

MobSF를 활용한 모바일 애플리케이션 자동 분석 시스템

팀장: 홍영창

팀원 : 구현모, 김동진, 장다솜, 최승희, 조운지

CONTENTS

- 프로젝트 개요
- 진행 프로세스
- 자동화 스크립트 구현
- 프로젝트 결과
- 자체평가 및 느낀점

프로젝트 개요

팀원 구성 및 담당 업무

이름	역할	담당 업무
공통진행		MobSF 환경 구축 및 분석
홍영창	팀장	jadx 코드 분석 및 노선 정리, 보고서 작성
구현모	팀원	분석 및 테스트
김동진	팀원	분석 및 테스트
장다솜	팀원	보고서 작성
조운지	팀원	정적분석, 동적분석 자동화 코딩
최승희	팀원	정적분석 자동화

프로젝트 목표

직접 실습한 과정 노선 정리를 통해 자동화 과정 발판 삼기

API 커스터 마이징을 통한 정적/동적 분석 전 자동화

최대한 시연 영상에 나오는 로그들을 보고 비슷하게 구현

시스템 환경

- windows10 , 11
- python version : 3.11.7
 - 파이썬버전이 여러종류가 있으면 인식을 하지못해서 MobSF실행 불가능
 - 윤지님, 동진님 run.bat 오류로 인한 3.11.5 파이썬 버전 다운그레이드
- android-studio-2023.1.1.28-windows
- genymotion-3.6.0
 - Samsung Galaxy S10, API25 - 7.1, 540X960, 240HDPI
- Win64 openssl v3.2.1 version
- jdk-11.0.20
- wkhtmltox-0.12.6-1.msvc2015-win64
- 분석 프로그램
 - Mobile-Security-Framework-MobSF-master
 - jadx-gui-1.4.7

진행 프로세스

WBS

[illegible]

단계 별 프로세스 – 1주차

일정	실행 내용
1주차	MobSF 환경 구축 및 프로젝트 파악
[Day 01]	프로젝트 계획 수립
24.02.19	샌드박스 구축에 대한 정의 및 프로젝트 이해하기
[Day 02-03]	MobSF 실행 환경 구축 및 실행
24.02.20~21	<u>모바일 샌드박스</u> 참고해서 MobSF 설치진행
	안드로이드 스튜디오 연동 오류로 지니모션으로 연동 성공
	가상환경구축 오류 발생으로 포기 → 로컬 호스트 환경에서 모두 진행 (승희님, 현모님 제외)
	MobSF 정적, 동적분석 환경 구축 및 실행
[Day 04-07]	MobSF 구조 이해 및 jadx를 통한 sample.apk 정적 분석 실습
24.02.22~25	MobSF 정적, 동적분석 동작 과정 파악
	jadx 도구를 이용해 sample.apk 정적 분석 실습
	dex파일 정적분석을 통한 Nested APK 호출관계 파악

단계 별 프로세스 - 2주차

2주차	MobSF 자동화를 위한 악성코드 앱 내부 코드 분석 및 수동 실습
[Day 08-11]	sample.apk 디버깅&리패키징
24.02.26~29	안드로이드 앱 구조 파악 공부
	jadx로 암호화된 kill 파일 안열림 / 복호화 필요 > CyberChef
	apktool을 이용하여 sample.apk의 kill-classes.dex를 복호화하고 리패키징
	nested apk(pgsHZz.apk)을 복호화하고 리패키징
	keytool, jarsigner도구를 이용한 keystore 파일 생성, 및 서명
	서명한 sample.apk 에뮬레이터에 설치 성공
[Day 10-13]	frida 우회
24.02.28~03.02	frida server/client 설치 후 에뮬레이터 연동
	anti-vm을 우회하기 위한 코드를 파악 및 frida 스크립트를 작성
	frida 우회 스크립트 주입한 후 앱 동작 과정 파악 (권한설정 등
	MobSF 자동화를 위한 소스코드 구조 파악 및 api 구조 공부

단계 별 프로세스 – 3주차

3주차	MobSF API를 이용한 샌드박스 분석 시스템 자동화
[Day 14-16]	MobSF API 정적 분석 자동화
24.03.04~03.06	MobSF API를 이용한 정적 분석 자동화
	정적 분석 자동화 분석 실행 성공 및 report 다운 성공
[Day 15-17]	MobSF API 동적 분석 자동화
24.03.05~03.07	frida 우회 실행 실패 > 성공 (kill-dex 원본 파일 살리기)
	액티비티별 동적분석 구현 성공
	권한 설정 자동화 성공
	tls /ssl 테스트
[Day 15-19]	자동화 프로그램 테스트
24.03.07	오류 수정 및 업데이트 (내부 Nested APK에 대한 보고서 추출 기능 추가)
	테스트 완료 및 시연 영상 촬영 완료
[Day 19-21]	프로젝트 결과 점검 및 보고서 작성 마무리
24.03.08	개인별 보고서 작성 마무리 및 개인별 느낀점 작성
	영창님 - 보고서 점검 및 발표 준비
24.03.09	자동화 API 트러블 슈팅 및 마무리
24.03.10	운지님 - + virustotalAPI 악성코드 패밀리 식별
	승희님 - mobsfAPI , get_apps, mobsfy 추가/ 복호화 과정 수정 / ssl/tls 추가
[Day 22]	제출 및 발표
24.03.11	영창님 - 제출 및 발표

DEX파일 복호화 및 리패키징

2.4 dex파일 복호화 및 리패키징

MobSF의 정적분석 기능을 이용해서 sample.apk를 분석하려 하는데 apk안에 암호화된 dex 파일들이 있어 암호화된 dex파일들(kill-classes.dex)이 정적분석에 안나온다. 이를 나오게끔 해결해야 한다.

sample.apk 리패키징하기

이름	수정된 날짜	유형	크기
assets	2024-02-26 오후 5:21	파일 폴더	
classes.dex.cache	2024-02-26 오후 5:11	파일 폴더	
error_prone	2024-02-21 오후 4:30	파일 폴더	
jsr305_annotations	2024-02-21 오후 4:30	파일 폴더	
kill-classes.dex.cache	2024-02-26 오후 5:11	파일 폴더	
lib	2024-02-21 오후 4:30	파일 폴더	
META-INF	2024-02-21 오후 4:30	파일 폴더	
res	2024-02-21 오후 4:30	파일 폴더	
third_party	2024-02-21 오후 4:30	파일 폴더	
AndroidManifest.xml	2023-10-16 오후 8:25	Microsoft Edge H...	13KB
classes.dex	2023-10-16 오후 8:25	DEX 파일	13KB
kill-classes.dex	2023-10-16 오후 8:25	DEX 파일	2,550KB
resources.arsc	2023-10-16 오후 8:25	ARSC 파일	266KB

sample.apk 파일 안에 1. classes.dex 정상파일 2. kill-classes.dex 암호화된 파일 이렇게 2가지가 있다.

이 상태에서 MobSF에다가 파일 업로드를 하게 되면

DEX	DETECTIONS
assets/pgsHZz.apk/classes.dex	<div>Findings: Compiler</div> <div>Showing 1 to 1 of 1 entries</div>
classes.dex	<div>Findings: Anti-VM Code</div>

sample.apk (unpack1) 디컴파일 후 리패키징

이제 안의 pgsHZz.apk 파일 안의 dex파일들은 다 복호화 된것들이니 한꺼번에 리패키징을 해 준다.(원래 한 앱에 Nested된 어플들이므로!)

이제 unpack에 다 모였으므로 리패키징 할 것이다.

apktool b [디컴파일된 폴더] -o [최종APK파일이름]

apktool b unpack -o sample2.apk

위 명령어로 리패키징 해준다.

이름	수정된 날짜	유형
unpack	2024-03-06 오후 7:02	파일 폴더
unpack2	2024-03-06 오후 6:57	파일 폴더
sample2.apk	2024-03-06 오후 7:02	APK 파일

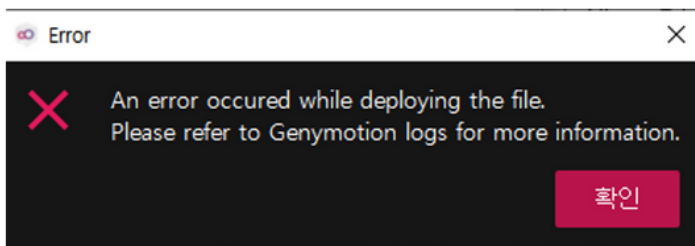
리패키징 성공! 이제 MobSF에 올려서 정적분석 해보자

DEX	DETECTIONS
assets/pgsHZz.apk! classes.dex	<div>Findings: Compiler</div> <div>Details: dx</div>
assets/pgsHZz.apk! classes2.dex	<div>Findings: Anti-VM Code</div> <div>Details: Build.FINGERPRINT check, Build.MODEL check, Build.MANUFACTURER check, Build.PRODUCT check, Build.HARDWARE check, Build.BOARD check, possible Build.SERIAL check, Build.TAGS check, SIM operator check, emulator file check, Compiler</div>
assets/pgsHZz.apk! classes3.dex	<div>Findings: Anti-VM Code</div> <div>Details: Build.FINGERPRINT check, Build.MODEL check, Build.MANUFACTURER check, Build.PRODUCT check, possible VM check, r8 without marker (suspicious), Compiler</div>
classes.dex	<div>Findings: Compiler</div> <div>Details: dx</div>
classes2.dex	<div>Findings: Anti-VM Code</div> <div>Details: Build.MODEL check, Build.TAGS</div>

APK파일 서명 후 에뮬레이터 앱 설치

▼ 2.5 리패키징한 apk파일 서명 후 에뮬레이터 설치

💡 apk파일을 리패키징하는것 뿐 아니라 이제 모바일 단말기에 설치해서 사용해보기 위해서는 APK Signing 작업도 해줘야 한다.



