2022년 대학 정보보호동아리(KUCIS) 프로젝트 결과보고서

악성코드 탐지 모델 연구

제출일	2022-11-26

대학/동아리	우석대학교 APS			
참여자 현황	**팀장**	이동근	**부팀장**	조용민
	팀원1	정선우	**팀원8**	양세희
	팀원2	고희상	**팀원8**	김건우
	팀원3	이승석	**팀원9**	김성두
	팀원4	최미린	**팀원10**	노재을
	팀원5	성서영	**팀원11**	김준희
	팀원6	이은주	**팀원12**	한지섭
	팀원7	최건우		
지도교수	**이진선**			

2022년 대학정보보호동아리(KUCIS) 프로젝트 결과보고서

연구 윤리 확보에 대한 자체 검증 확인서

우리 동아리는 대학정보보호동아리(KUCIS) 지원사업을 수행함에 있어 제출한 프로젝트 보고서등 전체 자료에 대해 연구윤리를 확보하였습니다.

아울러 해당사항에 대해 자체 검증을 시행하였음을 확인하며, 향후 연구윤리 미 확보(표절 등)로 인한 문제 발생시 해당 지원금 반납 등 어떠한 제제조치도 감수하겠습니다.

지도교수: 이진선

동아리 회장: 이동근 이동근

요 약 문

최근 4차 산업혁명이 시작되며 많은 변화가 불고 있는데 보안업계도 그중 하나에 속한다. 공격자는 해킹 공격에 인공지능, IoT, 빅데이터를 활용하고 있고 방어자 또한 인공지능을 활용하고 있다.

우리 APS 동아리는 이러한 시대에 맞추어 신기술을 활용한 연구를 진행하기로 정하였고 그 중 인공지능을 활용한 네트워크 탐지 시스템을 구축하여 작동원리를 파악하고 성능 분석 및 장단점을 알아보고자 했다.

그러나 시스템 구축을 하는데 너무 많은 시간이 소요됐고, 구축되고 나서도 데이터 가공 처리, 데이터 수집, 시각화 등에 실패하여 주제를 변경하게 되었다.

본래 목표였던 인공지능에 대한 이해도를 높이기 위해 다른 방안을 모색하던 중 인공지능을 접목하여 악성 파일 탐지 시스템을 구축하기로 하였다.

악성코드 탐지 모델을 구축하면서 악성 파일 분석, 특징 공학, 분류 모델 등을 공부하고 연구할 수 있었고, 우리가 원하던 대로 인공지능 모델을 만드는 것에 대한 이해도를 높일 수 있었다.

이번 연구 최종 결과에서 오픈 소스인 키콤 백신에 우리가 연구한 모델을 적용하고 테스트해 보았다. 데이터의 양도 부족하고, 데이터를 학습시키기에 시스템 성능 또한 문제였다. 또한, 주제 변경으로 인해 시간도 부족하여 좋은 모델을 구축하진 못했다.

다만 이번 연구를 바탕으로 더 좋은 모델을 구축하고 발전시킬 수 있는 발판이 마련됐다고 생각한다. 자세한 내용은 최종 결과 부분에 남겨놓았다.

목 차

1. 프로젝트 목표

최근 4차 산업혁명이 시작되며 많은 변화가 불고 있는데 보안업계도 그중 하나에 속한다. 공격자는 해킹 공격에 인공지능, IoT, 빅데이터를 활용하고 있고 방어자 또한 인공지능을 활용하고 있다.

우리 APS 동아리는 이러한 시대에 맞추어 신기술을 활용한 연구를 진행하기로 정하였다. 본래 목표는 네트워크 침입 탐지 인공지능 모델에 관한 연구였지만, 시스템을 구축하는 과정에 있어 많은 오류가 있어 연구를 포기하게 되었다.

다른 방안을 모색하던 중 인공지능을 활용한 악성코드 탐지 시스템을 구축하기로 하였고, 작동원리를 파악하고 성능 분석 및 장단점을 알아보고자 한다.

또한, 탐지 모델 구축에 있어 필요한 좋은 특징의 데이터를 위해 악성코드의 다양한 특징을 공부하여 인공지능뿐 아니라 악성코드에 대한 이해도도 높일 것이다.

2. 프로젝트 수행

2.1 연구개요

가. 인공지능 백신 구축 프로젝트

1) 악성코드에 관한 사전 지식 습득

- 악성코드 분석에 필요한 전반적인 지식을 습득(PE 구조, Assembly Language)
- 악성코드의 구조, 유형, 유포 방법, 동작 원리 등을 공부하여 데이터 셋을 만들고 이해하는 데에 큰 어려움이 없도록 한다.

2) 머신러닝 모델에 관한 사전 지식 습득

- 참고서에 나오는 머신러닝 모델에 관해 공부하여 머신러닝 이해도를 높인다. (SVM, randomforest, naivebayes, CNN, DNN)
- 머신러닝 모델에 대하여 장단점을 작성한다.

3) 실습에 필요한 다양한 핵심 패키지 이해

- 데이터 분석과 모델링 관련 기능을 제공하는 패키지들을 공부하고 실습한다.

4) 인공지능 백신 구축에 대한 보고서 작성

- 인공지능이 접목된 백신을 구축한 후 구축한 내용에 대한 구축 결과보고서를 작성한다.
- 우리가 만든 모델에 대한 평가를 작성한다.

2.2 프로젝트 주요 추진내용

가. 특징 추출에 필요한 악성코드 지식 습득

- 악성코드 데이터 셋을 만들고 이해하는 데 어려움이 없도록 하기 위해 동아리원들과 악성코드를 정적으로 분석하는 프로젝트를 진행했다.

1) 악성코드 유형

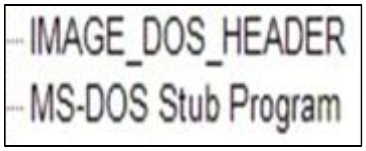
유형	설명
랜섬웨어	파일을 암호화해 접근을 차단하고 잠금 해제를 위한 금전 요구
트로이목마	신뢰할 수 있는 소프트웨어나 응용 프로그램으로 위장
웜	네트워크 상의 취약점을 찾아 스스로 복제
키로거	키 입력을 추적해 숨김 파일로 저장 후 자동으로 파일 전송, 정보 탈취
봇	자동화된 컴퓨터 프로그램으로 작동하는 악성 소프트웨어의 일종
루트킷	컴퓨터에 액세스하거나 제어하도록 설계된 정보를 훔치는 악성코드
스파이웨어	사용자 활동을 감시하는 악성 소프트웨어
바이러스	프로그램이나 응용 프로그램에 붙어 실행되는 악의적인 프로그램

2) PE 구조 학습

- 윈도우 운영체제에서 사용되는 실행 파일, DLL, object 코드 등을 위한 파일 형식이다. 이는 윈도우 로더가 실행 가능한 코드를 관리하는데 필요한 정보를 모아 놓은 데이터 구조체이다.

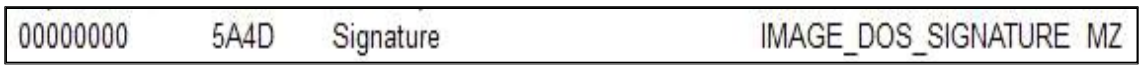
3) PE view를 이용한 실습

가) MS-DOS



나) MS-DOS 정보는 IMAGE_DOS_HEADER와 DOS stub으로 구성된다. 이 정보는 MS-DOS 버전의 파일에 대한 호환성을 위해 존재한다.

다) IMAGE_DOS_HEADER



- MZ" 는 매직 문자로 Mark Zbikowski의 이름에서 가져왔다.

- 구조체의 마지막 값인 e_lfanew는 NT 헤더의 위치를 나타낸다.

pFile	Data	Description	Value
00000084	014C	Machine	IMAGE_FILE_MACHINE_I386
00000086	000F	Number of Sections	
00000088	62A6F6BC	Time Date Stamp	2022/06/13 08:35:08 UTC
0000008C	00012C00	Pointer to Symbol Table	
00000090	000004E7	Number of Symbols	
00000094	00E0	Size of Optional Header	
00000096	0107	Characteristics	
		0001	IMAGE_FILE_RELOCS_STRIPPED
		0002	IMAGE_FILE_EXECUTABLE_IMAGE
		0004	IMAGE_FILE_LINE_NUMS_STRIPPED
		0100	IMAGE_FILE_32BIT_MACHINE

라) IMAGE_FILE_HEADER

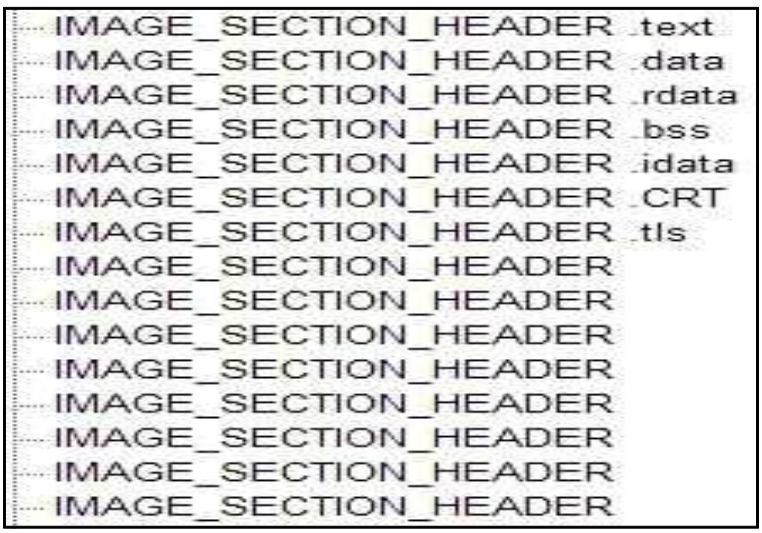
- Machine: CPU의 종류를 나타낸다.
- Number of Sections: 코드, 데이터, 리소스 등과 같은 섹션의 수를 나타낸다.
- Time Date Stamp: 빌드 된 시간(UTC(협정 세계시) 기반 시간 표현).
- Size of Optional Header: Optional_Header의 크기이다.
- Characteristics: 파일 종류와 속성을 나타낸다.

