + CommandArgument + Variable
4	Command + CommandParameter + CommandArgument + Keyword + Variable
5	Command + CommandParameter + CommandArgument + Number
6	Command + CommandParameter + CommandArgument + String
7	Command + CommandParameter + CommandArgument + Member

Results \ Combination	1	2	3	4	5	6	7
Recall	93.4	93.8	93.6	**94.5**	92.7	92.5	93.1
FPR	10	4.9	5	**5.5**	8.9	4.9	5.3

그림 2. 명령어 종류에 따른 성능 분석표

2.3.1. Run-PsParser 함수

Tokenize 메소드를 통해 코드를 토큰으로 분리하는 작업을 진행한다.

2.3.2. Test-Parser 함수

입력한 경로에서 반복적으로 돌면서 txt파일을 찾고 Run-PsParser로 받아온 토큰 중 'Command', 'CommandArgument', 'CommandParameter', 'Keyword', 'Variable' 이 5가지 타입의 토큰만 읽어서 다시 txt파일로 저장한다. TextPsParser 파일을 exe 파일로 만든 뒤 파이썬에서 exe 파일을 실행하여 이용할 수 있도록 구현하였다.

Powershell에서 단일문자 자체는 의미가 없기 때문에 길이가 2이상인 토큰만 사용할 것이다. 또한 Powershell은 대소문자 구별도 없으므로 모든 토큰을 소문자화 해준다. 이 과정은 textPsParser로 토큰화된 스크립트를 다시 읽어 lower_case함수를 통해 소문자화를 진행, 길이가 2 이상인 토큰만 남기는 형식으로 진행된다.

2.3.3. 데이터 분석

데이터 분석을 진행하기 전에 분석과 전처리 그리고 모델 훈련을 위해 사용한 라이브러리는 다음과 같다.

	version
Scikit-learn	1.1.2
Xgboost	1.6.2
Pandas	1.3.4
Numby	1.22.3
Joblib	1.1.0
Matplotlib	3.4.3

표 1. 필요 라이브러리

토큰화 과정으로 생성된 파일들을 읽고 라벨링 작업을 거친다.

	라벨
정상	0
악성	1

표 2. 라벨링

이때 정상은 0, 악성은 1로 라벨링 하였다.

결측값과 중복 값의 개수를 파악하여 제거하였다. 이후 히스토그램으로 토큰 개수 분포를 살펴본 결과는 다음과 같다.

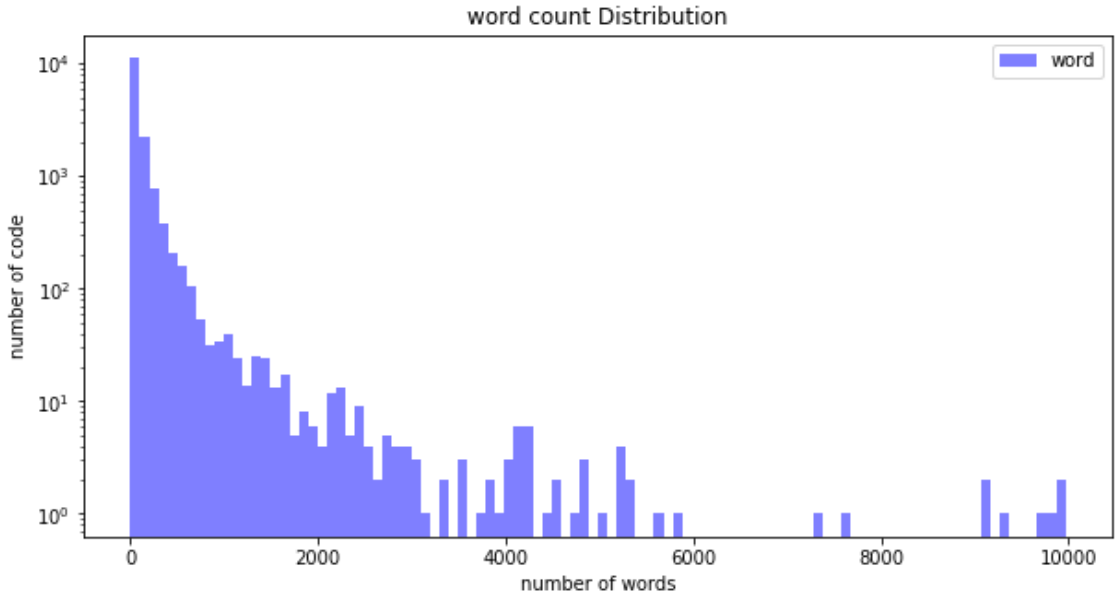

그림 3. 토큰 개수 분포 (히스토그램)