리패키징한 sample2.apk를 지니모션에 옮기려니까 위와같은 에러가 뜬다. 재서명 안해서 뜨는 오류라고 한다.

그렇다 어플은 서명을 해야 무결성이 보장되고 신뢰할 수 있는 어플이어야 다운받을 수 있다.

- 앱을 서명하는 이유
 1. **앱의 신뢰 확인:** 서명은 앱이 개발된 개발자나 개발팀을 확인하는 데 사용됩니다. 앱이 공식적으로 출시된 것이 확인되면 사용자들은 앱을 더 신뢰하게 됩니다.
 2. **앱의 무결성 보장:** 서명을 통해 앱이 변경되지 않았음을 검증할 수 있습니다. 즉, 앱의 내용이 개발자가 제공한 것과 동일한지 확인할 수 있습니다.
 3. **앱의 보안 강화:** 서명된 앱은 블랙해킹, 데이터 변조 또는 악성 코드 삽입과 같은 보안 위협으로부터 더욱 안전합니다.
 4. **플랫폼 또는 스토어 요구 사항:** 특정 플랫폼이나 앱 스토어는 서명된 앱만을 허용할 수 있습니다. 서명된 앱은 이러한 요구 사항을 충족시키기 위해 필요합니다.

APK Signing 해주기

APK Signing 하기 위해서 사용하는 keystore 가 없다면 미리 만들어줘야 한다. 먼저 keystore 서명을 만들기 위해서는 아래 명령어를 이용한다.

```
keytool -genkey -v -keystore [keystore이름] -alias [alias이름] -keyalg [키알고리즘] -
```

APK 파일에 서명하기(self-sign)

이번에는 jarsigner 라는 도구를 이용해서 APK 파일에 서명을 한다.

아래 명령어를 입력하면 된다.

```
jarsigner -verbose -sigalg [서명알고리즘] -digestalg [digest알고리즘] -keystore [keystore파일] [서명할APK] [alias명]
```

```
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore honghong.keystore sample2.apk honghong
```

```
...중략...
signing: res/layout/select_dialog_item_material.xml
signing: res/layout/select_dialog_multichoice_material.xml
signing: res/layout/select_dialog_singlechoice_material.xml
signing: res/layout/support_simple_spinner_dropdown_item.xml
signing: res/layout-v26/abc_screen_toolbar.xml
signing: res/layout-watch/abc_alert_dialog_button_bar_material.xml
signing: res/layout-watch/abc_alert_dialog_title_material.xml
signing: res/menu/activity_main_drawer.xml
signing: res/menu/main.xml
signing: res/xml/accessibilityservice.xml
signing: res/xml/file_paths.xml
signing: resources.arsc
signing: assets/pgsHZz.apk
signing: error_prone/Annotations.gwt.xml
signing: jsr305_annotations/jsr305_annotations.gwt.xml
signing: third_party/java_src/error_prone/project/annotations/Annotations.gwt.xml
signing: third_party/java_src/error_prone/project/annotations/Google_interna1.gwt.xml

>>> Signer
X.509, CN=honghong, OU=honghong, O=honghong, L=Seoul, ST=Seoul, C=KO
[
Signature algorithm: SHA256withRSA, 2048-bit key
[trusted certificate]
```

Jadx분석 및 Frida 우회 Script 만들기

2.6 jadx분석 및 Frida 우회Script 만들기

MobSF 정적 보고서 이용해 Anti-VM code 찾기

우선 Frida로 우회하는 이유는 sample.apk를 실행하면 VM(지니모션같은 에뮬레이터 등)을 탐지하는 코드가 있기 때문에 가상환경 안에서 어플 실행이 안된다. 그러므로 Frida를 이용해 script를 인젝션하여 우회하고 어플 실행에 성공시키기 위함 목적이다.

re_sample_report.pdf 222.9KB

MobSF에서의 sample_repacked.apk 정적 분석 보고서

APKID ANALYSIS

FILE	DETAILS	
classes.dex	FINDINGS	DETAILS
	Compiler	dx
classes2.dex	FINDINGS	DETAILS
	Anti-VM Code	Build.MODEL check Build.TAGS check
	Compiler	r8

sample.apk 인 원본파일을 분석하면 Anti-VM 코드가 안나오지만 복호화후 리패키징 과정을 거친

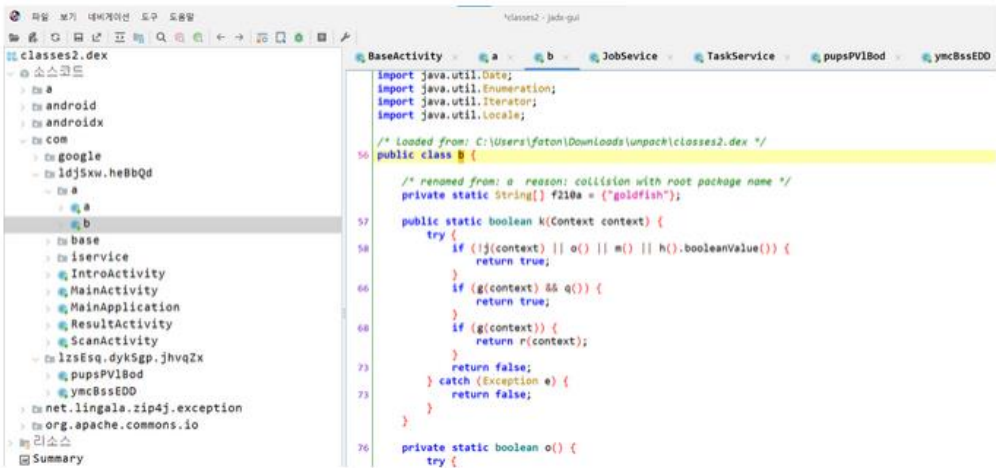
sample_repacked.apk 에서는 classes2.dex 에서 Anti-VM 코드가 분석보고서를 통해 나온다.

- Build.MODEL check: 애플리케이션이 실행 중인 디바이스의 모델명을 확인하는 코드가 포함되어 있음을 나타냅니다. 가상 머신을 사용할 경우 특정 모델명이 설정되지 않거나 예상치 못한 값을 가질 수 있습니다.

jadx도구를 통한 classes2.dex 파일 분석 과정

내 PC > 다운로드 > unpack >		unpack 검색
이름	수정된 날짜	유형
assets	2024-02-28 오후 4:15	파일 폴더
build	2024-02-28 오후 4:16	파일 폴더
classes2.dex.cache	2024-03-04 오후 12:48	파일 폴더
dist	2024-02-28 오후 4:33	파일 폴더
lib	2024-02-28 오후 3:50	파일 폴더
original	2024-02-28 오후 3:50	파일 폴더
res	2024-02-28 오후 3:50	파일 폴더
unknown	2024-02-28 오후 3:50	파일 폴더
AndroidManifest.xml	2024-02-28 오후 3:50	Microsoft Edge H... 8KB
apktool.yml	2024-02-28 오후 3:50	Yaml 원본 파일 1KB
classes.dex	2024-02-28 오후 3:50	DEX 파일 13KB
classes2.dex	2024-02-28 오후 3:58	DEX 파일 2,550KB

VM을 탐지하는 코드 분석을 위해 sample.apk에서 kill-classes.dex 파일을 복호화한 classes2.dex 파일을 jadx로 분석한다. (classes.dex 파일에는 VM 탐지 코드가 없음을 확인함)



위의 b 클래스에서 vm을 우회하는 함수들이 들어있다. 이 코드들을 보면서 Frida우회script 코드를 짜야한다.

Jadx분석 및 Frida 우회 Script 만들기 - 코드 난독화

2.6 jadx분석 및 Frida 우회Script 만들기

jadx도구를 통한 classes2.dex 파일 분석 과정

MobSF 정적 보고서 이용해 Anti-

우선 Frida로 우회하는 이유는 sample를 탐지하는 코드가 있기 때문에 가상 머신으로 실행하는 것보다 Frida를 이용해 script를 인젝션하는 것이 목적이다.

re_sample_report.pdf 222.9KB

MobSF에서의 sample_repacked.apk 정적 분석 결과

APKID ANALYSIS

FILE
classes.dex
classes2.dex

sample.apk 인 원본파일을 분석하면 Anti-Frida 관련 코드가 발견된다.

sample_repacked.apk에서는 classes2.dex

코드 난독화

코드 난독화는 총 3가지가 있다.

1. 기본적으로 변수 이름이나 함수 이름을 난독화 ← sample.apk는 이것만 해당!
2. 문자열을 난독화 시키는 방법
 - a. 예제를 보면 문자열은 읽을 수 있기 때문에 문자열 난독화는 걸려있지 않다.
3. 코드를 가상화 시키는 방법
 - a. 코드 가상화가 걸려있으면 소스코드를 볼 수 없기 때문에 코드 가상화도 걸려있지 않다.

google	
ldjSxw.heBbQd	
a	
a	
b	
base	
BaseActivity	
a	
b	
c	
d	
mLoadingDialog	

```
import java.util.Enumeration;
import java.util.Iterator;
import java.util.Locale;

/* Loaded from: C:\Users\faton\Downloads\unpack\classes2.dex */
56 public class b {
    /* renamed from: a reason: collision with root package name */
    private static String[] f210a = {"goldfish"};

    57 public static boolean k(Context context) {
        58     try {
            if (!j(context) || o() || m() || h().booleanValue()) {
                return true;
            }
        }
    }
}
```

a, b 같이 읽기 또는 식별하기 어려운 변수명들이 있다. 이 사람들이 실제로 개발할때 이런 변수를 썼을까? → 아니다. 클래스 이름, 변수 이름, 함수 이름 정도는 난독화 시켰구나 라는 것을 알 수 있다. 이것을 **proguard**라고 부른다. proguard 실행 옵션에 따라서 좀 다르긴한데 이런식으로 코드 난독화를 시킬 수 있다.