마) IMAGE_OPTIONAL_HEADER

pFile	Data	Description	Value
00000098	010B	Magic	IMAGE_NT_OPTIONAL_HDR32_MAGIC
000000A8	000014E0	Address of Entry Point	
000000B4	00400000	Image Base	
000000D0	0001D000	Size of Image	
000000D4	00000400	Size of Headers	

- Magic: 32(010B) & 64(020B) bit를 표현한다.
- Address of Entry Point: 프로그램의 시작 주소이다.
- Image Base: 메모리 로딩 주소이다. (EP(프로세스의 코드 실행 진입점) = ImageBase + AddressOfEntryPoint)
- Size of Image: 메모리에 로딩되는 크기이다.
- Size of Header: PE 헤더의 크기이다.

바) Section Header



- .text: 실행되는 코드를 담고 있는 섹션이다.
- .data: 초기화된 전역 변수를 담고 있으며, 읽고 쓰기가 가능한 섹션이다.
- .rdata: 읽기 전용의 데이터 섹션이다.
- .bss: 초기화되지 않은 전역 변수를 위한 섹션이다.
- .idata: 다른 dll에서 가져다 쓰는 함수들의 정보를 담고 있는 섹션이다(import table).
- .edata: 현재 PE 파일에 정의되어있는 함수를 다른 모듈이 사용할 수 있도록 함 수 목록을 담고 있는 섹션이다.

사) IMPORT_DIRECTORY_TABLE

pFile	Data	Description	Value
00002400	0000603C	Import Name Table RVA	
00002404	00000000	Time Date Stamp	
00002408	00000000	Forwarder Chain	
0000240C	000064DC	Name RVA	KERNEL32.dll
00002410	00006100	Import Address Table RVA	
00002414	0000608C	Import Name Table RVA	
00002418	00000000	Time Date Stamp	
0000241C	00000000	Forwarder Chain	
00002420	0000655C	Name RVA	msvcrt.dll
00002424	00006150	Import Address Table RVA	

- Import_Directory_Table은 import 하는 DLL의 정보를 가지고 있다.
- Import Name Table RVA: DLL 이름이 로딩되어있는
- ImageBase로부터의 상대주소이다.
- Import Address Table RVA: DLL 주소가 로딩되어있는 ImageBase로부터의 상대주소이다.

아) IMPORT_ADDRESS_TABLE

- Import_Address_Table은 프로그램에서 사용하는 라이브러리 테이블이다.

pFile	Data	Description	Value
00002500	000061C4	Hint/Name RVA	00D4 DeleteCriticalSection
00002504	000061DC	Hint/Name RVA	00EF EnterCriticalSection
00002508	000061F4	Hint/Name RVA	01C4 GetCurrentProcess
0000250C	00006208	Hint/Name RVA	01C5 GetCurrentProcessId
00002510	0000621E	Hint/Name RVA	01C9 GetCurrentThreadId
00002514	00006234	Hint/Name RVA	0203 GetLastError
00002518	00006244	Hint/Name RVA	0264 GetStartupInfoA
0000251C	00006256	Hint/Name RVA	027B GetSystemTimeAsFileTime
00002520	00006270	Hint/Name RVA	0297 GetTickCount
00002524	00006280	Hint/Name RVA	02EB InitializeCriticalSection
00002528	0000629C	Hint/Name RVA	0326 LeaveCriticalSection
0000252C	000062B4	Hint/Name RVA	0393 QueryPerformanceCounter
00002530	000062CE	Hint/Name RVA	0467 SetUnhandledExceptionFilter
00002534	000062EC	Hint/Name RVA	0474 Sleep
00002538	000062F4	Hint/Name RVA	0482 TerminateProcess
0000253C	00006308	Hint/Name RVA	0489 TlsGetValue
00002540	00006316	Hint/Name RVA	0496 UnhandledExceptionFilter
00002544	00006332	Hint/Name RVA	04B6 VirtualProtect
00002548	00006344	Hint/Name RVA	04B9 VirtualQuery
0000254C	00000000	End of Imports	KERNEL32.dll

3) 레지스터

- 레지스터는 프로세스에서 사용하는 소프트웨어 레지스터와 하드웨어 레지스터로 구분할 수 있다.

가) 범용 레지스터

- 범용 레지스터는 논리, 산술 연산에 사용되는 오퍼랜드나 주소 계산을 위한 오퍼랜드, 메모리 포인터에 사용된다.

레지스터	설명
EAX	대부분의 입출력과 산술, 논리연산을 수행. 함수의 리턴 값을 저장
EBX	주소 지정을 확장하기 위해 인덱스로서 사용
ECX	반복문을 수행할 때 반복 횟수를 지정하는데 사용
ESI	복사/비교의 대상의 주소를 가리킴 (Source)
EDI	복사/비교의 할 곳의 주소를 가리킴 (Destination)
ESP	스택 영역의 최상단을 가리킴
EBP	스택 영역의 기준이 되는 주소를 가리킴(현재 스택의 가장 밑바닥을 가리킨다고 생각하면 됨)
EIP	다음 실행할 명령이 들어 있는 메모리의 주소를 가리킴

- 바이트(8bit): AL, AH, BL, BH, CL, CH, DL, DH
- 워드(16bit) : AX, BX, CX, DX, SI, DI, BP. SP, IP
- 더블워드(32bit) : EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, EIP (32bit부터는 포인터, 기본 정수 단위로 쓰임)

나) 세그먼트 레지스터

- 각 세그먼트의 위치를 나타내며, 물리 주소로 변환할 때 사용한다.

레지스터	설명
CS	코드 세그먼트의 시작 주소
DS	데이터 세그먼트의 시작 주소
SS	스택 세그먼트의 시작 주소
ES FS GS	추가 세그먼트를 가리키는 시작 주소

다) 플래그 레지스터

- 플래그 레지스터는 프로그램이 수행되는 순간마다 수행 상태를 비트 단위로 저장한다.

레지스터	이름	설명
Z	제로	연산 결과가 0일 경우에 1로 설정
C	캐리	마지막 산술 연산이 레지스터 크기를 초과하여 비트를 빌리거나(뺄셈) 올림(덧셈)이 발생 한 경우 1로 설정
A	보조 캐리	스택 세그먼트의 시작 주소
O	오버 플로	산술 연산 후 상위비트의 오버플로를 나타냄
S	사인	산술 연산의 결과 값에 대한 부호를 포함
I	인터럽트	키보드 입력과 같은 외부 인터럽트의 처리 여부를 나타냄
P	패리티	연산 결과 1비트들의 개수에 따라 짝수, 홀수로 나타냄
D	디렉션	스트링 데이터를 이동시키거나 비교할 때 방향을 결정

4) 어셈블리어

- 어셈블리어란? 하드웨어와 소프트웨어의 가장 밑바닥에 있는 언어로 기계어와 거의 같은 수준에 있는 언어이다.

Opcode	Operand1	Operand2
ADD	EAX	EBX

- 위 그림의 경우 Operand 2(EBX)가 Source이고 Operand 1(EAX)이 Destination 이 되며 Opcode(명령어) 에 따라 EBX + EAX를 하여 EAX에 저장한다.

명령어	예제	설명	분류
push	push eax	eax의 값을 스택에 저장	스택 조작
pop	pop eax	스택 가장 상위에 있는 값을 꺼내서 eax에 저장	스택 조작
mov	mov eax, ebx	메모리나 레지스터의 값을 옮길때 사용	데이터 이동
inc	lnc eax	eax의 값을 1증가시킨다 (++)	데이터 조작
dec	dec eax	eax의 값을 1감소시킨다 (--)	데이터 조작
add	add eax, ebx	레지스터나 메모리의 값을 덧셈할때 쓰인다.	논리, 연산
sub	sub eax, ebx	레지스터나 메모리의 값을 뺄셈할때 쓰인다.	논리, 연산
call	call proc	프로시저를 호출한다.	프로시저
ret	ret	호출했던 바로 다음 지점으로 이동	프로시저
cmp	cmp eax, ebx	레지스터와 레지스터의 값을 비교	비교
jmp	jmp proc	특정한 곳으로 분기	분기
int	int $0x80	OS에 할당된 인터럽트 영역을 system call	인터럽트
nop	nop	아무 동작도 하지 않는다. (No Operation)	

나. 분류 알고리즘 학습

- 지도학습의 일종으로 기존에 존재하는 데이터의 Category 관계를 파악하고, 새롭게 관측된 데이터의 Category를 스스로 판별하는 과정이다.

1) SVM

- 두 개의 영역을 임의의 선으로 나누었을 때 그 선이 양쪽에 있는 점들과의 거리가 최대가 되게 하는 것.
- 선형분리를 통해 두 개의 영역을 나누고 선을 기준으로 양면에서 가장 가까운 점 2개의 거리를 계산 후 중심선과 합동이게 2개의 선을 긋는다.
- 최종 결과로는 2개의 선과 합동인 중간선을 찾아 데이터를 분류할 수 있게 된다.

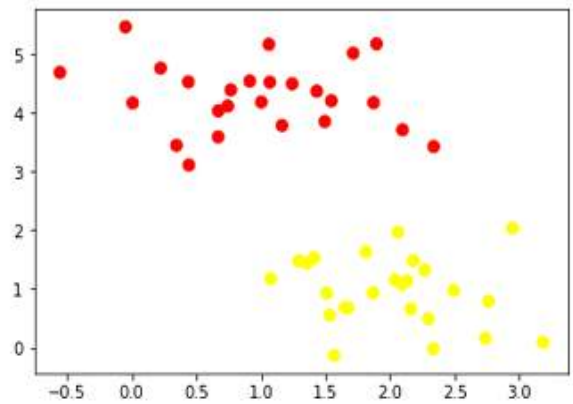
가) SVM 실습

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets.samples_generator import make_blobs
from sklearn.svm import SVC
```

▶ SVM 실습을 하는 데 필요한 모듈들을 불러온다.