그림 3을 보면 1000개 이하의 토큰을 가진 스크립트들이 많다는 것을 확인할 수 있다. 최솟값이 1이고 최댓값이 9967인 것으로 나타났다. 이를 통해 개수 차가 크다는 것을 알 수 있다.

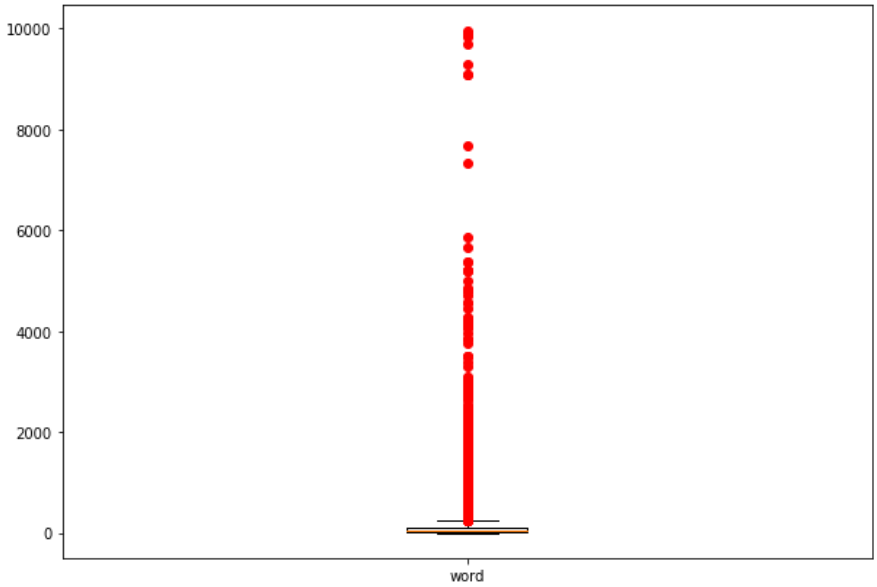

그림 4. 토큰 개수 분포 (박스 플롯)

박스 플롯으로 토큰 개수 분포를 나타내었다. 박스 플롯을 통해 이상치 데이터가 많이 보이는 것을 확인할 수 있다.
이상치 개수는 1656개로 전체의 10.7%를 차지한다. 이 중 정상은 1555개, 악성은 101개를 차지한다.

데이터 분석이 끝난 후 전체 데이터 세트는 다음과 같다.

분류	개수
정상	14516개 (93.5%)
악성	1002개 (6.5%)

표 3. 전체 데이터 세트

2.3.4. 벡터화

데이터를 모델의 입력값으로 주기 위해 숫자로 변환하는 과정이 필요하다. 이를 카운트 기반의 단어 표현 방법인 BoW를 사용하여 벡터화를 진행하였다. BoW는 단어들의 순서는

고려하지 않고 출현 빈도만 고려하는 방법이다.

Scikit-learn의 CountVectorizer 클래스를 사용하여 BoW를 구현하였다. 정규 표현식을 사용하여 'a-z0-9-._:'을 제외한 나머지 문자들은 모두 공백으로 치환하였으며 n-그램의 범위는 (1,2)로 설정하였다. 또한 최대 feature 값은 4000으로 설정하였다.

2.4. 모델 구현

스크립트가 악성 스크립트인지 정상 스크립트인지 판별하는 이진 분류 모델을 구현하였다. 구현한 머신러닝 모델은 다음과 같다.

머신러닝	
RF (Random Forest)	scikit-learn
Ada Boost	scikit-learn
GBM (Gradient Boost)	scikit-learn
XGB (XGBoost)	xgboost

표 4. 머신러닝 모델

scikit-learn과 xgboost를 사용하여 모델을 구현하였으며 Jupyter Notebook에서 작성하고 실행하였다.