경색	유형	크기
4:15	파일 폴더	
4:16	파일 폴더	
12:48	파일 폴더	
4:33	파일 폴더	
3:50	파일 폴더	
3:50	파일 폴더	
3:50	파일 폴더	
3:50	파일 폴더	
3:50	Microsoft Edge H...	8KB
3:50	Yaml 원본 파일	1KB
3:50	DEX 파일	13KB
3:58	DEX 파일	2,550KB

classes.dex 파일을 복호화한 classes2.dex (코드가 없음 확인함)

```
JobService TaskService pupsPV1Bod ymcBssEDD

Downloads\unpack\classes2.dex */
/* collision with root package name */
Ba = {"goldfish"};
text context() {
o() || m() || h().booleanValue() {
i() {
t);
```

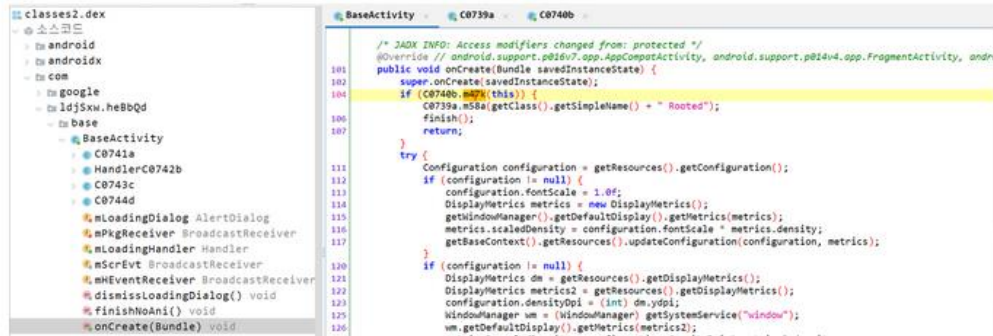
- Build.MODEL check: 애플리케이션이 실행되고 있음을 나타냅니다. 가상 머신을 사용할 경우 특정 모델명이 설정되지 않거나 예상치 못한 값을 가질 수 있습니다.

위의 b 클래스에서 vm을 우회하는 함수들이 들어있다. 이 코드들을 보면서 Frida우회script 코드를 짜야한다.

Jadx분석 및 Frida 우회 Script 만들기

VM을 우회하는 7개 Detection 함수 확인

- BaseActivity class - onCreate method
 - BaseActivity: 모든 Activity의 부모 클래스
 - BaseActivity를 시작하면서 Detection 탐지 후, 탐지된 사항이 존재할 경우 루팅 상태로 인식하고 프로그램 종료



com.ldjSxw.heBbQd.p017a.C0740b class - m47k method

1. Language

```
172  /* renamed from: j */
173  public static boolean m48j(Context context) {
174      locale locale = context.getResources().getConfiguration().locale;
175      String language = locale.getLanguage();
176      return language.contains("ko") || language.contains("KO");
177  }
178
```

m48j method

- 시스템 언어가 한글이 아닐 경우 탐지 (True 반환)

2. test-keys

```
57  /* renamed from: o */
58  private static boolean m43o() {
59      try {
```

5. ADB

```
112  /* renamed from: g */
113  private static boolean m51g(Context context) {
114      try {
115          if (Settings.Secure.getInt(context.getContentResolver(), "adb_enabled", 0) <= 0) {
116              return false;
117          }
118          return true;
119      } catch (Exception e) {
120          return false;
121      }
122  }
```

m51g method - 디바이스의 ADB(Android Debug Bridge)가 활성화되어 있는지 확인

- adb 활성화 여부 확인, 활성화 시 탐지 (True 반환)
- 안드로이드 시스템 설정에서 adb_enabled 값을 가져와 1이면 ADB가 활성화, 0이면 비활성화 ADB가 비활성화 되었다면 false를 반환

6. VPN

```
150  /* renamed from: q */
151  public static boolean m41q() {
152      try {
153          Enumeration niList = NetworkInterface.getNetworkInterfaces();
154          if (niList != null) {
155              Iterator it = Collections.list(niList).iterator();
156              while (it.hasNext()) {
157                  Object intf = it.next();
158                  NetworkInterface net2 = (NetworkInterface) intf;
159                  if (net2.isUp() && net2.getInterfaceAddresses().size() != 0 && ("tun0".equals(net2.getName()) || "ppp0".equals(net2.getName()))) {
160                      return true;
161                  }
162              }
163              return false;
164          }
165          return false;
166      } catch (Throwable e) {
167          e.printStackTrace();
168          return false;
169      }
170  }
```

m41q method - 디바이스에 VPN(Virtual Private Network) 연결이 활성화되어 있는지 확인

- tun0 또는 ppp0 이름의 네트워크 인터페이스가 사용중일 경우 탐지 (True 반환)

7. Proxy

```
124  /* renamed from: r */
125  private static boolean m40r(Context context) {
126      String proxyAddress;
127      int proxyPort;
128      try {
129          boolean is_ics_or_later = Build.VERSION.SDK_INT >= 14;
130          if (is_ics_or_later) {
```


Jadx분석 및 Frida 우회 Script 만들기

VM을 우회하는 7개 Detection함수 확인

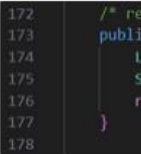
- BaseActivity class - onCreate method
 - BaseActivity: 모든 Activity의 부모 클래스
 - BaseActivity 인식



분석한 코드를 바탕으로 Frida 우회 코드 작성

```
Java.perform(function () {
    var TargetClass = Java.use('com.ldjSxw.heBbQd.a.b');
    TargetClass.k.implementation = function () {
        send("[+] Intercepted com.ldjSxw.heBbQd.a.b.k()");
        return false;
    };
});
```

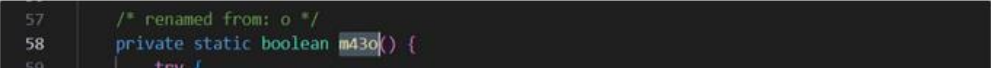
1. Language



m48j method

- 시스템 언어가 한글이 아닐 경우 탐지 (True 반환)

2. test-keys

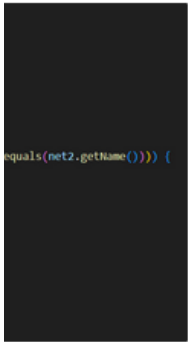


5. ADB



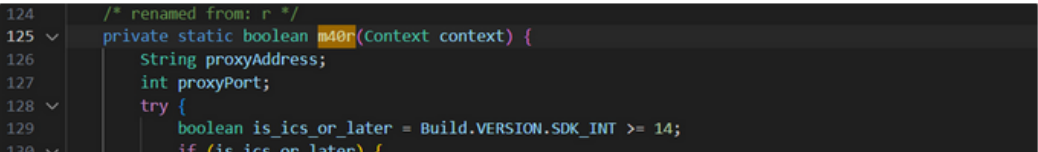
확인

0이면 비활성



확인
(반환)

7. Proxy



Frida코드 주입을 통한 우회하기 및 앱 실행

▼ 2.7 Frida코드 주입을 통한 우회하기 및 어플 실행

💡 Frida도구를 통해 우회를 하지 않으면 **My Scan APP** 어플이 실행이 안되고 꺼진다. 그 이유는 **My Scan APP** 안에 VM을 탐지하는 코드들이 들어있기 때문이다. (7개)
때문에 **My Scan APP** 앱이 실행이 되어야 동적 분석이 진행 가능하므로 우회를 위해 Frida 도구를 이용해서 JavaScript 코드를 삽입해서 우회한다.

Frida 설치 과정





frida Client 설치

```
$ pip3 install --upgrade pip
$ pip3 install frida-tools
```

Frida Server 설치 및 실행

1. Frida Server 최신 release 다운로드: <https://github.com/frida/frida/releases>

- 디바이스 운영체제에 맞는 버전으로 다운로드 할 것

 frida-server-16.2.1-android-arm.xz	6.63 MB	2 weeks ago
 frida-server-16.2.1-android-arm64.xz	14.9 MB	2 weeks ago
 frida-server-16.2.1-android-x86.xz	15.1 MB	2 weeks ago
 frida-server-16.2.1-android-x86_64.xz	30.3 MB	2 weeks ago

2. 압축 해제 후 adb 명령어를 이용하여 디바이스로 해당 파일 전송

```
$ adb push frida-server-16.2.1-android-x86 /data/local/tmp/
```

```
frida -U -n "My Scan APP" -l frida_script.js
```

▶ frida_script.js

```
C:\Users\faton\Downloads>frida -U -n "My Scan APP" -l frida_script.js

[-----]
[  ( )  ]  Frida 16.2.1 - A world-class dynamic instrumentation toolkit
[  / \  ]
[-----]

Commands:
  help      -> Displays the help system
  object?   -> Display information about 'object'
  exit/quit -> Exit

. . . . .
  More info at https://frida.re/docs/home/
  . . . . .
  Connected to Galaxy S10 (id=192.168.56.102:5555)
Failed to spawn: unable to find process with name 'My Scan APP'
```

Failed 뜬 이유는 앱을 한번 실행시켜야 한다.