```
X, y = make_blobs(n_samples=50, centers=2, random_state=0, cluster_std=0.60)
plt.scatter(X[:,0], X[:, 1], c=y, s=50, cmap='autumn')
```

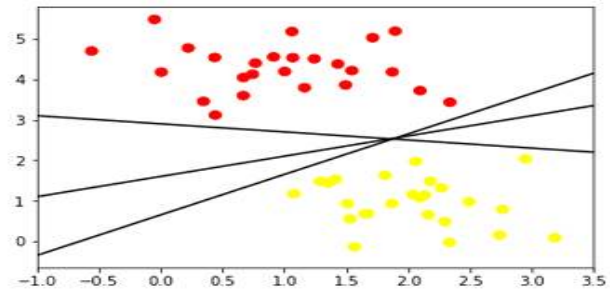
```
<matplotlib.collections.PathCollection at 0x7f18762f9fd0>
```



▶ 그래프에 데이터들을 표현하기 위해서는 가공되지 않은 데이터들이 필요하다.

▶ 겹치지 않게 중심을 2번 잡고 각 중심을 기준으로 50개씩 데이터를 생성한다.

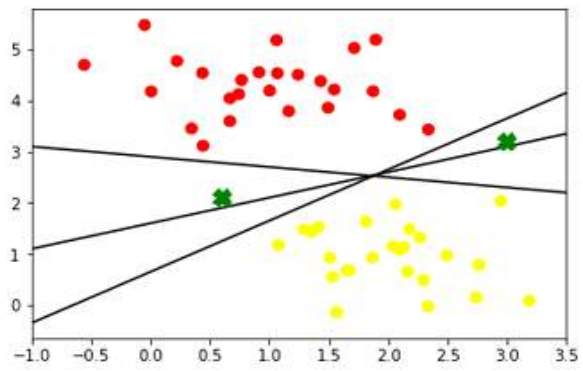
```
xfit = np.linspace(-1, 3.5)
plt.scatter(X[:,0], X[:,1], c=y, s=50, cmap='autumn')

for m,b in [(1,0.65), (0.5, 1.6), (-0.2, 2.9)]:
  plt.plot(xfit, m*xfit+b, '-k')

plt.xlim(-1, 3.5);
```



▶ 두 중심으로부터 생성된 데이터들을 겹치지 않게끔 지나는 선을 생성한다.

```
xfit = np.linspace(-1, 3.5)
plt.scatter(X[:,0], X[:,1], c=y, s=50, cmap='autumn')

for m,b in [(1,0.65), (0.5, 1.6), (-0.2, 2.9)]:
  plt.plot(xfit, m*xfit+b, '-k')

plt.xlim(-1, 3.5);

plt.plot([0.6], [2.1], 'X', color='Green', markeredgewidth=2, markersize=10)
plt.plot([3.0], [3.2], 'X', color='Green', markeredgewidth=2, markersize=10)
```

```
[<matplotlib.lines.Line2D at 0x7f18718f3c90>]
```



▶ 3개의 선 사이에 데이터 X가 추가되었을 때 Y축 3.1부터 시작되는 선과 0.5로부터 시작되는 선 2개가 존재하는데, 이 3.1선으로 비교했을 때와 0.5선으로 비교했을 때에 결괏값이 다를 수도 있다.

▶ 이 문제를 해결하기 위해 SVM을 이용해 최적의 선(즉 평균)을 찾는다.

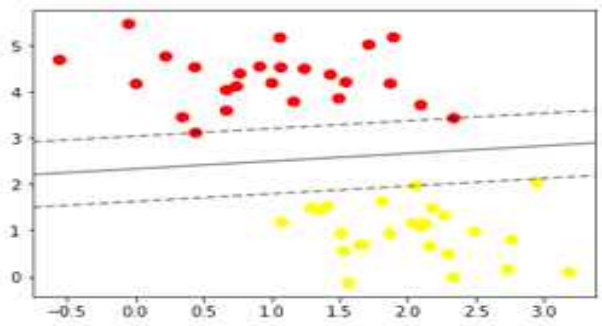
```
model = SVC(kernel='linear', C=1E10) # Support Vector Classifier
model.fit(X, y)
```

```
SVC(C=10000000000.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

▶ 사이킷런 모듈에서 SVC를 사용할 것이고, 커널의 형태를 직선으로 정의하며 fit 함수를 통해 학습을 완료한다.

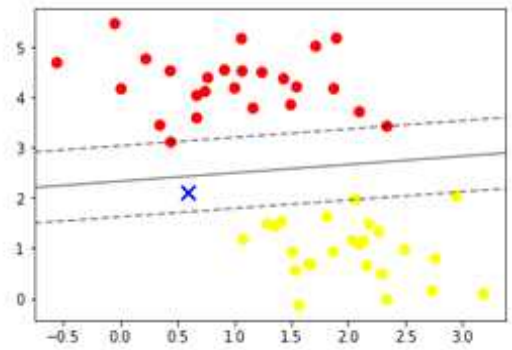
```
def plot_svc_decision_function(model, ax=None, plot_support=True):
    """Plot the decision function for a 2D SVC"""
    if ax is None:
        ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()

    # create grid to evaluate model
    x = np.linspace(xlim[0], xlim[1], 30)
    y = np.linspace(ylim[0], ylim[1], 30)
    Y, X = np.meshgrid(y, x)
    xy = np.vstack([X.ravel(), Y.ravel()]).T
    P = model.decision_function(xy).reshape(X.shape)

    # plot decision boundary and margins
    ax.contour(X, Y, P, colors='k',
               levels=[-1, 0, 1], alpha=0.5,
               linestyles=['--', '-', '--'])

    # plot support vectors
    if plot_support:
        ax.scatter(model.support_vectors_[:, 0],
                   model.support_vectors_[:, 1],
                   s=300, linewidth=1, facecolors='none');
    ax.set_xlim(xlim)
    ax.set_ylim(ylim)
```

```
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plot_svc_decision_function(model);
```



▶ SVM을 통해 중심선(직선)을 기준으로 위아래의 데이터들을 분류하였고 그중에서 가장 중심선과 가까운 데이터들을 1개씩 잡은 후 중심선과 합동이게 긋는다(점선).

```
plt.scatter(X[:, 0], X[:, 1], c=y, s=50, cmap='autumn')
plt.plot([0.6], [2.1], 'x', color='blue', markeredgewidth=2, markersize=10)
plot_svc_decision_function(model);
```

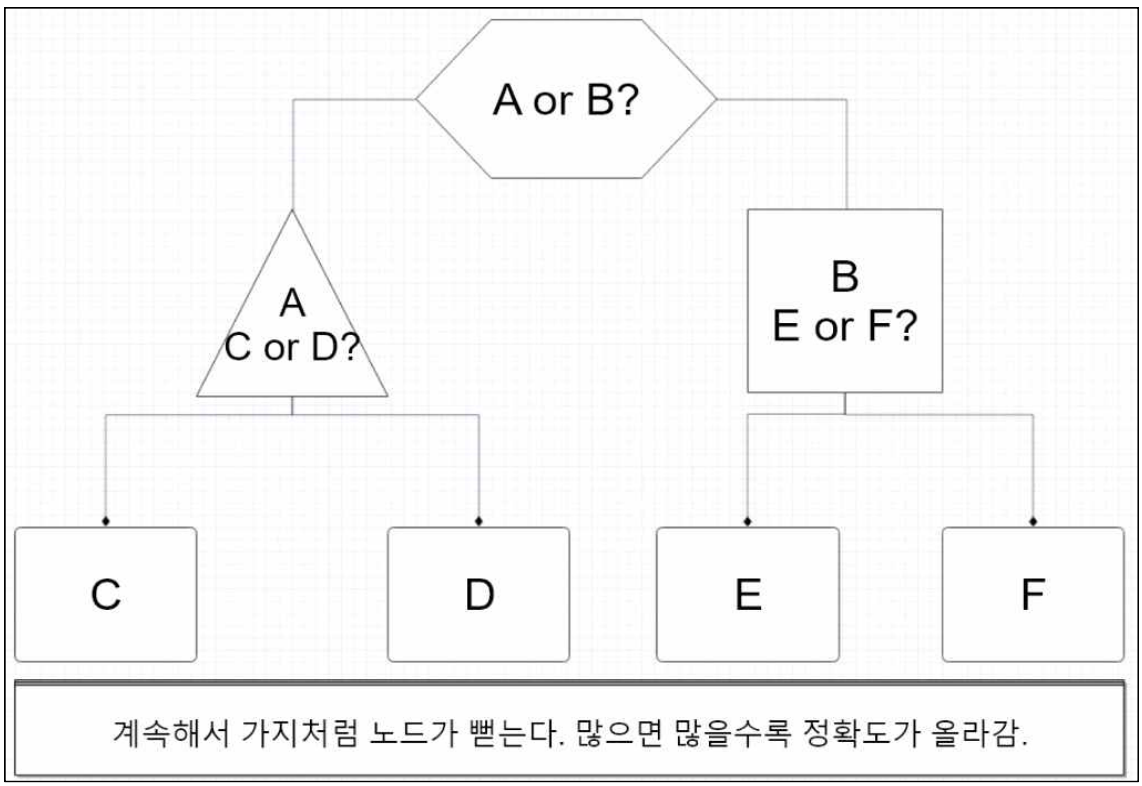


▶ X라는 임의의 데이터가 들어와도 SVM으로 만들어진 중심선에 의해 데이터를 정확히 분류할 수 있다.

2) randomforest

가) 의사결정 트리

- 설계자가 만든 질문들로 이루어진 트리들로 구성된다.
- 최상단에 있는 부모 노드의 첫 번째 질문으로부터 시작해 일정한 결괏값에 따라 자식 노드로 내려가며 그 과정에서 일어나는 결정들의 특징들이 군집 되어 있는 부분을 추출해 가장 좋은 결과가 나올 확률이 높은 부분을 보장하는 방식으로 구성된다.



나) randomforest 실습

- 하위 집합 기능을 이용해 다양한 옵션들을 빠르게 평가할 수 있다.
- 다수의 의사결정 트리로부터 들어오는 결괏값들의 집합 지점이 가장 많은 부분을 선택함으로써 의사결정 트리의 성능을 개선하는 모델.

```
import math


P_slow = 0.5
P_fast = 0.5

Entropy = - P_slow * math.log(P_slow, 2) - P_fast * math.log(P_fast, 2)
Entropy
```

```
1.0
```

- 계산 후 나올 엔트로피 값과 비교하기 위해 엔트로피 계산.

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.model_selection import train_test_split
```

```
mr = pd.read_csv('agaricus-lepiota.data', header=None)
mr.head(3)
```

	0	1	2	3	4	5	6	7	8	9	...	13	14	15	16	17	18	19	20	21	22
0	p	x	s	n	t	p	f	c	n	k	...	s	w	w	p	w	o	p	k	s	u
1	e	x	s	y	t	a	f	c	b	k	...	s	w	w	p	w	o	p	n	n	g
2	e	b	s	w	t	l	f	c	b	n	...	s	w	w	p	w	o	p	n	n	m

3 rows × 23 columns

- 예제 데이터를 삽입 후, head 일부분을 표시해 정확하게 로드되었는지 확인한다.
- 예제 데이터는 UCI 머신러닝 독버섯 데이터셋을 이용했음.
- 이 과정에서 ,으로 구분된 데이터들이 한꺼번에 정렬된다.