앙상블 학습이란 여러 개의 분류기를 생성하고 예측 결과를 결합하여 보다 정확하고 신뢰성이 높은 예측값을 도출하는 기법을 말한다. 단일 분류기에 비해 분류 성능이 우수하기 때문에 많은 곳에서 사용하고 있다. 앙상블 학습에는 일반적으로 보팅, 배깅, 부스팅으로 나눌 수 있다. 이 중 배깅 방식을 사용하는 Random Forest, 부스팅 방식을 사용하는 Ada Boost, GBM, XGBoost를 선정하여 구현하였다.

2.5. 모델 학습

전체 데이터 세트의 80%를 모델 학습에 사용하였다. 각 모델의 하이퍼 파라미터 튜닝을 진행하였다. 최적의 값을 찾아 모델의 성능을 높이고자 하였다.

2.5.1. 하이퍼 파라미터 튜닝

GridSearchCV를 통해 교차 검증과 하이퍼 파라미터 튜닝을 수행하였다. 고정된 학습 데이터와 테스트 데이터로 평가하면 편향된 모델이 생성될 수 있다. 이를 개선하고 일반화된 모델을 만들기 위해 3-fold 교차 검증을 통해 최적의 하이퍼 파라미터를 계산하고 이를 각 모델에 적용하여 학습하였다.

또한 정상에 비해 악성 데이터 개수가 매우 적기 때문에 Stratified K-fold 방식을 사용하여 악성, 정상 데이터 세트의 비율이 동일하게 분포하도록 하였다. 아래 표는 각 모델의 최적의 하이퍼 파라미터 값을 보여준다.

머신러닝	
RF	estimators = 100, max depth = 15, min samples split = 10, min samples leaf = 3
Ada Boost	estimators = 300
GBM	estimators = 150, learning rate = 0.04
XGB	estimators = 150, learning rate = 0.2, max depth = 8

표 5. 최적의 하이퍼 파라미터

각 모델에 최적의 하이퍼 파라미터를 적용하여 학습하였다. 모델은 크게 (1) 4가지 분류 모델, 그리고 이 중 일부 모델을 결합한 (2) 앙상블 모델 두 파트로 나누어 학습하고 평가하였다.

2.6. 모델 평가

테스트 데이터 세트에서 악성의 개수가 매우 적기 때문에 정확도로 평가하기에는 무리가 있다. 따라서 정확도보다는 정밀도, 재현율, F1 score를 중점으로 모델을 평가하였다.

정밀도는 모델이 악성이라고 예측한 데이터 중 실제로 악성인 데이터를 의미한다. 또한 재현율은 실제 악성 데이터 중 모델이 악성이라고 판단한 데이터를 의미한다. F1 score는 정밀도와 재현율의 조화 평균을 의미한다.

2.6.1. 머신러닝 분류 모델 비교

RF, Ada Boost, GBM, XGBoost 모델의 평가 결과는 다음과 같다.

머신러닝	정확도	정밀도	재현율	F1
RF	0.979	**1.00**	0.68	0.809
GBM	0.986	0.981	0.80	0.881
Ada	0.984	0.922	0.83	0.874
XGB	**0.988**	0.977	**0.84**	**0.903**

표 6. 머신러닝 모델 평가

위의 표를 보면 RF의 정밀도는 1이지만 재현율은 0.68로 낮은 값을 나타낸다. 이는 정상은 정확하게 탐지하지만 악성은 잘 탐지하지 못한다는 것을 알 수 있다. 또한 4가지 머신러닝 모델 중에서 XGBoost가 가장 좋은 성능을 보인다는 것을 알 수 있다.

각 모델의 ROC curve를 그려보고 AUC 값을 계산해보았다. ROC curve는 모델의 판단 기준에 따라 달라지는 FPR, TPR의 변화를 나타낸 선이다. ROC curve는 분류 모델의 성능을 판단하고자 할 때 사용할 수 있다. 이때 좌상단으로 가장 많이 치우쳐있는 ROC curve가 가장 높은 성능을 보인다고 할 수 있다. ROC curve 아래 영역의 크기를 표현한 것이 AUC이다. AUC 값이 1에 가까울수록 분류 성능이 좋다.