스크립트 주입해서 한번에 자동실행

```
frida -U -l "bypass_script.js" -f "com.ldjSxw.heBbQd"
```

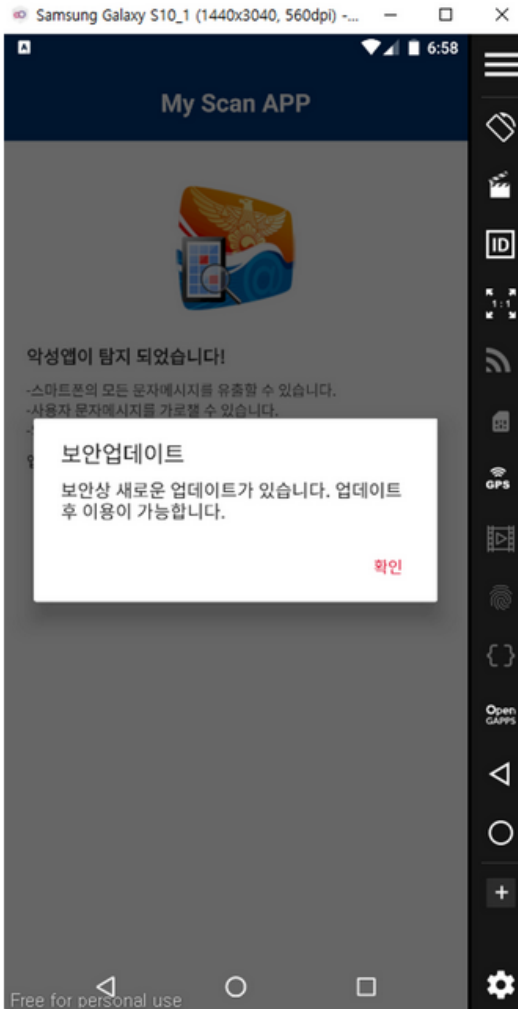
```
frida-ps -Uai
```

```
#Frida가 설치된 Android 디바이스에서 실행 중인 프로세스 목록을 표시
frida-ps -Uai
```

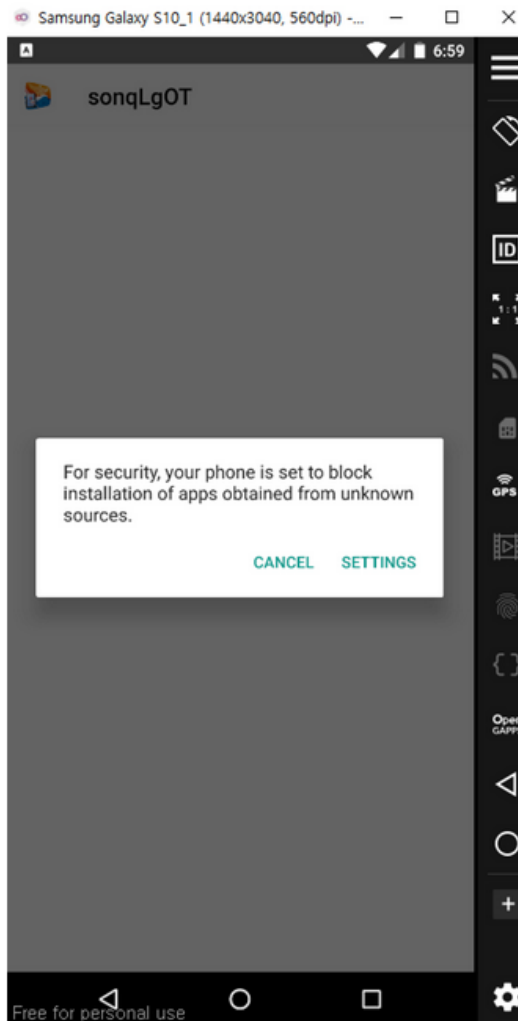
Frida가 설치된 Android 디바이스에서 실행 중인 프로세스 목록을 표시하는 명령어 이다.

```
C:\Users\faton\Downloads>frida-ps -Uai
PID  Name          Identifier
-----
1032  Clock         com.android.deskclock
1771  Gallery      com.android.gallery3d
```

Frida코드 주입을 통한 우회하기 및 앱 실행

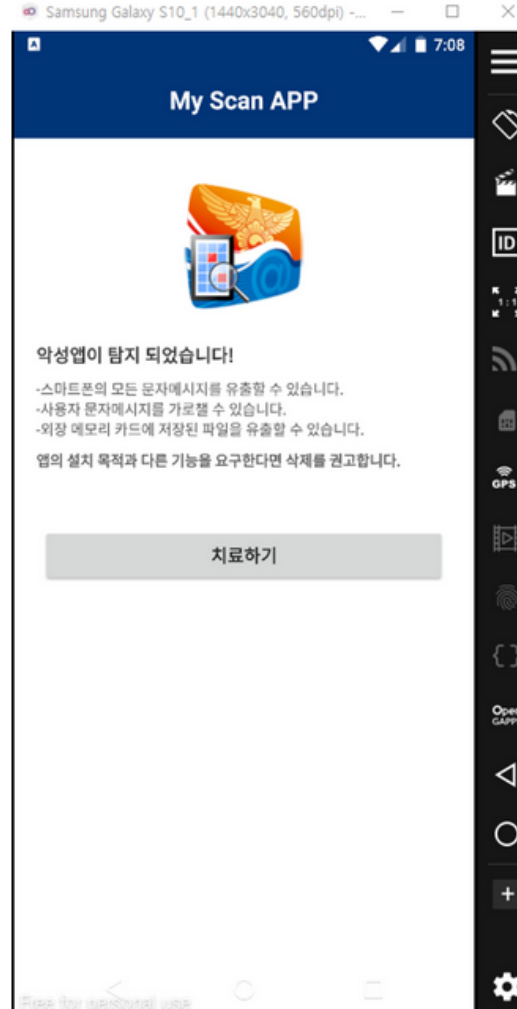


치료를 누르면 다음과 같은 팝업이 뜨면서 새로운 앱을 설치하도록 유도한다.

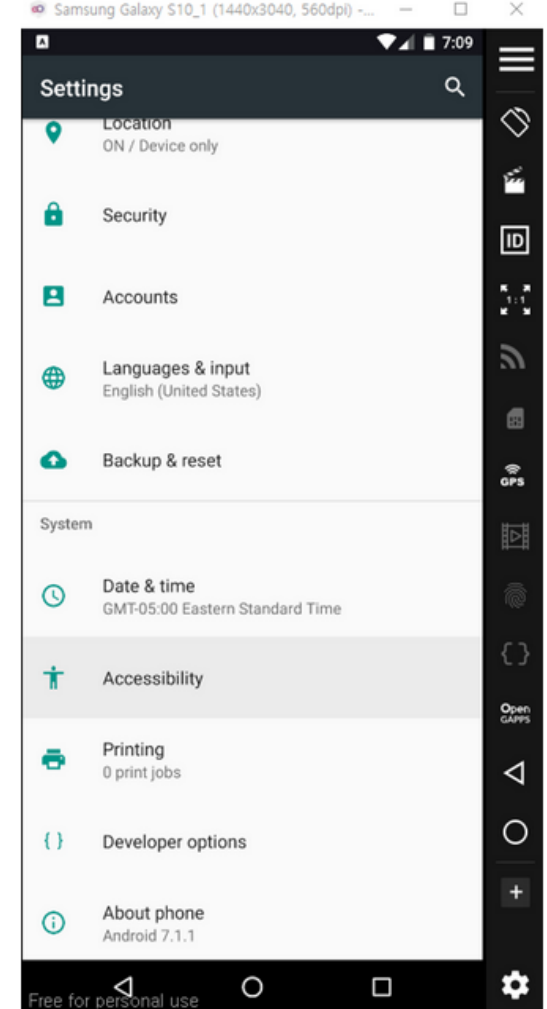


보안을 위해 휴대전화는 출처를 알 수 없는 앱의 설치를 차단하도록 설정되어 있습니다.

라는 문구가 뜬다. 설정을 해줘야한다.

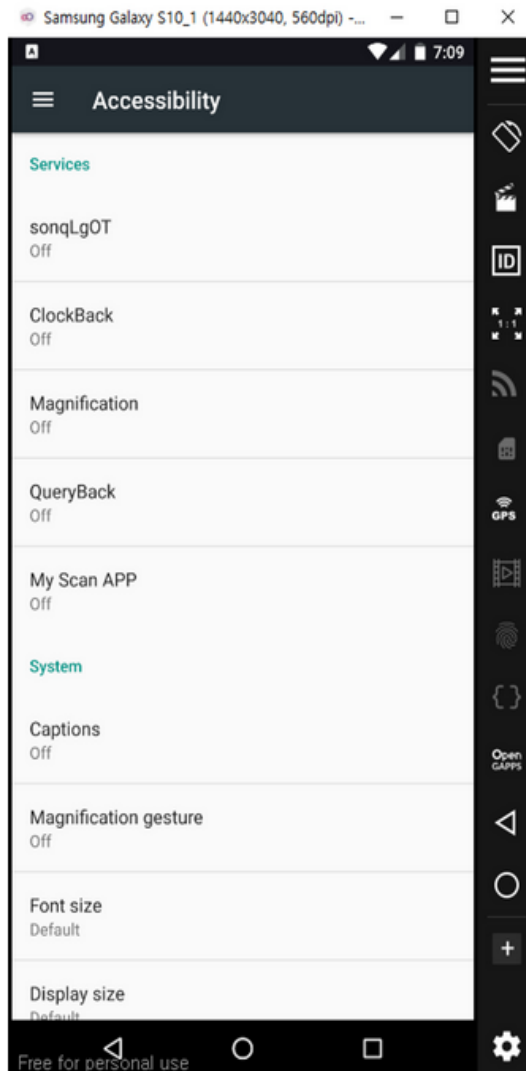


근데 위의 화면에서 안넘어간다.