```
In [39]: label = []
         data = []
         attr_list = []

         for row_index, row in mr.iterrows():
             label.append(row.loc[0])
             row_data = []

             for v in row.loc[1:]:
                 row_data.append(ord(v))

             data.append(row_data)
```

```
In [40]: data[:1]

Out[40]: [[120,
           115,
           110,
           116,
           112,
           102,
           99,
           110,
           107,
           101,
           101,
           115,
           115,
           119,
           119,
           112,
           119,
           111,
           112,
           107,
           115,
           117]]
```

- 데이터 내부에 있는 기호들의 가시성을 높이기 위해 숫자로 변환한다.
- 이 과정은 학습 전용 데이터와 테스트 전용 데이터로 나누기 전에 데이터의 형식에 맞게 가공하는 과정이다.

```
data_train, data_test, label_train, label_test = train_test_split(data, label)
```

```
clf = RandomForestClassifier()
clf.fit(data_train, label_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

```
predict = clf.predict(data_test)
predict
```

```
array(['p', 'e', 'e', ..., 'e', 'p', 'p'], dtype='|S1')
```

- 가공된 데이터를 학습 전용 데이터(Train)와 테스트 전용(test) 데이터로 2개를 생성한다.
- clf.fit구문에서 생성된 데이터를 기반으로 학습시키고 predict 함수를 통해 데이터를 예측한다.

```
ac_score = metrics.accuracy_score(label_test, predict)
cl_report = metrics.classification_report(label_test, predict)

print('currect = ', ac_score)
print('report = \n')
print(cl_report)
```

```
('currect = ', 1.0)
report =

             precision    recall  f1-score   support

          e       1.00      1.00      1.00      1031
          p       1.00      1.00      1.00      1000

avg / total       1.00      1.00      1.00      2031
```

- 결과를 테스트하면 정답률(currect)과 내역보고(report)내용이 출력된다.

3) naivebayes

- 나이브 베이즈 알고리즘은 베이즈 정리를 이용한 확률적 기계학습 알고리즘이다.
- 사전 확률을 바탕으로 사후 확률을 추론할 수 있다.
- 주어진 학습 데이터의 양이 적어도 다른 알고리즘에 비해 좋은 성능을 낸다.

가) naivebayes 실습

```
In [1]: from sklearn import datasets
        from sklearn.naive_bayes import GaussianNB
        import pandas as pd
        iris = datasets.load_iris()
        df_X=pd.DataFrame(iris.data)
        df_Y=pd.DataFrame(iris.target)
        df_X.head()
```

Out[1]:

	0	1	2	3
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

```
In [2]: df_Y.head()
```

Out[2]:

	0
0	0
1	0
2	0
3	0
4	0

- 우선 sklearn 모듈에 내장되어있는 데이터셋에서 예제 데이터를 불러오고, sklearn.naive_bayes 모듈에서 가우시안 나이브 베이즈를 불러온다.
- pandas 모듈은 예제 데이터들을 DataFrame 형식으로 변환해 시각성과 서로 다른 종류의 데이터 타입을 가질 수 있게 함.
- iris는 sklearn 데이터셋의 예제 데이터를 품고 있는 변수이다.

```
In [3]: gnb = GaussianNB()
        fitted=gnb.fit(iris.data,iris.target)
        y_pred=fitted.predict(iris.data)
        fitted.predict_proba(iris.data)[[1,48,51,100]]

Out[3]: array([[1.00000000e+000, 1.51480769e-017, 2.34820051e-025],
               [1.00000000e+000, 2.63876217e-018, 2.79566024e-025],
               [7.27347795e-102, 9.45169639e-001, 5.48303606e-002],
               [3.23245181e-254, 6.35381031e-011, 1.00000000e+000]])
```

- gnb는 가우시안 나이브 베이즈 모델을 선언한 것이며 gnb.fit(iris.data,iris.target) 코드를 통해 iris안에 있는 예제 데이터들을 기반으로 학습한다.
- 학습된 결과는 predict 함수를 통해 예측 결괏값을 얻을 수 있다. predict 함수는 데이터들을 넣었을 때 그 클래스에 속하는지 속하지 않는지를 나타내는 0 또는 1로 구성된 벡터를 반환한다. 이때 y_pred에 결괏값이 담긴다.
- redict_proba 함수 또한 predict와 유사한 것처럼 보이지만 벡터값이 아닌 0과 1 사이의 값으로 반환한다. 이때 proba는 iris의 각 target에 대한 데이터가 어떤 y 값을 가리키는지 확률을 체크한다.

```
In [4]: fitted.predict(iris.data)[[1,48,51,100]]

Out[4]: array([0, 0, 1, 2])
```

- predict 함수를 통해 y_pred 값을 예측한다. 이때 예측된 값은 y_pred에 들어가는 것이 아니다.

```
In [5]: from sklearn.metrics import confusion_matrix
        confusion_matrix(iris.target,y_pred)

Out[5]: array([[50,  0,  0],
               [ 0, 47,  3],
               [ 0,  3, 47]])
```

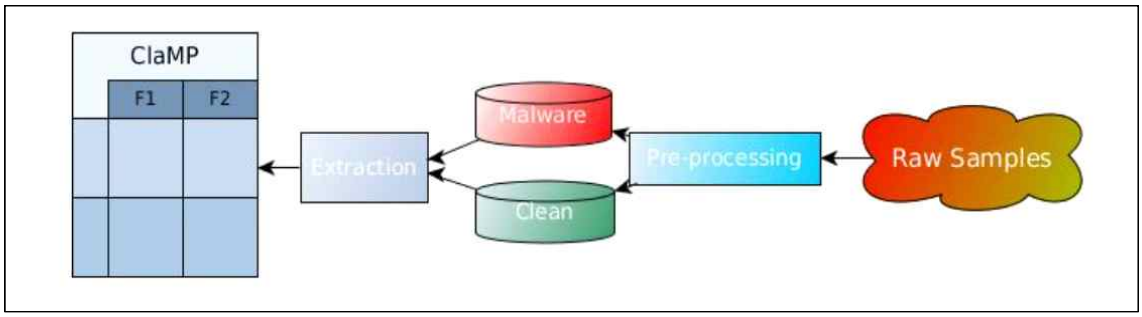
- 혼돈 행렬(confusion_matrix)을 통해 값 y_pres와 예측 결괏값이 얼마나 맞는지 합해준다.
- 3개의 오답이 발생했다.

다. 악성코드 특징 추출 및 분석

- 특징 추출은 pe 헤더 정보, N-gram을 이용한 어셈블러 특징 추출, 이미지 분석을 위한 악성코드 이미지화로 총 3가지를 진행했다.
- 특징 분석은 SVM, randomforest, naivebayes, dnn, cnn을 적용해 분석했다.

1) pe_header 추출

- PE header 특징은 ClaMP라는 오픈소스 프로젝트에서 제공하는 스크립트를 이용했다.



<그림 30> ClaMP 추출도

가) PE 특징 추출(pe_header.py 코드 분석)

(1) main():

```
def main():

    source_path= raw_input("Enter the path of samples (ending with /) >>  ")
    output_file= raw_input("Give file name of output file. (.csv) >>")
    label = raw_input("Enter type of sample( malware(1)|benign(0))>>")

    features = pe_features(source_path,output_file,label)
    features.create_dataset()

if __name__ == '__main__':
    main()
```

- source_path(샘플 파일이 위치하는 폴더), output_file(결과를 저장할 csv파일 이름), label(악성코드(1), 정상 프로그램(0))을 입력받은 후 pe_features Class 생성
- create_dataset() 함수를 호출

(2) create_dataset()

```
    def create_dataset(self):
        self.write_csv_header()
        count = 0

        #run through all file of source and extract features
        for file in os.listdir(self.source):
                filepath = self.source + file
                data = self.extract_all(filepath)
                hash_ = self.getMD5(filepath)
                print "hash: ", hash_
                data.insert(0, hash_)
                data.insert(0, file)

                self.write_csv_data(data)
                count += 1
                print "Successfully Data extracted and written for {}.".format(file)
                print "Processed " + str(count) + " files"
```

- csv 헤더를 생성 후 source_path에 있는 파일을 불러와 extract_all() 함수로 특징을 추출

(3) extract_all()

- extract_all 함수는 간단하게 PE 헤더 파싱으로 뽑아낼 수 있는 Raw 특징을 추출한 후 PE 헤더 요솟값들을 한 번 더 해석하여 의미 있는 정보를 추출한다 (Derived)

```
def extract_all(self,filepath):
    data =[]
    #load given file
    try:
        pe = pefile.PE(filepath)
    except Exception, e:
        print "{} while opening {}".format(e,filepath)
    else:
        data += self.extract_dos_header(pe)
        data += self.extract_file_header(pe)
        data += self.extract_optional_header(pe)
        # derived features
        #number of suspicisou sections and non-suspicsious section
        num_ss_nss = self.get_count_suspicious_sections(pe)
        data += num_ss_nss
        # check for packer and packer type
        packer = self.check_packer(filepath)

        # Appending the packer info to the rest of features
        data += packer[0]
        entropy_sections = self.get_text_data_entropy(pe)
        data += entropy_sections
        f_size_entropy = self.get_file_entropy(filepath)
        data += f_size_entropy
        fileinfo = self.get_fileinfo(pe)
        data.append(fileinfo)
        data.append(self.type)

    return data
```

▶ Derived 추출 정보

- get_count_suspicious_sections: 섹션 이름 검사(정상/악성에서 많이 보이는 이름 확인)
- check_packer: 파일에 적용된 패커 알고리즘을 검사(yara룰셋 사용)
- get_text_data_entropy: 코드와 데이터 섹션의 엔트로피 값 계산
- get_file_entropy: 파일 전체의 엔트로피 값 계산
- get_fileinfo: 파일 버전, 제품 버전, 제품 이름, 회사 이름 조회

(4) check_packer()

▶ 패커를 식별해주는 함수

```
def check_packer(self,filepath):

    result=[]
    matches = self.rules.match(filepath)

    try:
        if matches == [] or matches == {}:
            result.append([0,"NoPacker"])
        else:
            result.append([1,matches['main'][0]['rule']])
    except:
        result.append([1,matches[0]])

    return result
```