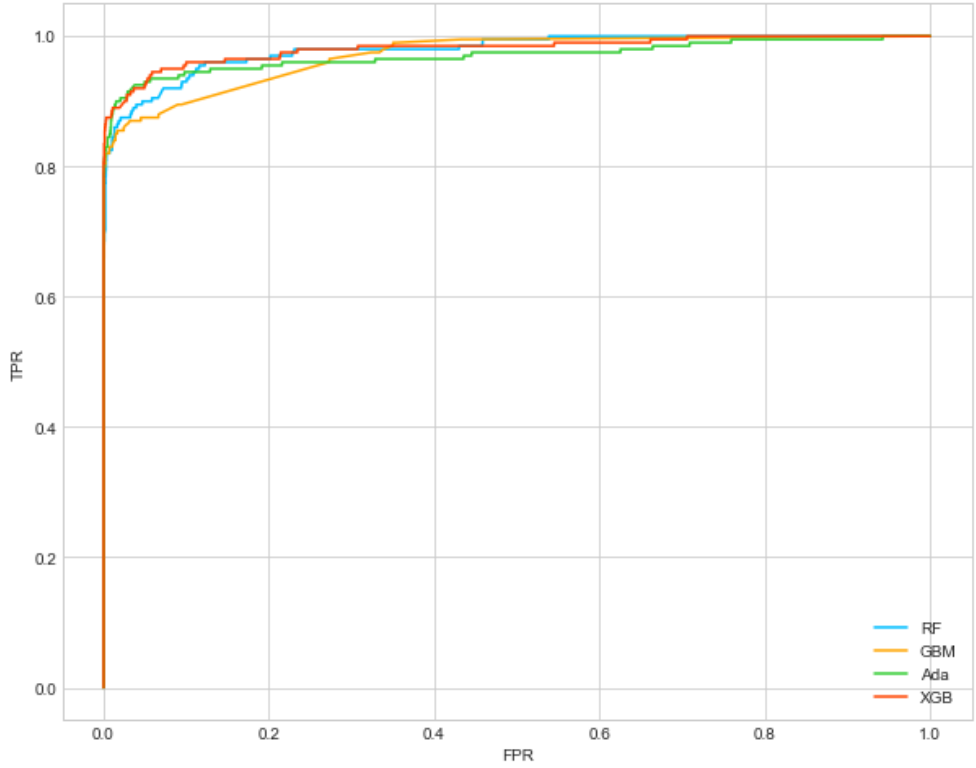

그림 5. 머신러닝 모델의 ROC curve

머신러닝	AUC
RF	0.976
GBM	0.97
Ada	0.97
XGB	0.981

표 7. 머신러닝 모델의 AUC

위의 그림 5와 표 7을 보면 XGBoost의 AUC가 가장 크다는 것을 알 수 있다. 결과적으로 4가지 모델 중 XGBoost의 성능이 가장 뛰어나다는 것을 확인할 수 있다.

2.6.2. 앙상블 모델 비교

재현율과 F1이 가장 낮은 RF를 제외하고 GBM, Ada boost, XGBoost를 앙상블 학습을 진행하였다. Ada Boost는 부스팅 기법 중 가장 기본적인 모델로 기존 부스팅 기법과 달리 학습에 이용된 개별 분류기에 서로 다른 가중치를 주어 최종 분류기를 만들어내는 방식이다. GBM은 잘못 분류된 점을 이용해 가중치를 결정하는 Ada Boost와 달리 Gradient Descent 기법을 이용하는 방식이고 XGBoost 또한 Gradient Boosting에 기반을 두지만 손실함수 뿐 아니라 모형 복잡도까지 고려를 하는 방식이다. 세 가지 모델 모두 부스팅 기법을 이용한다는 것은 동일하나 각각의 차이점이 있기에 선택하게 되었다.
소프트 보팅 방식을 사용하여 모든 분류기의 결정 확률 평균을 구한 뒤 가장 확률이 높은 레이블의 값을 최종 레이블로 선정하였다.

가장 좋은 성능을 보인 XGB에 더 큰 가중치를 주고 학습을 진행하였다. GBM+XGB, GBM+Ada, Ada+XGB, GBM+XGB+Ada 총 4개의 앙상블 모델을 평가한 결과는 다음과 같다.