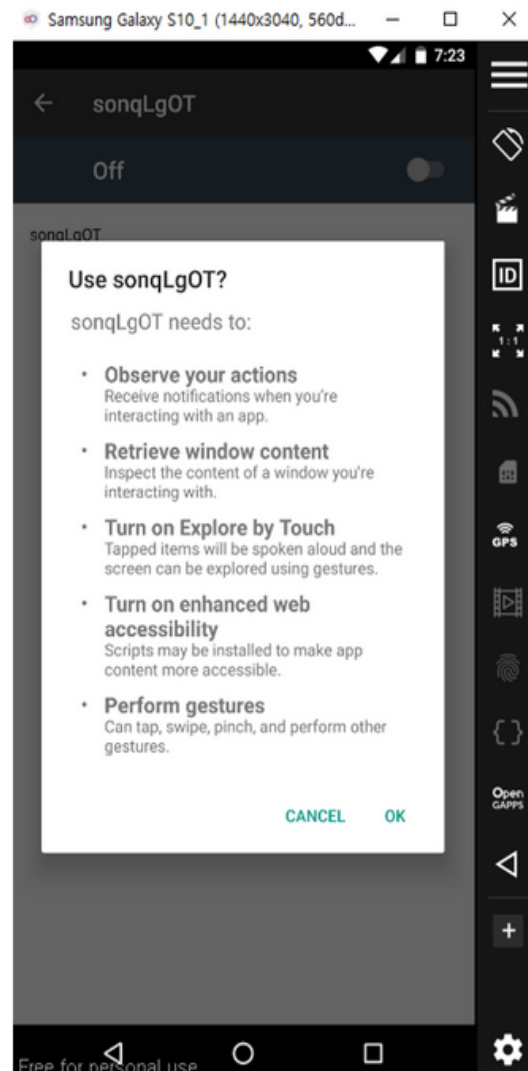


잠시 홈 버튼을 눌러서 **Settings** 설정 → **Accessibility** 에서 권한 설정을 해줘야 한다.

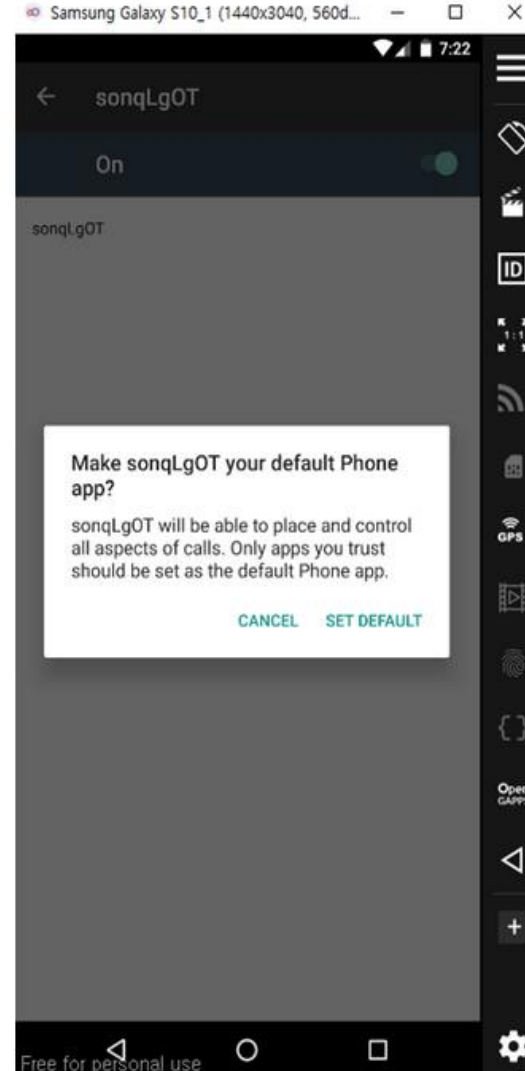
Frida코드 주입을 통한 우회하기 및 앱 실행



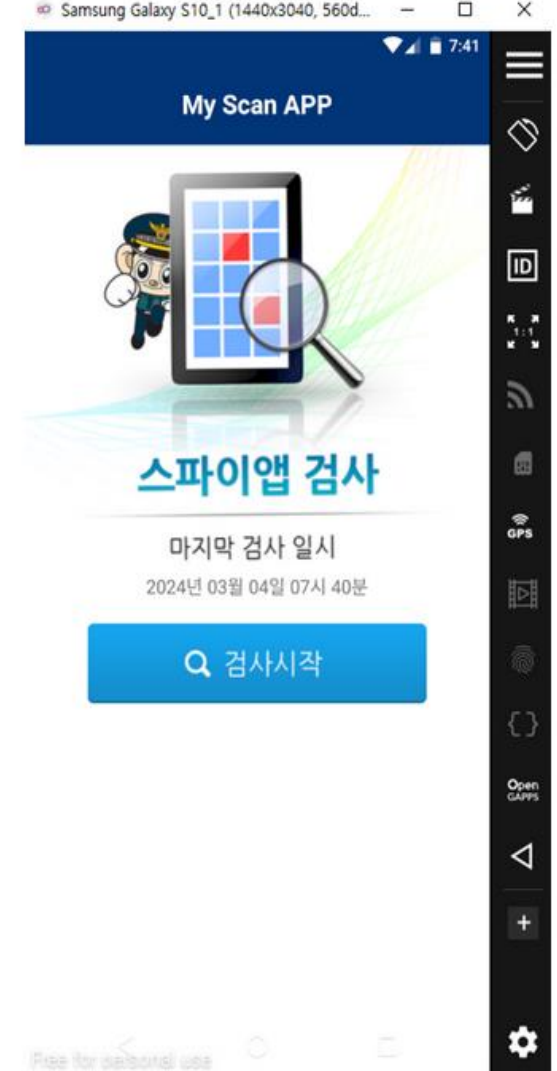
install된 sonqLgOT 클릭



OK 클릭

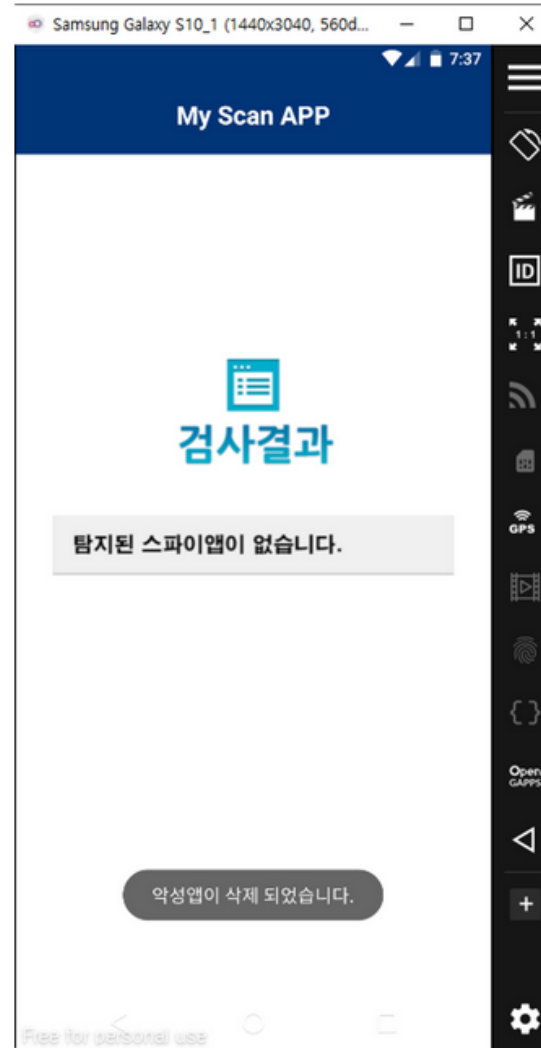


기본 Default 앱으로 설정해준다.



다시 검사시작 버튼 클릭!

Frida코드 주입을 통한 우회하기 및 앱 실행



그러면 악성앱이 삭제되었다고 알림이 뜨면서 검사결과가 나오고 탐지된 스파이 앱이 없다고 나온다.



앱을 다시 실행하고 검사시작을 하면 중간에 치료하기 과정이 없어지고 위와같이 검사결과 화면이 나온다.