- 위 코드에서 rules 변수에 담긴 데이터 정보와 파일을 비교해 패커를 찾아내는 것을 확인할 수 있다.

```
def __init__(self,source,output,label):
    self.source = source
    self.output = output
    self.type = label
    #Need PEiD rules compile with yara
    self.rules= yara.compile(filepath='./peid.yara')
```

- __init__에 정의된 rules를 찾아가 보면 peid.yara라는 파일을 담고 있는 것을 확인할 수 있음.

```
rule MSLRHv032afakePCGuard4xxemadicius
{
strings:
        $a0 = { FC 55 50 E8 00 00 00 00 5D EB 01 E3 60 E8 03 00 00 00 D2 EB 0B 58 EB 01
48 40 EB 01 35 FF E0 E7 61 58 5D EB 05 E8 EB 04 40 00 EB FA E8 0A 00 00 00 E8 EB 0C 00 00 E8 F6
FF FF FF E8 F2 FF FF FF 83 C4 08 74 04 75 02 EB 02 EB 01 81 50 E8 02 00 00 00 29 5A 58 6B C0 03
E8 02 00 00 00 29 5A 83 C4 04 58 74 04 75 02 EB 02 EB 01 81 0F 31 50 0F 31 E8 0A 00 00 00 E8 EB
0C 00 00 E8 F6 FF FF FF E8 F2 FF FF FF }

condition:
        $a0 at entrypoint
}
```

<그림 36> peid.yara

2) 정적 코드 패턴 추출

가) N-gram 추출 방식

1. 4-gram 패턴 추출
2. 패턴 개수 상위 100의 특징만 선택: 패턴 이름만 남기고 개수는 제거
3. 개별 파일을 대상으로 100개 특징에 해당하는 패턴 개수를 실제 특징값으로 사용

나) N-gram 추출 (ngram.py)

(1) main()

```
def main():

    num_of_features = 100

    mal_path = '../samples/malware/'
    nor_path = '../samples/normal/'
    output_file = "./ngram.csv"

    print '[*] Extracting ngram patterns from files'

    ef = NGRAM_features(output_file)
    i = 0
    #악성코드 폴더에서 4-gram 추출
    for file in os.listdir(mal_path):
        i += 1
        print "%d file processed (%s)," % (i, file),
        file = mal_path + file
        byte_code = ef.get_opcodes(0, file)
        grams = ef.n_grams(4, byte_code, 1)
        print "%d patterns extracted" % (len(grams))

    print '- Malware Completed'
    #정상 프로그램에서 4-gram 추출
    for file in os.listdir(nor_path):
        i += 1
        print "%d file processed (%s)," % (i, file),
        file = nor_path + file
        byte_code = ef.get_opcodes(0, file)
        grams = ef.n_grams(4, byte_code, 1)
        print "%d patterns extracted" % (len(grams))
    print '- Normal Completed'


    print "[*] Total length of 4-gram list :", len(grams)
    # 상위 100개 빈도를 가지는 패턴을 뽑아 csv 헤더로 기록
    sorted_x = sorted(grams.items(), key=operator.itemgetter(1), reverse=True)
    print "[*] Using %s grams as features" % (num_of_features)
    features = sorted_x[0:num_of_features]
    headers = list(chain.from_iterable(zip(*features)))[0:num_of_features]
    ef.write_csv_header(headers)

    print "#" * 80
```

```
    # 상위 100개 빈도를 가지는 패턴을 뽑아 csv 헤더로 기록
    sorted_x = sorted(grams.items(), key=operator.itemgetter(1), reverse=True)
    print "[*] Using %s grams as features" % (num_of_features)
    features = sorted_x[0:num_of_features]
    headers = list(chain.from_iterable(zip(*features)))[0:num_of_features]
    ef.write_csv_header(headers)

    print "#" * 80

    #malware 추출
    i = 0
    for file in os.listdir(mal_path):
        i += 1
        print "%d file processed (%s)," % (i, file)
        filepath = mal_path + file
        byte_code = ef.get_opcodes(0, filepath)
        grams = ef.n_grams(4, byte_code, 0)     #파일에서 4-gram 추출
        gram_count = ef.get_ngram_count(headers, grams, 1)  # 100개 특징에 해당하는 빈도 값 추출
        hash_ = ef.getMD5(filepath)
        all_data = [file, hash_]
        all_data.extend(gram_count)
        ef.write_csv_data(all_data)
```

- 정상 프로그램도 악성코드 추출과 동일하게 진행한다.

(2) get_opcodes()

```
# 프로그램 파일에서 코드 섹션을 찾아 어셈블리어로 변환
# capstone 라이브러리를 사용하여 disassamble 했다.
    def get_opcodes(self, mode, file):

        asm = []

        pe = pefile.PE(file)
        byte_all = []

        ep = pe.OPTIONAL_HEADER.AddressOfEntryPoint
        end = pe.OPTIONAL_HEADER.SizeOfCode
        ep_ava = ep+pe.OPTIONAL_HEADER.ImageBase
        for section in pe.sections:
            addr = section.VirtualAddress
            size = section.Misc_VirtualSize

            if ep > addr and ep < (addr+size):
                #print(section.Name)
                ep = addr
                end = size

        data = pe.get_memory_mapped_image()[ep:ep+end]
        offset = 0

        temp = data.encode('hex')
        temp = [temp[i:i+2] for i in range(0,len(temp), 2)]

        if(mode):
            return temp

        md = Cs(CS_ARCH_X86, CS_MODE_32)
        md.detail = False

        for insn in md.disasm(data, 0x401000):
            #print("0x%x:\t%s\t%s" % (insn.address, insn.mnemonic, insn.op_str))
            #print(insn.mnemonic)
            asm.append(insn.mnemonic)

        return asm
```

- 이 함수에서 추출한 어셈블리 코드를 n_grams() 함수에서 추출한다.

(3) n_grams()

```
def n_grams(self, num, asm_list, ex_mode):
    if ex_mode == 1:
        gram = self.gram
    elif ex_mode == 0:
        gram = dict()

    gen_list = self.gen_list_n_gram(num, asm_list)

    for lis in gen_list:
        lis = " ".join(lis)
        try:
            gram[lis] += 1
        except:
            gram[lis] = 1

    return gram
```

- 4-gram 단위로 추출한다.

3) 바이너리 -> 이미지 변환

가) image.py()

(1) get_image()

```
def get_image(self, path, file):

    filename = path + file

    f = open(filename,'rb')
    ln = os.path.getsize(filename) # 파일 길이(바이트 단위)

    width = int(ln**0.5)          # 파일 길이의 제곱근을 구함(정사각형 모양 이미지 만들기)
    rem = ln % width

    a = array.array("B")          # uint8 배열
    a.fromfile(f,ln-rem)          # 파일의 바이너리로 배열을 구성
    f.close()

    g = np.reshape(a, (int(len(a)/width), width)) # 정사각형 모양의 배열을 그대로 이미지 형태로 저장
    g = np.uint8(g)

    fpng = self.out_path + file + ".png"
    scipy.misc.imsave(fpng, g)

    outfile = self.out_path + file + "_thumb.png"
    print(outfile)
    size = 256, 256

    if fpng != outfile:
        im = Image.open(fpng)
        im.thumbnail(size, Image.ANTIALIAS)       # 이미지를 256x256 크기의 썸네일로 생성
        im.save(outfile, "PNG")
```

4) 특징 추출 결과

구분	파일 이름
PE 헤더	mal.csv nom.csv
4-gram	Ngram.scv
이미지	Images/malware/*.png Images/normal/*.png

라. 특징 분석

- 뽑아낸 특징을 분류 알고리즘에 넣어 최적의 특징 조합을 찾는 분석을 진행

1) 분류 알고리즘 (model.py)

- model.py에 그동안 학습한 분류 알고리즘으로 수행 함수가 정의되어있다.

함수	기능 (옵션)
do_svm	SVM모델 학습 및 평가 (기본)
do_randomforest	랜덤포렌스트 모델 학습 및 평가 (기본)
do_naivebayes	나이브 베이즈 모델 학습 및 평가 (기본)
do_dnn	총 4개의 히든 레이어를 가진 딥 뉴럴 네트워크

```
def do_svm(self):
        clf = SVC()
        clf.fit(self.x_train, self.y_train)
        y_pred = clf.predict(self.x_test)

        return accuracy_score(self.y_test, y_pred)

def do_randomforest(self, mode):

        clf = RandomForestClassifier()
        clf.fit(self.x_train, self.y_train)

        if mode == 1:
                return clf.feature_importances_
        y_pred = clf.predict(self.x_test)

        return accuracy_score(self.y_test, y_pred)

def do_naivebayes(self):
        clf = GaussianNB()
        clf.fit(self.x_train, self.y_train)
        y_pred = clf.predict(self.x_test)

        return accuracy_score(self.y_test, y_pred)
```

```
def do_dnn(self):

        if "Series" in str(type(self.y_train)):
                self.y_train = self.y_train.to_frame()
                self.y_test = self.y_test.to_frame()
                input_len = len(self.x_train.columns)
        else:
                self.y_train = self.y_train.reshape(len(self.y_train), 1)
                self.y_test = self.y_test.reshape(len(self.y_test), 1)
                input_len = np.size(self.x_train, 1)

        learning_rate = 0.001
        batch_size = 128
        training_epochs = 15
        keep_prob = 0.5

        x_train = self.x_train
        y_train = self.y_train

        X = tf.placeholder(tf.float32, [None, input_len])
        Y = tf.placeholder(tf.float32, [None, 1])

        W1 = tf.Variable(tf.random_normal([input_len, 1024]), name='weight1')
        b1 = tf.Variable(tf.truncated_normal([1024]), name='bias1')
        L1 = tf.sigmoid(tf.matmul(X, W1) + b1)

        W2 = tf.Variable(tf.random_normal([1024, 128]), name='weight4')
        b2 = tf.Variable(tf.truncated_normal([128]), name='bias4')
        L2 = tf.sigmoid(tf.matmul(L1, W2) + b2)

        W3 = tf.Variable(tf.random_normal([128, 1]), name='weight5')
        b3 = tf.Variable(tf.truncated_normal([1]), name='bias5')
```

```
output = tf.sigmoid(tf.add(tf.matmul(L2, W3), b3))

cost = -tf.reduce_mean(Y * tf.log(output) + (1 - Y) * tf.log(1 - output))
train = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)

predicted = tf.cast(output > 0.5, dtype=tf.float32)
accuracy = tf.reduce_mean(tf.cast(tf.equal(predicted, Y), dtype=tf.float32))

with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())

        for epoch in range(training_epochs):
                avg_cost = 0
                total_batch = int(len(x_train) / batch_size)

                for i in range(total_batch-1):
                        batch_xs = x_train[i*batch_size:(i+1)*batch_size]
                        batch_ys = y_train[i*batch_size:(i+1)*batch_size]

                        _ , c =sess.run([train, cost], feed_dict={X: batch_xs, Y: batch_ys})
        |               print "Epoch :", epoch, "cost: ", c

        acc =  sess.run(accuracy, feed_dict={X: self.x_test, Y: self.y_test})

return acc
```