앙상블	정확도	정밀도	재현율	F1
GBM + XGB	0.989	0.988	0.84	0.908
GBM + Ada	0.986	0.987	0.80	0.883
Ada + XGB	0.988	0.976	0.84	0.903
GBM + Ada + XGB	0.989	0.988	0.84	0.908

표 8. 앙상블 모델 평가

위 표 7을 보면 GBM+XGB, GBM+XGB+Ada 두 앙상블 모델이 가장 성능이 좋다는 것을 알 수 있다. 또한 이전의 XGB 모델과 비교했을 때 전반적으로 더 높은 성능을 보여준다. 4개의 앙상블 모델의 ROC 곡선과 AUC를 살펴본 결과는 다음과 같다.

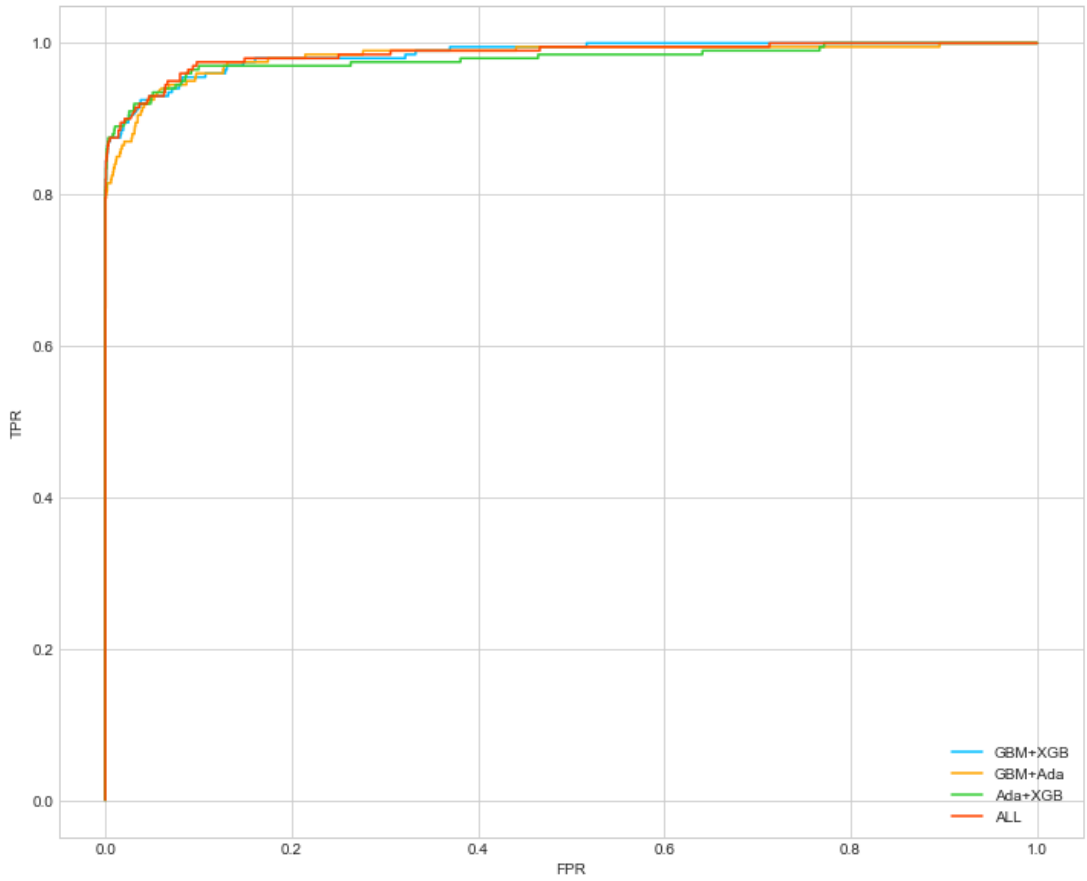

그림 6) 앙상블 모델의 ROC curve

앙상블	AUC
GBM + XGB	0.9853
GBM + Ada	0.9829
Ada + XGB	0.9789
GBM + Ada + XGB	**0.9867**

표 9. 앙상블 모델의 AUC

결과적으로 앙상블 모델의 성능이 각 머신러닝 모델의 성능보다 향상된 것을 확인할 수 있다. GBM, Ada boost, XGBoost 모델을 앙상블한 모델이 0.9867 AUC로 가장 높다. 따라서 이 모델을 최종적으로 악성 스크립트 분류 모델로 사용하였다.