자동화 스크립트 구현

스크립트 실행 조건

↑

📁 > 내 PC > 다운로드 > test > sandbox

▼ ↺

sandbox 검색

	이름	수정한 날짜	유형	크기
기	📄 sample.apk	2023-10-22 오후 8:15	APK 파일	34,309KB
화면	📄 mobsfAPI.py	2024-03-10 오후 11:09	Python 원본 파일	19KB
링크	📄 honghong.keystore	2024-03-06 오후 7:37	KEYSTORE 파일	3KB
ortcut	📄 bypass_vm_detection.js	2024-03-04 오전 10:28	JavaScript 파일	3KB
프로젝트				

정적분석 코드

```
def decompile_apk(apk_path, output_dir):
    # APK 파일 디컴파일
    try:
        subprocess.run(['C:\\Windows\\apktool.bat', 'd', '-r', '-s', '-f', apk_path, '-o', output_dir], input=b'\n', check=True)
        print(f"APK 디컴파일 성공: {apk_path}")
        return True
    except subprocess.CalledProcessError as e:
        print(f"APK 디컴파일 실패: {e}")
        return False

def find_nested_apks_and_decompile(decompiled_dir):
    # 중첩된 APK 찾기 및 디컴파일
    for root, dirs, files in os.walk(decompiled_dir):
        for file in files:
            if file.endswith('.apk'):
                nested_apk_path = os.path.join(root, file)
                nested_decompiled_dir = nested_apk_path + '_decompiled'
                if decompile_apk(nested_apk_path, nested_decompiled_dir):
                    process_dex_files(nested_decompiled_dir)

# 복호화 수행
def decrypt_dex_file(encrypted_dex_path, decrypted_dex_path, aes_key):
    try:
        # encrypted dex file open
        with open(encrypted_dex_path, 'rb') as encrypted_file:
            encrypted_data = encrypted_file.read()

        # 복호화과정
        cipher = AES.new(aes_key, AES.MODE_ECB)

        decrypted_data = unpad(cipher.decrypt(encrypted_data), AES.block_size)

        with open(decrypted_dex_path, 'wb') as decrypted_file:
            decrypted_file.write(decrypted_data)

        print(f"Dex decrypted complete: {encrypted_dex_path} -> {decrypted_dex_path}")

        # 복호화한 디컴파일 삭제 여부
        # os.remove(encrypted_dex_path)

    return True
```


정적분석 코드

```
def process_dex_files(decompiled_dir):
    # 'kill'로 시작하는 DEX 파일 처리
    dex_files = glob.glob(os.path.join(decompiled_dir, '*dex'))
    encrypted_dex = []
    for dex_file_path in dex_files:
        with open(dex_file_path, 'rb') as file:
            header = file.read(8)
            if not header.startswith(b'dex\n035'):
                encrypted_dex.append(dex_file_path)

    print(f'encrypted_dex: {encrypted_dex}')
    for encrypted_dex_path in encrypted_dex:
        new_dex_number = len([f for f in dex_files if os.path.basename(f).startswith('classes')]) + 1

        # 복호화 한 dex파일 경로 및 이름 지정
        decrypted_dex_path = os.path.join(decompiled_dir, f'classes{new_dex_number}.dex')
        decrypt_dex_file(encrypted_dex_path, decrypted_dex_path, aes_key)

        dex_files.append(decrypted_dex_path)

def recompile_apk(decompiled_dir, output_apk_path, keystore_path, keystore_password, key_alias):
    # 디컴파일된 APK 재컴파일
    try:
        subprocess.run(['C:\\Windows\\apktool.bat', 'b', '-f', decompiled_dir, '-o', output_apk_path], input=b'\n', check=True)
        print(f"APK 재컴파일 성공: {output_apk_path}")

        # APK 재서명
        sign_cmd = [
            'jarsigner',
            '-verbose',
            '-sigalg', 'SHA256withRSA', # 새로운 키스토어에 맞는 서명 알고리즘
            '-digestalg', 'SHA-256', # 새로운 키스토어에 맞는 다이제스트 알고리즘
            '-keystore', keystore_path,
            '-storepass', keystore_password,
            output_apk_path,
            key_alias
        ]
        subprocess.run(sign_cmd, check=True)
        print(f"APK 서명 성공: {output_apk_path}")
```

정적분석 코드 - API

```
class MobSF_API:
    def __init__(self, server, api_key, file_path):
        self.server = server
        self.api_key = api_key
        self.file_path = file_path

    def upload(self):
        print("파일을 MobSF 서버에 업로드 중...")
        with open(self.file_path, 'rb') as f:
            files = {'file': (os.path.basename(self.file_path), f, 'application/octet-stream')}
            headers = {'Authorization': self.api_key}
            response = requests.post(f'{self.server}/api/v1/upload', files=files, headers=headers)
            result = response.json()
            if response.status_code == 200 and 'hash' in result:
                print("업로드 성공.")
                return result['hash']
            else:
                print("업로드 실패.")
                return None

    def scan(self, file_hash):
        print("업로드된 파일 스캔 시작...")
        data = {'hash': file_hash}
        headers = {'Authorization': self.api_key}
        response = requests.post(f'{self.server}/api/v1/scan', data=data, headers=headers)
        result = response.json()
        if response.status_code == 200:
            print("스캔 성공적으로 시작됨.")
            return result
        else:
            print("스캔 시작 실패.")
            return None
```

정적분석 코드 - API

```
def download_pdf_report(self, file_hash):
    print("스캔된 파일의 PDF 보고서 다운로드 중...")
    data = {'hash': file_hash}
    headers = {'Authorization': self.api_key}
    response = requests.post(f'{self.server}/api/v1/download_pdf', data=data, headers=headers, stream=True)
    if response.status_code == 200:
        pdf_path = f'{os.path.splitext(self.file_path)[0]}_report.pdf'
        with open(pdf_path, 'wb') as f:
            for chunk in response.iter_content(chunk_size=1024):
                if chunk:
                    f.write(chunk)
        print(f"PDF 보고서가 {pdf_path}에 성공적으로 다운로드됨.")
    else:
        print("PDF 보고서 다운로드 실패.")

def get_APPS(self):
    headers = {'Authorization': self.api_key}
    response = requests.get(f'{self.server}/api/v1/dynamic/get_apps', headers=headers)
    result = response.json()

    if response.status_code == 200:
        identifier = result.get('identifier', '')
        print(f"Get APPS: {result}")
        return identifier
    else:
        print("Failed to Get APPS", response.status_code, result)

def mobsfying(self, identifier):
    """API to MobSFY android environmnet"""
    headers = {'Authorization': self.api_key}
    data = {'identifier': identifier}
    response = requests.post(f'{self.server}/api/v1/android/mobsfy', headers=headers, data=data)
    if response.status_code == 200:
        print("mobsfy: ", response.json())
    else:
        print("mobsfy Error: ", response.status_code, response.json())
```

정적 분석 실행 결과

↑

내 PC > 다운로드 > test > sandbox

▼ ↺

sandbox 검색

!기

화면

로드

shortcut

주 프로젝트

rive - Personal

이름

수정한 날짜

유형

크기

sample.apk

mobsfAPI.py

honghong.keystore

bypass_vm_detection.js

re_pgsHZz_report.pdf

re_sample.apk

re_sample_report.pdf

re_pgsHZz.apk

decompiled_apk

2023-10-22 오후 8:15

2024-03-10 오후 11:09

2024-03-06 오후 7:37

2024-03-04 오전 10:28

2024-03-10 오후 10:55

2024-03-10 오후 10:53

2024-03-10 오후 10:57

2024-03-10 오후 10:53

2024-03-10 오후 10:52

APK 파일

Python 원본 파일

KEYSTORE 파일

JavaScript 파일

PDF 파일

APK 파일

PDF 파일

APK 파일

파일 폴더

34,309KB

19KB

3KB

3KB

200KB

68,550KB

224KB

32,945KB

동적 분석 - API

```
def start_dynamic_analysis(self, file_hash):
    """동적 분석 시작"""
    print("동적 분석 시작 중...")
    data = {'hash': file_hash}
    headers = {'Authorization': self.api_key}
    response = requests.post(f'{self.server}/api/v1/dynamic/start_analysis', data=data, headers=headers)
    result = response.json()
    if response.status_code == 200:
        print("동적 분석이 성공적으로 시작됨.")
        return result
    else:
        print("동적 분석 시작 실패.")
        return None

def bypass_anti_vm(self, package_name, bypass_script_path):
    """안티 VM을 우회하기 위해 Frida 스크립트 실행. 스크립트는 지속적으로 실행되어야 함."""
    def run_script():
        print("안티 VM 우회를 위해 Frida 스크립트 실행 중...")
        self.process = subprocess.Popen(['frida', '-U', '-l', bypass_script_path, '-f', package_name],
                                          stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        print(f"{package_name}에 대한 안티 VM 우회 시작됨.")

    self.frida_thread = threading.Thread(target=run_script)
    self.frida_thread.start()

def tls_test(self, file_hash):
    """SSL/TLS TESTER"""
    print("SSL/TLS TEST STARTING...")
    headers = {'Authorization': self.api_key}
    data = {'hash': file_hash}
    try:
        response = requests.post(f'{self.server}/api/v1/android/tls_tests', headers=headers, data=data)
        if response.status_code == 200:
            print("SSL/TLS Success: ", response.json())
        else:
            print("SSL/TLS Error: ", response.status_code, response.json())
    except requests.exceptions.RequestException as e:
        print("HTTP Request Failed: ", e)
    except json.decoder.JSONDecodeError:
        print("JSON decoding failed")
```