2) 특징 학습

가) hot_encoding()

- packer_type의 문자형 자료를 수치형으로 변환해주는 함수

```
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder

def hot_encoding(df):

    enc = OneHotEncoder(handle_unknown='ignore', sparse=False)
    lab = LabelEncoder()

    dat = df['packer_type']
    lab.fit(dat)
    lab_dat = lab.transform(dat)

    df = df.drop('packer_type', 1)
    lab_dat = lab_dat.reshape(len(lab_dat), 1)
    enc_dat = enc.fit_transform(lab_dat)
    enc_dat = pd.DataFrame(enc_dat, columns=lab.classes_)

    df = df.reset_index(drop=True)
    enc_dat = enc_dat.reset_index(drop=True)

    df = pd.concat([df, enc_dat], axis=1)

    return df, lab.classes_
```

그림 47. hot_encoding

나) 모델 학습

```
import numpy as np
import pandas as pd
import seaborn as sns

import matplotlib.pyplot as plt
import model
import operator
import cnn_model

cols = ["svm", "randomforest", "naivebayes", "dnn"]
df = pd.DataFrame(columns=cols)

# PE 특징 데이터 로드
pe_nor = pd.read_csv('nom.csv')
pe_mal = pd.read_csv('mal.csv')
pe_all = pd.concat([pe_nor, pe_mal])  # 832 x 72

# ngram 특징 데이터 로드
gram_all = pd.read_csv('ngram.csv')   # 791 x 103
```

그림 48. 모듈 임포트 및 특징 데이터 로드

```
print "[*] Before Filtering NA values: ", pe_all.shape
NA_values = pe_all.isnull().values.sum()
print "[*] Missing Values: ", NA_values
pe_all = pe_all.dropna()
print "[*] After Filtering NA values: ", pe_all.shap
```

그림 49. 누락 값 제거

```
[*] Before Filtering NA values:   (832, 72)
[*] Missing Values:   14830
[*] After Filtering NA values:   (591, 72)
```

그림 50. 누락 값 제거 후 (누락 값을 가진 241개의 행 제거)

```
pe_all_tmp = pe_all  # 데이터 백업
pe_all = pe_all.drop(['filename', 'MD5', 'packer_type'], 1) # 파일이름, MD5, packer_type 열 제거

Y = pe_all['class'] # 카테고리 열을 별도로 추출
X = pe_all.drop('class', 1) # 카테고리 열 제거
Y_bak = Y # 뒤에서 진행할 특징 선택 작업을 위해 데이터 백업

md_pe = model.Classifiers(X, Y)  # 학습 모듈 인스턴스 초기화
df.loc['pe'] = md_pe.do_all()  # 분류 모델 학습
```

그림 51 packer_type 제거 학습

```
pe_all = pe_all_tmp
pe_all = pe_all.drop(['filename', 'MD5'], 1)  # 파일이름, MD5 열 제거

pe_all, classes_ = hot_encoding(pe_all)  # One-Hot 인코딩 변환

print "Found %d Categories in packer-type" % len(classes_)
# dataset for modeling
pe_all = pd.DataFrame(pe_all)
pe_all.to_csv('pe_packer.csv', index=False)

Y = pe_all['class']  # 카테고리 열을 별도로 추출
X = pe_all.drop('class', axis=1)

md_pe_packer = model.Classifiers(X, Y) # 학습 모듈 인스턴스 초기화
df.loc['pe_packer'] = md_pe_packer.do_all() # 분류 모델 학습
```

그림 52 packer_type encoding(pe_packer.csv 생성) 적용 학습

```
gram_all = gram_all.drop(['filename', 'MD5'], 1) # 파일이름, MD5 열 제거

# dataset for modeling
# gram_all.to_csv('../3-modeling/ngram.csv', index=False)

Y = gram_all['class'] # 카테고리 열을 별도로 추출
X = gram_all.drop('class', 1) # 카테고리 열 제거
md_gram = model.Classifiers(X, Y) # 학습 모듈 인스턴스 초기화
df.loc['ngram'] = md_gram.do_all()# 분류 모델 학습
df.loc['image'] = [0,0,0,0]
```

그림 53 ngram 특징 학습

```
cn = cnn_model.CNN_tensor()
cn.load_images()
cnn_acc = cn.do_cnn()
```

그림 54 cnn 특징 학습

3) 특징 분석

```
avg_pe = df.loc['pe'].mean(axis=0)
avg_pe_packer = df.loc['pe_packer'].mean(axis=0)
avg_ngram = df.loc['ngram'].mean(axis=0)

df['cnn'] = [0,0,0,cnn_acc]
df['avg'] = [avg_pe, avg_pe_packer, avg_ngram, cnn_acc]
df
```

	svm	randomforest	naivebayes	dnn	cnn	avg
pe	0.554622	0.941176	0.521008	0.857143	0.000000	0.718487
pe_packer	0.554622	0.974790	0.521008	0.865546	0.000000	0.728992
ngram	0.849057	0.886792	0.773585	0.867925	0.000000	0.844340
image	0.000000	0.000000	0.000000	0.000000	0.886792	0.886792

가) svm
- 패커 정보를 추가해도 동일한 결괏값을 보여준다.
- ngram의 경우 84% 정도의 정확도로 pe 특징보다 약 30% 높은 정확도를 보여준다.

나) randomforest
- pe 특징은 94% 패커까지 포함한 pe 특징의 경우 97%의 정확도를 보여준다.
- ngram 특징의 경우도 88%로 높은 정확도를 보여줬다.

다) naivebayes
- 패커 정보를 추가해도 동일한 결괏값을 보여준다.
- ngram의 경우 pe 특징보다는 정확도가 높으나, 다른 모델들에 비하면 좋은 결과라고 보기 어렵다.

라) dnn
- 패커 정보 추가 시 1% 정도 정확도가 올랐다.
- pe 정보와 ngram의 경우도 큰 차이가 없는 것으로 보인다.

마) cnn
- cnn의 경우 ngram + randomforest를 사용한 결과와 같은 수치를 보여줬다.

◎ 결론
- 평균값을 기준으로 볼 때 ngram 특징이 가장 높은 정확도를 보여줬다.
- 가장 높은 정확도를 보여준 특징은 pe_packer(패커를 포함한 특징)이다.
- 패커 정보를 추가하는 것이 패커 정보를 제거하는 것보다 높은 정확도를 보여준다.
- randomforest 모델이 가장 좋은 성능을 보여 준다.

마. 모델링

- 프로그램의 다양한 측면을 모델에 포함할 수 있으며, 다른 특징에 비해 더 높은 정확도를 보여준 pe_packer를 최종 특징으로 선택하고 randomforest 모델로 학습했다.

1) 최적의 매개변수로 학습

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import model

df = pd.read_csv('pe_packer.csv')

df.shape
```

그림 56 import 및 csv 불러오기

```
Y = df.loc[:, '68']
df = df.drop('68', axis=1)
X = df

md = model.Classifiers(X, Y)
acc = md.do_randomforest(0)
acc
```

```
/home/master/anaconda3/envs/
e default value of n_estimat
  "10 in version 0.20 to 100

0.9361702127659575
```

그림 57 randomforest 모델 학습

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

#n_estimators(판단에 사용할 트리 개수, max_feature(노드 분할 시 고려할 랜덤 특정 조합 크기)
parameters = {'n_estimators':[100, 200, 500, 1000], 'max_features':['auto', None]}

rfc = RandomForestClassifier() #기본 랜덤 포레스트 분류기 초기화
clf = GridSearchCV(rfc, parameters, cv=10) #GridSearchCV 초기화
clf.fit(X, Y) #GridSearchCV 수행
```

```
GridSearchCV(cv=10, error_score='raise-deprecating',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid={'n_estimators': [100, 200, 500, 1000], 'max_features': ['auto', None]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

그림 58 최적의 매개변수 조합 확인

- 아래 두 값은 랜덤 포레스트의 매개변수 옵션으로 판단에 영향을 미친다.

▶ n_estimators: 판단에 사용할 트리 개수 (값이 클수록 좋은 결과를 얻을 수 있음.)

▶ max_features: 최적의 트리 분할에 사용할 특징 개수 (auto=sqrt(n_features)로 가장 좋음)

```
from sklearn.model_selection import cross_val_score
#from sklearn.externals import joblib

clf = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=200, n_jobs=None,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)

clf.fit(X, Y)

scores = cross_val_score(clf, X, Y, cv=10)
print scores
print np.mean(scores)

#joblib.dump(clf, 'model.joblib')
```

```
[0.94680851 0.95744681 1.         1.         1.         0.9893617
 0.9893617  0.94623656 0.92473118 0.98924731]
0.9743193777167697
```

- 교차 검증 결과 97.4%의 정확도가 나왔다.
- 맨 밑에 주석 처리되어있는 부분을 해제하고 실행하여 model.joblib(라이브러리)로 저장한다.

바. 인공지능 백신

1) 모델 배치 (pe_packer + randomforest)

▶ 실제로 백신 엔진에 탑재할 부분만 하나의 코드로 모았다.

- 특징 추출: pe_headers.py에 정의된 PE 특징 추출 기능(peid.yara파일도 필요함)
- 모델 파일: 랜덤 포레스트 모델을 저장한 model.joblib 파일
- One-hot 인코딩 범주: peid_yara 패턴 중 실제 특징으로 사용한 패턴 목록 파일
- 판단 기능: 위에서 만든 단일모델 파일을 모듈로 작성

가) PE 헤더 특징 추출(extract.py)

▶ pe_headers.py에서 csv 파일로 저장하는 기능과 관련된 함수를 모두 제거한다.

▶ __init__과 extract_all 함수를 수정한다.