GBM, Ada boost, XGBoost을 결합한 앙상블 모델의 혼동 행렬(confusion matrix)은 다음과 같다.

예측 \ 실제	악성 (1)	정상 (0)
악성 (1)	168 (TP)	2 (FP)
정상 (0)	32 (FN)	2902 (TN)

표 10. GBM, Ada boost, XGBoost 앙상블 모델의 혼동 행렬

표 10을 보면 정상에 비해 악성을 탐지하는 능력이 떨어지는 것을 알 수 있다. 이전에 표 8에서 재현율이 0.84라고 명시하였다. 이는 100개의 악성 데이터 중 84개는 정확하게 분류하지만 16개는 정상으로 분류함을 나타낸다. 본 과제에서 악성 스크립트를 정상 스크립트라고 판별하는 것은 위험하다고 생각한다. 따라서 이 부분을 개선할 필요가 있다.

2.7. GUI 시각화 도구 개발

GUI 시각화 도구 개발을 위해서는 tkinter라는 GUI에 대한 표준 Python 인터페이스를 사용하였다. 그리고 tkinter로 만들어진 시각화된 파이썬 프로그램을 실행 파일로 만들기 위해 파이썬 패키지 중에 하나인 pyinstaller를 사용하였다.

3. 연구 결과 분석 및 평가

그림 7. 파일 선택 화면

첫 번째 페이지에서는 분석을 위한 스크립트 파일을 선택하게 된다.

그림 8. 비난독화 방식 선택 화면

두 번째 페이지에서는 복호화 진행을 위해 비난독화 방식을 선택한다. 만약 선택된 스크립트가 난독화 되어 있지 않으면 '비난독화 진행 X'를 이용하여 비난독화를 진행하지 않을 수도 있다.

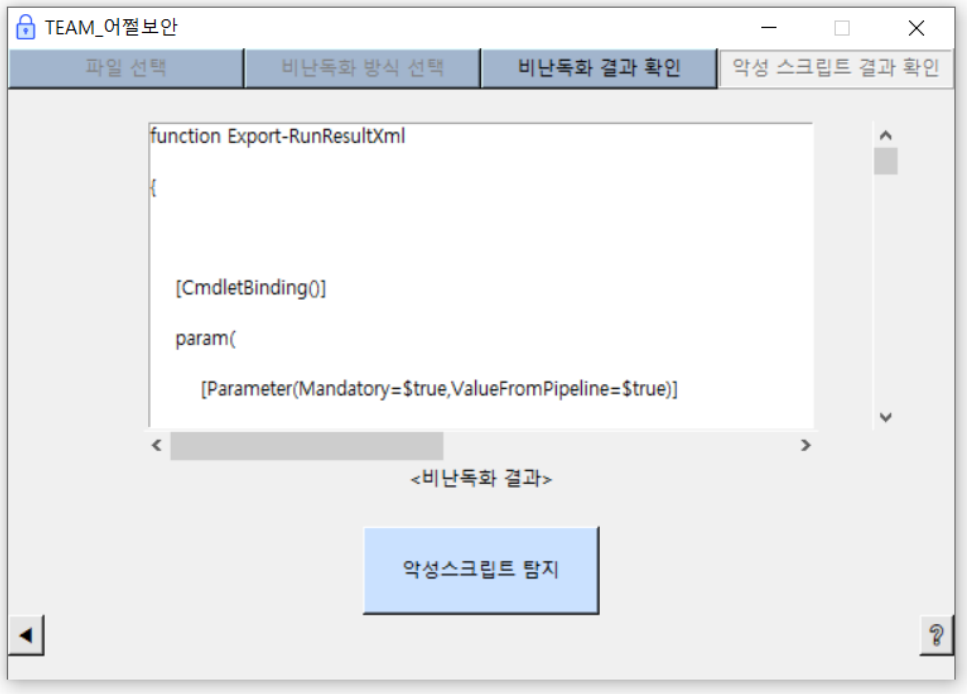

그림 9. 비난독화 결과 확인 화면

세 번째 페이지에서는 이전에 진행된 비난독화의 결과를 보여준다.