동적 분석 - API

```
def start_activity_analysis(self, file_hash, activity_name):
    """앱의 특정 activity에 대한 동적 분석 시작."""
    print(f"{activity_name} 동적 분석 시작 중...")
    data = {'hash': file_hash, 'activity': activity_name}
    headers = {'Authorization': self.api_key}
    response = requests.post(f'{self.server}/api/v1/android/start_activity', data=data, headers=headers)
    if response.status_code == 200:
        print(f"{activity_name} 동적 분석이 성공적으로 시작됨.")
        return response.json()
    else:
        print(f"{activity_name} 동적 분석 시작 실패.")
        return None

def tap_screen(self, x, y):
    """화면의 특정 위치를 탭하는 ADB 명령어 실행"""
    adb_cmd = f'adb shell input tap {x} {y}'
    try:
        subprocess.run(adb_cmd, shell=True, check=True)
        print(f"화면 위치 ({x}, {y}) 탭 성공")
    except subprocess.CalledProcessError as e:
        print(f"화면 위치 ({x}, {y}) 탭 실패: {e}")

def scroll_down(self, start_x, start_y, end_x, end_y):
    cmd = f"adb shell input swipe {start_x} {start_y} {end_x} {end_y}"
    try:
        subprocess.run(cmd.split(), stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        print(f"화면 스크롤 다운 성공: {start_x}, {start_y} -> {end_x}, {end_y}")
    except subprocess.CalledProcessError as e:
        print(f"화면 스크롤 다운 실패: {e}")

def stop_frida_script(self):
    """Frida 스크립트를 종료합니다."""
    if self.process:
        self.process.terminate()
        self.process = None
    print("Frida 스크립트 종료됨.")
```

동적 분석 - API

```
def stop_dynamic_analysis(self, file_hash):
    """동적 분석을 멈추고 필요한 정리 작업을 수행"""
    print("동적 분석을 멈추는 중...")
    data = {'hash': file_hash}
    headers = {'Authorization': self.api_key}
    response = requests.post(f'{self.server}/api/v1/dynamic/stop_analysis', data=data, headers=headers)
    if response.status_code == 200:
        print("동적 분석이 성공적으로 멈춰짐.")
    else:
        print("동적 분석을 멈추는 데 실패했습니다. 응답 코드:", response.status_code)

def download_dynamic_report(self, file_hash):
    """동적 분석 리포트 다운로드"""
    print("동적 분석 리포트 다운로드 중...")
    headers = {'Authorization': self.api_key}
    data = {'hash': file_hash}
    # 동적 분석 리포트 다운로드를 위한 엔드포인트: /api/v1/dynamic/report_json
    response = requests.post(f'{self.server}/api/v1/dynamic/report_json', headers=headers, data=data)
    if response.status_code == 200:
        report_path = f'{os.path.splitext(self.file_path)[0]}_dynamic_report.json'
        with open(report_path, 'wb') as f:
            f.write(response.content)
        print(f"동적 분석 리포트가 {report_path}에 성공적으로 다운로드됨.")
    else:
        print("동적 분석 리포트 다운로드 실패.")
```

동적 분석 - API

```
if dynamic_results:
    mobSF.bypass_anti_vm(package_name, bypass_script_path)
    time.sleep(30)

    mobSF.start_activity_analysis(file_hash, 'com.ldjSxw.heBbQd.IntroActivity')
    # 동적 분석 중 TLS/SSL 보안 테스트 수행
    # mobSF.dynamic_tls_ssl_test(file_hash) #비동기 백그라운드로 해봤으나 애도 앱을 실행시키는데 프리다 우회 스크립트를 붙여야하고, 순서가 중요
    mobSF.start_activity_analysis(file_hash, 'com.ldjSxw.heBbQd.MainActivity')
    mobSF.start_activity_analysis(file_hash, 'com.ldjSxw.heBbQd.ScanActivity')
    mobSF.start_activity_analysis(file_hash, 'com.ldjSxw.heBbQd.ResultActivity')

    mobSF.tap_screen(273, 698)#치료하기
    time.sleep(5)

    mobSF.tap_screen(434, 541)#예
    time.sleep(5)

    mobSF.tap_screen(473, 844)#install
    time.sleep(9)

    mobSF.tap_screen(381, 841)#done
    time.sleep(5)

    mobSF.tap_screen(268, 926)#home
    time.sleep(5)

    mobSF.tap_screen(335, 610)# settings
    time.sleep(15)

    mobSF.scroll_down(329, 881, 358, 87)
    time.sleep(3)
    mobSF.scroll_down(329, 881, 358, 87)
    time.sleep(3)
    mobSF.scroll_down(329, 881, 358, 87)
    time.sleep(3)
    mobSF.scroll_down(329, 881, 358, 87)
    time.sleep(3)
```


동적 분석 실행 결과

↑

내 PC > 다운로드 > test > sandbox

▼ ↺

sandbox 검색

기	이름	수정한 날짜	유형	크기
화면	sample.apk	2023-10-22 오후 8:15	APK 파일	34,309KB
드	mobsfAPI.py	2024-03-10 오후 11:09	Python 원본 파일	19KB
	honghong.keystore	2024-03-06 오후 7:37	KEYSTORE 파일	3KB
ortcut	bypass_vm_detection.js	2024-03-04 오전 10:28	JavaScript 파일	3KB
프로젝트	re_pgsHZz_report.pdf	2024-03-10 오후 10:55	PDF 파일	200KB
ve - Personal	re_sample.apk	2024-03-10 오후 10:53	APK 파일	68,550KB
	re_sample_report.pdf	2024-03-10 오후 10:57	PDF 파일	224KB
	re_pgsHZz.apk	2024-03-10 오후 10:53	APK 파일	32,945KB
	decompiled_apk	2024-03-10 오후 10:52	파일 폴더	
체	re_sample_dynamic_report.json	2024-03-10 오후 11:02	JSON 원본 파일	386KB

악성코드 패밀리 식별 - Virustotal 사이트

27
/ 65

Community Score

27/65 security vendors and no sandboxes flagged this file as malicious

Reanalyze Similar More

82d644a1f3bba120327e7eb6029f6b986c95c35f0c40cd43001f2dbedee2ee6f

Size
33.50 MB

Last Modification Date
21 hours ago

APK

sample.apk

android reflection contains-elf apk obfuscated

DETECTION

DETAILS

RELATIONS

BEHAVIOR

TELEMETRY

COMMUNITY

Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label trojan.fakecalls/bankbot

Threat categories trojan banker dropper

Family labels fakecalls bankbot frgf

Security vendors' analysis

Do you want to automate checks?

AhnLab-V3	⚠ Dropper/Android.Kaishi.1204837	Alibaba	⚠ TrojanBanker:Android/Fakecalls.755b9000
Antiy-AVL	⚠ Trojan/Generic.ASMalWAD.2D	Avast-Mobile	⚠ Android:Evo-gen [Trj]
Avira (no cloud)	⚠ ANDROID/Bankbot.FRGF.Gen	BitDefenderFalx	⚠ Android.Trojan.Banker.ACX
DrWeb	⚠ Android.Spy.1080.origin	ESET-NOD32	⚠ A Variant Of Android/TrojanDropper.Age...
Fortinet	⚠ Android/Agent.DSI!tr	Google	⚠ Detected
Ikarus	⚠ Trojan-Dropper.AndroidOS.Agent	K7GW	⚠ Trojan (005752a51)
Kaspersky	⚠ HEUR:Trojan-Banker.AndroidOS.Fakecal...	Lionic	⚠ Trojan.AndroidOS.Fakecalls.C!c
MAX	⚠ Malware (ai Score=99)	McAfee	⚠ Artemis!6C5BBFB51342
Microsoft	⚠ Trojan:AndroidOS/Multiverze	NANO-Antivirus	⚠ Trojan.Android.Bankbot.kamfge

악성코드 패밀리 식별 – VirustotalAPI 코드

```
#업로딩중 스피너
def spinner():
    spinner_chars = itertools.cycle(['-', '/', '|', '\\'])
    while not upload_finished:
        sys.stdout.write(next(spinner_chars))
        sys.stdout.flush()
        time.sleep(0.1)
        sys.stdout.write('\b')

def get_upload_url(api_key):
    url = 'https://www.virustotal.com/api/v3/files/upload_url'
    headers = {'x-apikey': api_key}
    response = requests.get(url, headers=headers)
    if response.status_code == 200:
        upload_url = response.json().get('data', '')
        return upload_url
    else:
        print(f"Failed to get upload URL: {response.status_code} - {response.text}")
        return None

#tqdm으로 업로드 진행률 표시. 한번에 올려야해서 청크로 나누는게 안되기때문에 시각적효과로만
def upload_file(api_key, file_path, upload_url):
    global upload_finished
    upload_finished = False

    headers = {'x-apikey': api_key}
    with open(file_path, 'rb') as file:
        files = {'file': (os.path.basename(file_path), file)}

    spinner_thread = threading.Thread(target=spinner)
    spinner_thread.start()

    response = requests.post(upload_url, headers=headers, files=files)

    #업로드 완료되고 스피너 종료
    upload_finished = True
    spinner_thread.join()
```