```
def __init__(self,source,output,label):
    self.source = source
    self.output = output
    self.type = label
    #Need PEiD rules compile with yara
    self.rules= yara.compile(filepath='./peid.yara')
```

그림 60 __init__ 수정 전

```
def __init__(self,source): #판단 대상 파일 경로만 인자로 받도록 수정
    self.source = source
    self.rules= yara.compile(filepath='./peid.yara')
```

그림 61 __init__ 수정 후

```
def extract_all(self,filepath):
    data =[]

    try:
        pe = pefile.PE(filepath)
    except Exception, e:
        print "{} while opening {}".format(e,filepath)
    else:
        data += self.extract_dos_header(pe)
        data += self.extract_file_header(pe)
        data += self.extract_optional_header(pe)

        num_ss_nss = self.get_count_suspicious_sections(pe)
        data += num_ss_nss

        packer = self.check_packer(filepath)

        data += packer[0]
        entropy_sections = self.get_text_data_entropy(pe)
        data += entropy_sections
        f_size_entropy = self.get_file_entropy(filepath)
        data += f_size_entropy
        fileinfo = self.get_fileinfo(pe)
        data.append(fileinfo)
        data.append(self.type)

    return data
```

그림 62 extract_all 수정 전

```
def extract_all(self):
    data =[]
    filepath = self.source
    try:
        pe = pefile.PE(filepath)
    except Exception, e:
        print "{} while opening {}".format(e,filepath)
    else:
        data += self.extract_dos_header(pe)
        data += self.extract_file_header(pe)
        data += self.extract_optional_header(pe)
        num_ss_nss = self.get_count_suspicious_sections(pe)
        data += num_ss_nss
        packer = self.check_packer(filepath)
        data += packer[0]
        entropy_sections = self.get_text_data_entropy(pe)
        data += entropy_sections
        f_size_entropy = self.get_file_entropy(filepath)
        data += f_size_entropy
        fileinfo = self.get_fileinfo(pe)
        data.append(fileinfo)
        magic = pe.OPTIONAL_HEADER.Magic

    return data, magic
```

그림 63 extract_all 수정 후

나) 악성코드 확률 분석 파일 (engine.py)

▶ pe_header.py, patterns.csv(pe_packer 특징), 랜덤 포레스트 모델을 모아서 engine.py 에 작성해준다.

▶ 테스트를 통해 이상 없이 작동하는지 확인 해줬다.

```
from sklearn.externals import joblib
import numpy as np
import extract
import csv

filepath = '/home/stud/PJ/PJ1_malware/samples/test/malware' # 판단 대상 파일 경로

ft = extract.PE_features(filepath)
data, magic = ft.extract_all()
# PE 특징 추출
# data[63] = packer_type
# magic = 32bit or 64bit ( 267 = 32bit )
# len(data) = 69 : ok

if magic != 267:                        # 매직 넘버가 32bit가 아닐 때 오류처리
        print "64-bit File. cannot process"
elif len(data) != 69:                   # 추출한 특징 개수가 69개가 아닌 경우 오류처리
        print "File corrupted"

f = open('patterns.csv', 'r')           # one-hot 인코딩 패턴 목록 코드
rd = csv.reader(f)
for row in rd:
        patterns = row

packer_type = [0] * len(patterns)

try:
        idx = patterns.index(data[63])
except ValueError:
        idx = 10

packer_type[idx] = 1
del data[63]                            # 문자열 pe_packer 데이터 삭제
data = data + packer_type               # pe 데이터에 packer_type 연결
print len(data)
data = np.asarray(data).reshape((1, -1))
print data.shape
clf = joblib.load('model.joblib')       # 랜덤포레스트 모델 변수 로드
rns = clf.predict_proba(data)[0][1]     # 해당 파일이 악성코드일 확률 추출
print rns
```

```
(mlsec_27) master@ubuntu:~/Desktop/mal/PJ1_malware/4_engine$ python engine.py
87
(1, 87)
/home/master/anaconda3/envs/mlsec_27/lib/python2.7/site-packages/sklearn/base.py:253: UserWarning
: Trying to unpickle estimator DecisionTreeClassifier from version 0.19.2 when using version 0.20
.4. This might lead to breaking code or invalid results. Use at your own risk.
  UserWarning)
/home/master/anaconda3/envs/mlsec_27/lib/python2.7/site-packages/sklearn/base.py:253: UserWarning
: Trying to unpickle estimator RandomForestClassifier from version 0.19.2 when using version 0.20
.4. This might lead to breaking code or invalid results. Use at your own risk.
  UserWarning)
0.994
```

- 악성코드 1개 테스트 결과 99.4% 확률로 악성코드로 판별됐다.

2) 키콤 백신에 적용

가) 키콤 백신 코드 분석

(1) 키콤 실행 파일 분석 (k2.py)

▶ 백신 실행 역할을 하는 파일

```
    # 숨겨진 기능 (인공지능 AI을 위해 만든 옵션)

    parser.add_option("", "--feature",
                      type="int", dest="opt_feature",
                      default=0xffffffff)
```

- 숨겨진 기능을 따라가 보면 pe 특징 추출을 할 수 있는 코드가 나온다.

```
# ------------------------------------------------------------------------
# pe_parse(mm)
# PE 파일을 파싱하여 주요 정보를 리턴한다.
# 입력값 : mm - 파일 핸들
# 리턴값 : {PE 파일 분석 정보} or None
# ------------------------------------------------------------------------
def parse(self):
    mm = self.mm

    pe_format = {'PE_Position': 0, 'EntryPoint': 0, 'SectionNumber': 0,
                 'Sections': None, 'EntryPointRaw': 0, 'FileAlignment': 0}

    try:
        if mm[0:2] != 'MZ':  # MZ로 시작하나?
            raise ValueError

        dos_header = DOS_HEADER()
        ctypes.memmove(ctypes.addressof(dos_header), mm[0:], ctypes.sizeof(dos_header))

        # PE 표식자 위치 알아내기
        pe_pos = dos_header.e_lfanew

        # PE 인가?
        if mm[pe_pos:pe_pos + 4] != 'PE\x00\x00':
            raise ValueError

        pe_format['PE_Position'] = pe_pos
```

- 위 PE 추출 기능을 활용해서 엔진을 만들 것이다.

(2) 악성코드 분석 파일 (k2engine.py)

▶ 실질적인 악성코드 분석 파일

```
for i, inst in enumerate(self.kavmain_inst):
    try:
        ret, vname, mid, scan_state = inst.scan(mm, filename, fileformat, filename_ex)
        if ret:  # 악성코드 발견하면 추가 악성코드 검사를 중단한다.
            eid = i  # 악성코드를 발견한 플러그인 엔진 ID

            if self.verbose:
                print '    [-] %s.__scan_file() : %s' % (inst.__module__, vname)

            break
    except AttributeError:
        continue
```

- verbose 옵션이 켜져 있는 상태라면 탐지된 모듈이 출력된다.

3) 플러그인 추가

- 머신러닝은 확률 기반으로 탐지를 하므로 정상 파일도 탐지해내는 위험성이 있다. 따라서 기존 백신에 있던 모든 플러그인을 먼저 실행한 후 악성코드 여부를 탐지하지 못할 때 머신러닝 기능이 동작해야 한다.
- 악성일 확률이 정상일 확률보다 크다고 해서 무작정 악성으로 탐지하는 것이 아니라 임계치를 설정해서 오진을 줄여야 한다.

가) 만든 플러그인(ml.py)

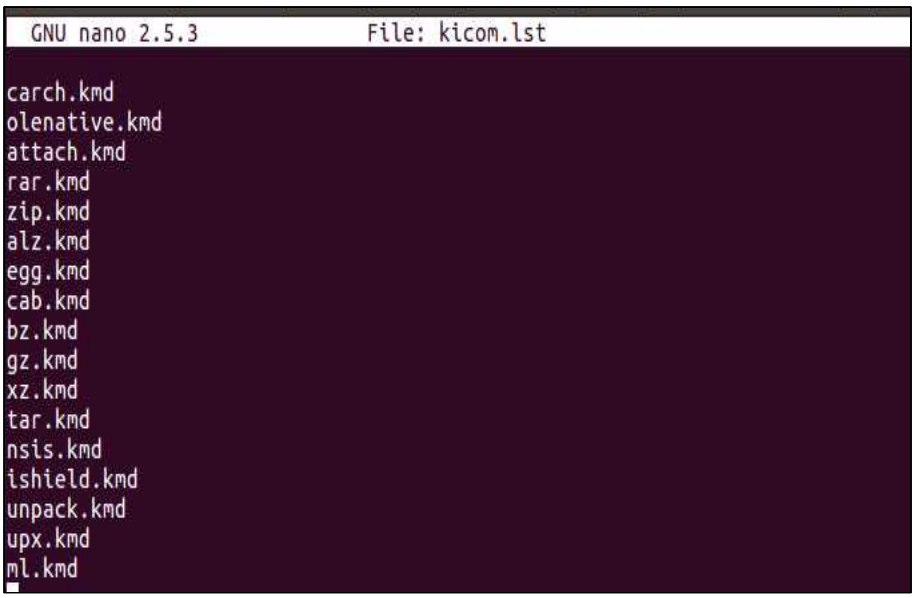


그림 69 kicom 플러그인 리스트에 ml.py -> ml.kmd(암호화된 파일 확장자) 추가

```
class KavMain:
        # ---------------------------------------------
        # init(self, model_path)
        # 플러그인 엔진을 초기화한다.
        # 입력값 : model_path - 모델 파일 경로
        # 리턴값 : 0 - 성공, 0 이외의 값 - 실패
        # ---------------------------------------------
        def init(self, plugins_path, verbose=False):
                # 악성코드 탐지 모델 등록
                path = plugins_path + "/model.joblib"
                self.ml_model = joblib.load(path)
                self.plugins_path = plugins_path

                return 0
```

그림 70 악성코드 탐지 모델 등록

```
# ------------------------------------------------------------------
# disinfect(self, filename, malware_id)
# 악성코드를 치료한다.
# 인력값 : filename - 파일 이름
#                 malware_id - 치료할 악성코드 ID
# 리턴값 : 발견한 악성코드 이름
# ------------------------------------------------------------------
def disinfect(self):
        try:
                if malware_id == 0: # 삭제를 통해 치료를 하는 코드
                        os.remove(filename) # 파일 삭제
                        return True # 치료 완료 리턴
        except IOError:
                pass

        return False # 치료 실패 리턴

# 플러그인 엔진이 진단/치료 가능한 악성코드의 리스트를 알려준다.
def viruslist(self):
        vlists = []
        return vlists

# 플러그인 엔진의 주요 정보를 알려준다.
def getinfo(self):
        info = dict()

        info['author'] = 'nababora'
        info['version'] = '1.0'
        info['title'] = 'ML Detection Engine'
        info['kmd_name'] = 'ML'

        return info
```

그림 71 disinfect()