그림 10. 악성 스크립트 결과 확인 화면(악성)

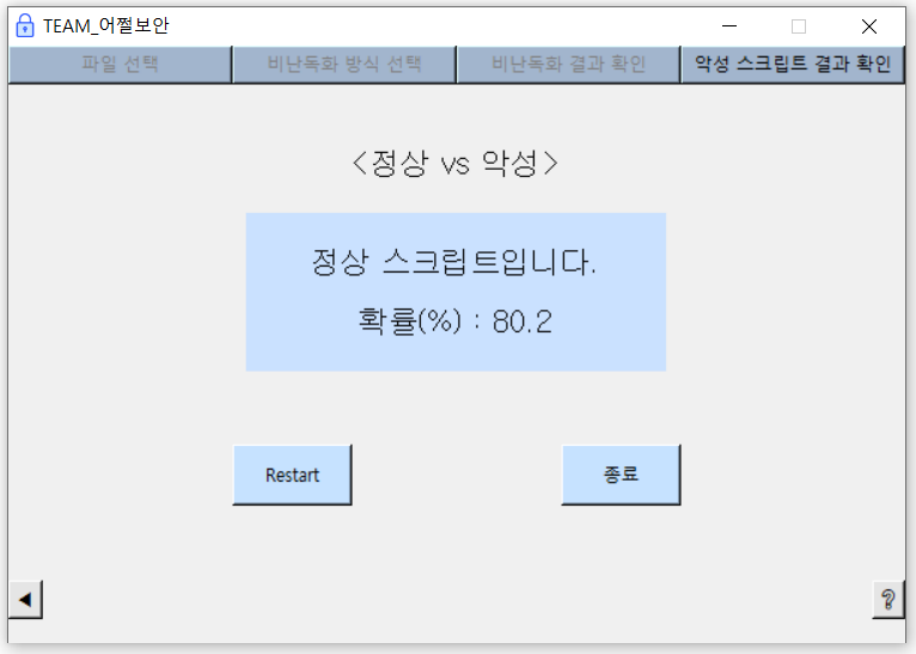

그림 11. 악성 스크립트 결과 확인 화면(정상)

마지막 페이지에서는 머신러닝을 이용한 악성 스크립트 분류 모델의 결과를 보여준다.

앞서 분류 모델 성능을 향상시키기 위해 하이퍼 파라미터 튜닝을 진행하였다. 튜닝은 하이퍼 파라미터 값 범위를 지정하고 3-fold 교차 검증을 통해 가능한 모든 경우의 수를 수행한다. 이는 하이퍼 파라미터 개수가 많아지고 값의 범위가 커질수록 시간이 매우 오래 걸린다는 단점이 있다. 본 과제에서 구현한 모델들은 하이퍼 파라미터 개수가 많아 다고려할 수 없다고 판단하였다. 따라서 일부 하이퍼 파라미터만 튜닝을 진행하게 되었다.

구현한 악성 스크립트 분류 모델에서 다른 요소에 비해 재현율이 낮은 값을 보였다. 위 표 2를 보면 전체 데이터 세트에서 악성은 6.5%를 차지한다. 정상에 비해 악성이 매우 적기 때문에 악성 스크립트를 잘 탐지하지 못하는 문제가 발생한 것으로 보인다.

본문에서는 명시하지 않았지만 이 문제를 해결하고자 오버 샘플링 방법 중에 하나인 SMOTE 방법을 시도하였다. 전체 데이터 세트에서 학습에 사용할 80%를 SMOTE로 오버 샘플링하여 학습하고 평가한 결과 정밀도는 낮아졌지만 재현율은 소폭 상승하였다. 하지만 더 낮은 AUC를 보였고 결과적으로 성능이 더 좋지 않다고 판단하였다. 노이즈에 영향을 받아 이러한 결과가 발생했다고 추측하였다. 이를 통해 실질적으로 악성 스크립트를 더 많이 수집하는 것이 성능을 개선할 수 있을 것으로 판단하였다.

4. 결론 및 향후 연구 방향

본 졸업 과제에서는 Powershell로 작성된 스크립트의 비난독화 툴 개발 및 악성 스크립트 분류 모델을 구현하여 스크립트를 판별하는 프로그램을 개발하였다. 전처리 과정에서

Powershell 스크립트로 작성한 ‘textPsParser.exe’를 통해 스크립트에서 필요한 토큰만 읽어올 수 있었다. 결론으로 Base64 encoding, 16진수로 encoding된 스크립트를 비난독화 할 수 있었으며, AdaBoost, GBM, XGBoost 세가지를 앙상블한 모델을 구현하여 결과를 얻을 수 있었다.