악성코드 패밀리 식별 – VirustotalAPI 코드

```
#popular threat은 api로 구현이 되어있다
def get_popular_threat_categories(api_key):
    url = "https://www.virustotal.com/api/v3/popular_threat_categories"
    headers = {'x-apikey': api_key}
    response = requests.get(url, headers=headers)
    if response.status_code == 200:
        categories = response.json()
        categories_str = json.dumps(categories, indent=4)
        print("Popular Threat Categories:")
        print(categories_str) # 콘솔

        with open('popular_threat_categories.txt', 'w') as file: # +파일
            file.write(categories_str)
        print("Popular threat categories saved to popular_threat_categories.txt")
    else:
        print(f"Failed to get popular threat categories: {response.status_code} - {response.text}")

#악성코드 패밀리 식별은 없어서 리포트에서 따로 추출
def extract_malicious_families(report):
    results = report.get('data', {}).get('attributes', {}).get('results', {})
    families = []

    for engine, result in results.items(): #엔진 _
        if result.get('category') == 'malicious':
            family_info = result.get('result', '')
            if family_info: # 결과가 있는 경우에만 추가
                families.append(family_info)

    # 중복 제거 및 정렬
    unique_families = sorted(set(families))

    # 파일만
    with open('malicious_families.txt', 'w') as file:
        for family in unique_families:
            file.write(f"{family}\n")

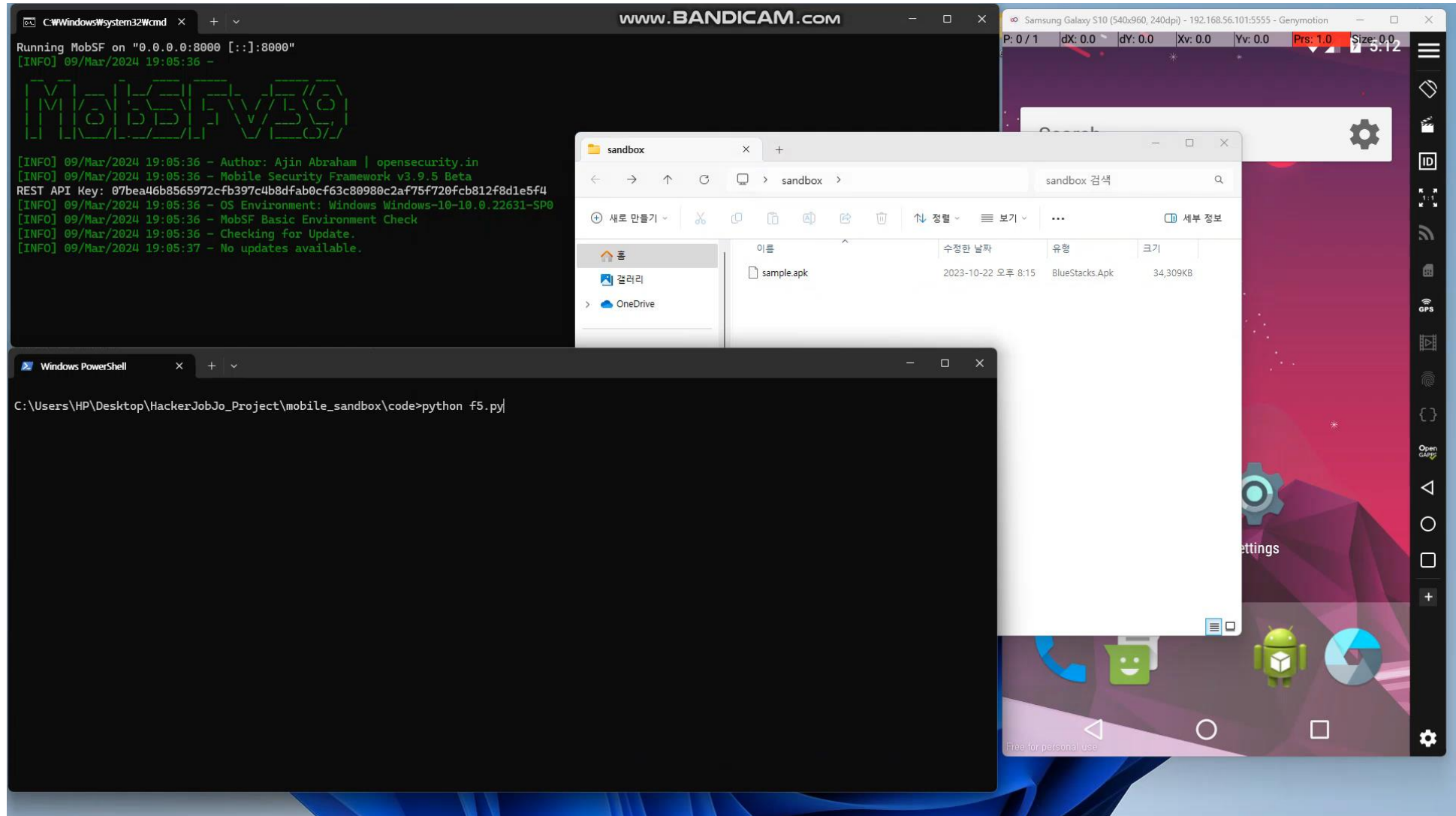
    print("Malicious families saved to malicious_families.txt")
```

악성코드 패밀리 식별 – VirustotaAPI 실행 결과

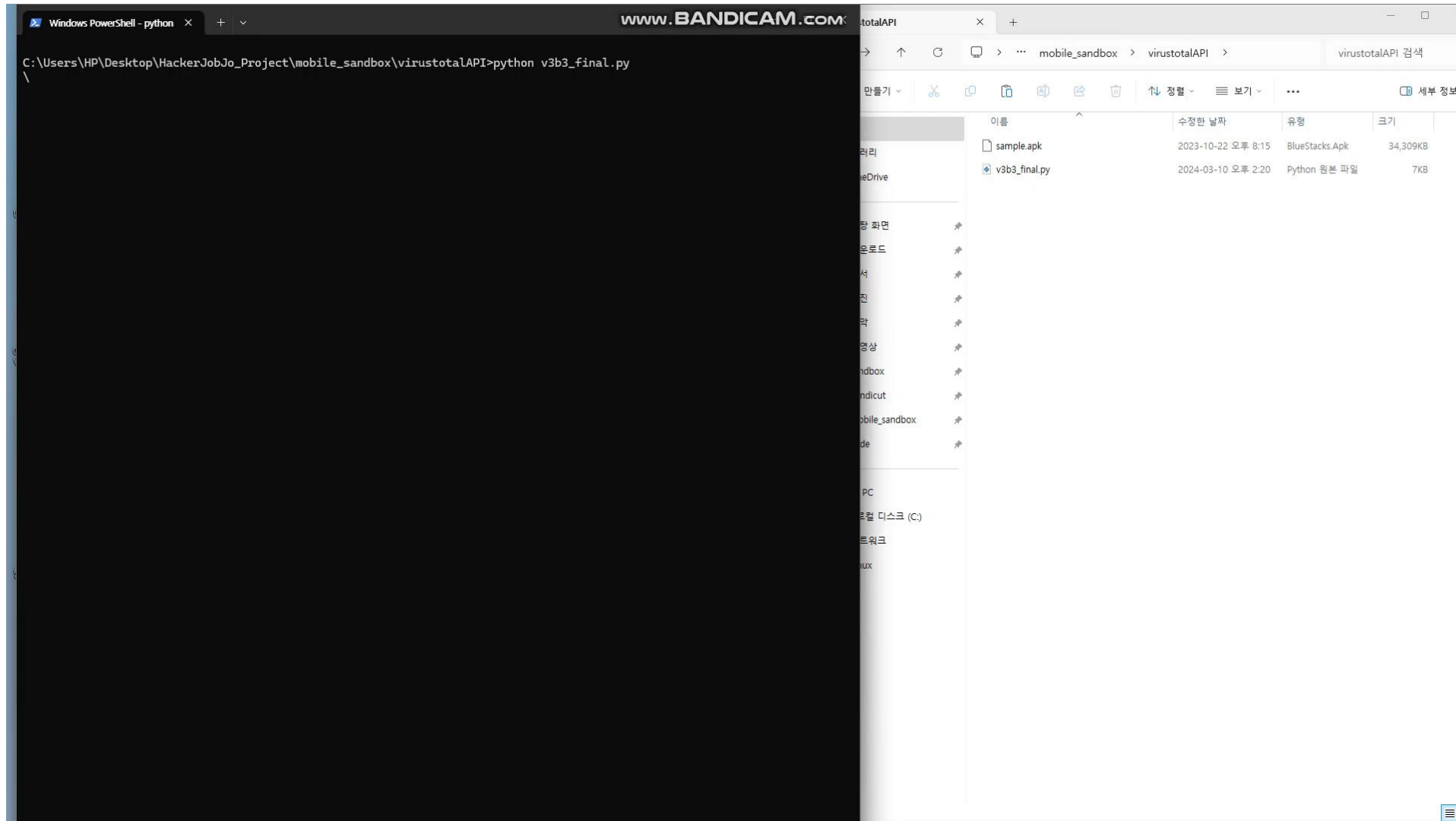
↑ VirustotalAPI_MaliciousFamilies (1) VirustotalA				
	이름	수정한 날짜	유형	크기
	Identifying Malicious Code Families.mp4	2024-03-10 오후 2:49	MP4 - MPEG-4 ...	10,787KB
4면	malicious_families.txt	2024-03-10 오후 3:44	텍스트 문서	6KB
드	popular_threat_categories.txt	2024-03-10 오후 2:29	텍스트 문서	1KB
	report.txt	2024-03-10 오후 2:29	텍스트 문서	27KB
rtcut	v3b3_final.py	2024-03-10 오후 4:00	Python 원본 파일	7KB
프로젝트				

프로젝트 결과

MobSF 커스텀 API 자동화 시연 영상



Virultotal_API로 악성코드 패밀리 식별 시연 영상



프로젝트 회고

자체 평가 및 느낀점

환경을 동일하게 세팅하라고 조언해주셨지만 서로 다르게 해서 같이 맞추기 힘들었다.

DEX파일 분석을 통해 Java언어 습득의 필요성을 느끼게 되었다.

내부 pgsHZz 악성코드 분석을 제대로 못해봐서 아쉬웠지만 좋은 경험이 되었다.

안드로이드 아키텍처에 대한 개념과 APK 파일 구조의 개념을 공부할 수 있어 좋았다.

MobSF API 자동화 코딩 과정 경험을 통해 프로그래밍에 재미를 다시한번 느꼈고

Virustotal API를 이용한 악성코드 패밀리 식별도 구현하게 되어 더욱 성장했다.

감사합니다.