```
# ------------------------------------------------------------------
# scan(self, filehandle, filename)
# 악성코드의 악성 정도를 검사한다.
# 인력값 : filehandle - 파일 핸들
#                filename - 파일 이름
#                 fileformat - 파일 포맷
# 리턴값 : 악성코드 발견 여부, 악성코드 점수
# ------------------------------------------------------------------
def scan(self, filehandle, filename, fileformat, filename_ex):
        path = self.plugins_path
        try:
                if 'ff_pe' in fileformat:
                        ret = self.__scan_ml(filehandle, filename, fileformat, path)
                        return ret
        except IOError:
                pass

        return False, '', -1, kernel.NOT_FOUND
```

그림 72 scan()

```
def __scan_ml(self, filehandle, filename, fileformat, path):
        clf = self.ml_model
        ft = PE_features(filename, path)
        data, magic = ft.extract_all()

        if magic != 267 or len(data) != 69:
                return False, '', -1, kernel.NOT_FOUND

        pattern_path = path + "/patterns.csv"
        f = open(pattern_path, 'r')
        ft = PE_features(filename, path)
        data, magic = ft.extract_all()

        if magic != 267 or len(data) != 69:
                return False, '', -1, kernel.NOT_FOUND

        pattern_path = path + "/patterns.csv"
        f = open(pattern_path, 'r')
        rd = csv.reader(f)
        for row in rd:
                patterns = row

        packer_type = [0] * len(patterns)

        try:
                idx = patterns.index(data[63])
        except ValueError:
                idx = 10

        packer_type[idx] = 1
        del data[63]
        data = data + packer_type
        data = np.asarray(data).reshape((1, -1))
        rns = clf.predict_proba(data)[0][1]
        pat = "ML Confidence - " + str(rns)

        if rns > 0.8:    # 악성일 확률이 80% 이상일 경우에만 악성코드로 탐지
                print pat
                return True, pat, 0, kernel.INFECTED
        else:
                return False, '', -1, kernel.NOT_FOUND
```

◎ 이로써 키콤 백신에 지금까지 공부한 모델을 적용했다. 이제 최종 결과로 넘어가 우리가 만든 인공지능 백신으로 악성 파일과 정상 파일을 분석해본다.

3. 최종 결과

3.1 분석

▶ 기존에 있던 샘플로 악성코드 분석을 실행해본 결과 ml_scan(만든 모듈)으로 탐지가 되고 악성코드일 확률을 보여준다.

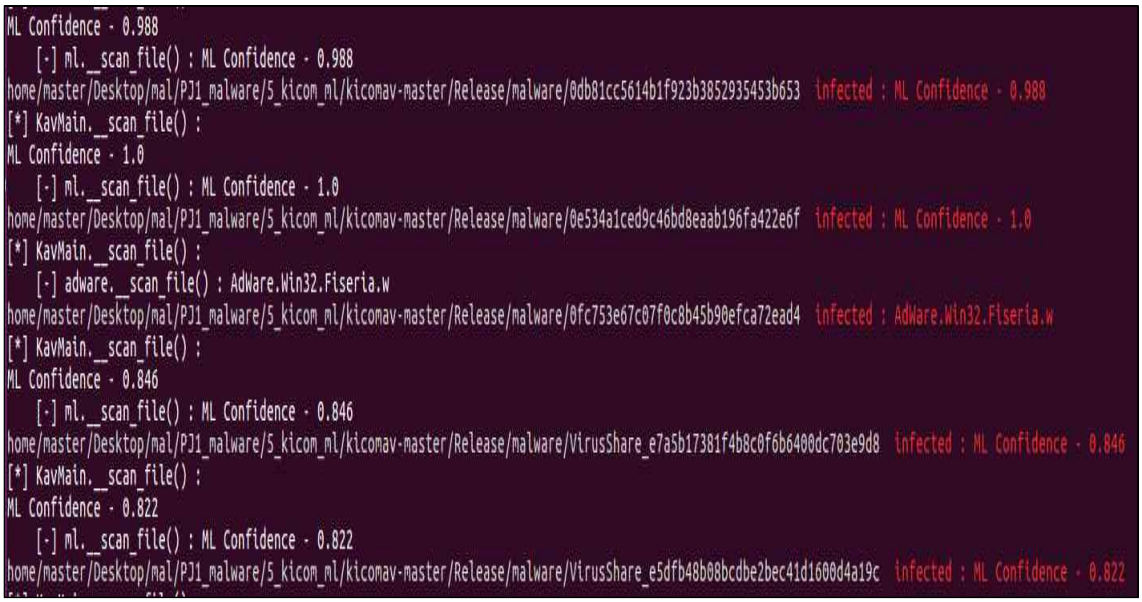


그림 74 악성 파일 분석 중

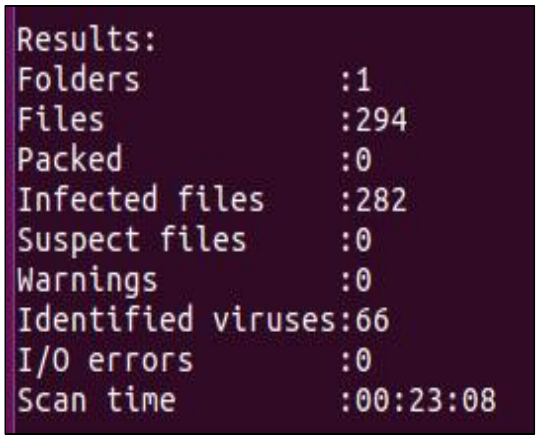


그림 75 악성 파일 검사 결과

- ml__scan()을 통해서 악성코드일 확률과 함께 출력된다.
- ml__scan() 실행되기 이전에 키콤 플러그인에서 탐지된 파일들은 탐지한 플러그인 이름이 출력된다.
- 기존 파일 294개 중에서 282개를 감염이 됐다고 판단했다.
- 모델을 만들 때 쓴 샘플 파일의 경우 탐지율이 95% 정도 된다.

▶ 다음으로는 정상 파일을 키콤 백신에서 분석해보았다.

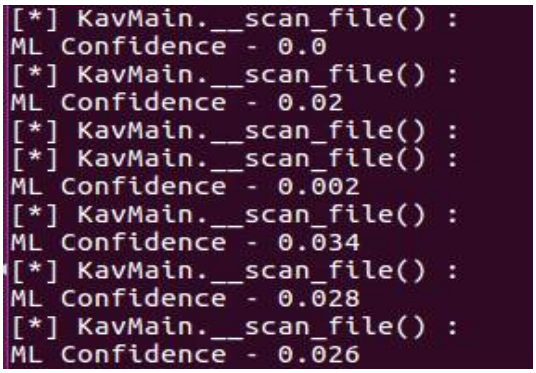


그림 76 정상 파일 분석 중

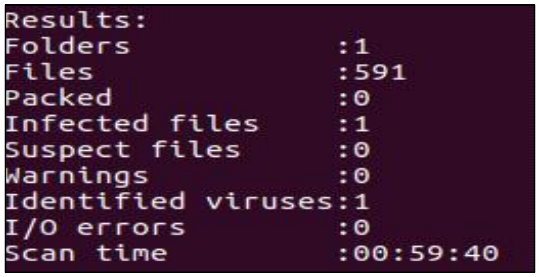


그림 77 정상 파일 분석 결과

정상 파일 591개 검사 결과 한 개의 샘플이 악성 파일로 분석되었다.

마지막으로 https://bazaar.abuse.ch/browse 사이트에서 다운로드 받아 따로 모은 악성 파일을 분석해보았다.

그림 78 모은 악성 파일 분석 중

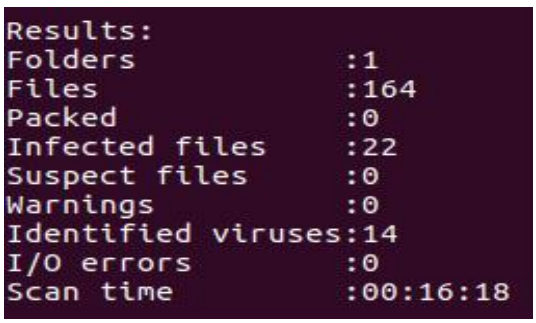


그림 79 모은 악성 파일 분석 결과

◎ 총 164개의 파일 중 22개만 악성코드로 판별했다.

3.2 결과

그림 75에서 Ml_Confidence 수치를 보면 80% 이상으로 판별하지 못하는 악성 파일이 대부분인 것을 볼 수 있다. 우리가 모델을 구축할 때에 썼던 샘플 데이터가 너무 부족했기 때문에 좋은 모델을 만들지 못한 것으로 판단된다. 보고서에 싣지는 않았지만, 참고문헌인 “인공지능 보안을 배우다”에 PE header 특징 중 중요도 특징 상위 20개를 뽑는 실습이 있었다. APS에서 뽑아낸 상위 특징과 달랐는데, 악성코드에 대한 지식이 부족했기 때문에 좋은 특징을 선별해 낼 수 없었다고 생각한다.

하지만 이번 프로젝트로 인해 APS는 악성코드 탐지 모델을 구축하고 백신에 연결하는 방법을 알게 되었다. 실제로 키콤 백신에 반영이 되는 탐지 모델을 구축하려면 더 긴 시간을 들여 연구하고 성능을 높여야 하겠지만, 실제 키콤 오픈소스에 적용할 플러그인을 개발할 수 있다는 자신감을 얻게 되었다.

그러므로 우리는 다음 프로젝트로 플러그인 개발을 진행해보고자 한다.
Kucis 프로젝트가 끝나고 나서도 악성 파일에 관한 공부를 더 깊게 하여 도메인을 확보하고, 더 나은 특징을 선별할 수 있도록 분류 알고리즘도 더 연구하여, 키콤 오픈소스에 실제로 적용할만한 플러그인을 만들어 보안 시장에 좋은 영향을 줄 수 있는 프로젝트를 진행할 것이다.

4. 참고문헌

1. 인공지능 보안을 배우다.
- 저자 : 서준석 출판사 : 비제이퍼블릭(BJ퍼블릭)

2. 리버싱 입문
- 저자 : 조성문 출판사 : 프리렉

3. 파이썬으로 배우는 Anti-Virus 구조와 원리
- 저자 : 최원혁 출판사 : 비제이퍼블릭(BJ퍼블릭)