'PowerDecode, malzilla, Revoke-Obfuscation-master, psDecode' 등 다양한 툴을 이용하여 비난독화 방식의 다양성을 높이고자 하였다. 하지만 '프로그램 자체의 정확성의 문제, 문자열 제한의 문제, 호환성 문제, 난독화 방식 탐지의 문제' 등으로 인해 해당 프로젝트에서 사용하는것에 어려움이 있었다.

향후에는 위의 프로그램의 오픈소스를 이용하여 API 개발을 진행하고, API 호출과 응답을 통해 비난독화를 진행하고자 한다. 또한, 난독화 방식을 자동으로 탐지하는 기능을 추가하여 비난독화 방식을 선택하는 것이 아닌 자동으로 비난독화가 진행될 수 있도록 한다.

악성 스크립트 분류 모델은 하이퍼 파라미터 튜닝을 통해 최적의 하이퍼 파라미터 값을 계산하였다. AUC 0.9867, 재현율은 0.84로 비교적 높은 성능을 보였으나 모델에 학습시켰던 악성 스크립트의 개수가 지속적인 수집에도 불구하고 정상 스크립트에 비해 많이 부족했기 때문에 향후 더 많은 악성 스크립트를 수집하여 다시 모델학습을 시킨다면 더 좋은 성능이 나올 것으로 예상된다. 또한 다른 난독화 방식을 추가한다면 조금 더 다양한 스크립트에 대한 분석이 가능해 질 것으로 기대된다.

5. 구성원별 역할 및 개발 일정

이름	진척도
김서영	데이터 수집 및 정리, 전처리, 악성 스크립트 분류 모델 구현, 모델 학습 및 평가
박지연	데이터 수집 및 정리, 파워쉘 실행 파일 개발, 데이터 분석 진행, 악성 스크립트 분류 모델 구현, 모델 학습 및 평가
최연재	비난독화 툴 분석, GUI 생성, 라이브러리를 이용한 비난독화 진행, 악성코드 데이터 분류
공통	보고서 작성, 프로젝스 소개 동영상 및 시안 제작

표 11. 구성원별 역

<table>
<tr><td colspan="4">6월</td><td colspan="4">7월</td><td colspan="4">8월</td><td colspan="4">9월</td></tr>
<tr><td>1주</td><td>2주</td><td>3주</td><td>4주</td><td>1주</td><td>2주</td><td>3주</td><td>4주</td><td>1주</td><td>2주</td><td>3주</td><td>4주</td><td>1주</td><td>2주</td><td>3주</td><td>4주</td></tr>
<tr><td></td><td colspan="3">스터디(비난독화, RF, RNN, 자연어 처리</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td colspan="3">비난독화 도구 테스트</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td colspan="5">데이터 분석 및 전처리</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">중간 보고서 작성</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="3">비난독화 모듈 개발</td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="4">분류 모델 스터디 및 개발</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="3">모델 평가</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="2">GUI 만들기</td><td></td><td></td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="3">테스트 및 디버깅</td><td></td><td></td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>최종발표/최종 보고서 준비</td><td></td></tr>
</table>

표 12 개발 일정

6. 참고 문헌

[1] S.-H. Lee and J.-S. Moon, "PowerShell-based Malware Detection Method Using Command Execution Monitoring and Deep Learning," Journal of the Korea Institute of Information Security & Cryptology, vol. 28, no. 5, pp. 1197-1207, Oct. 2018.

[2] 송지현. "Study on the Optimal Feature Selection for Malicious PowerShell Script Detection." 국내석사학위논문 과학기술연합대학원대학교, 2020. 대전

[3] 파워셸 설명서: https://learn.microsoft.com/ko-kr/powershell/scripting/overview?view=powershell-7.2&viewFallbackFrom=powershell-7.1

[4] Mamoru Mimura, Yui Tajiri, "Static detection of malicious PowerShell based on word embeddings", Internet of Things, Volume 15, 2021, 100404, ISSN 2542-6605