



2차 프로젝트 보고서

태그

1. 기획

프로젝트 명: 모바일 샌드박스 구축

팀명: 해커잡조

1.1 팀 구성원 및 담당 업무

1.2 프로젝트 목적 및 필요성

2. 내용

2.1 모바일 샌드박스 구축 프로젝트 일정

2.2 단계 별 프로세스

2.3 시스템 환경

2.4 dex파일 복호화 및 리패키징

2.5 리패키징한 apk파일 서명 후 에뮬레이터 설치

2.6 jadx분석 및 Frida 우회Script 만들기

2.7 Frida코드 주입을 통한 우회하기 및 어플 실행

2.8 MobSF API를 이용한 분석 자동화

2.9 최종 결과

2.9.1 소스코드 압축파일

2.9.2 시연 동영상

3. 회고

프로젝트 자체 평가 및 느낀 점

홍영창

구현모

김동진

장다솜

조운지

최승희

4. 참고문헌

1. 기획

프로젝트 명: 모바일 샌드박스 구축

팀명: 해커잡조

1.1 팀 구성원 및 담당 업무

이름	역할	담당 업무
----	----	-------

공통진행		MobSF 환경 구축 및 분석
홍영창	팀장	jadx 코드 분석 및 노션 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

1.2 프로젝트 목적 및 필요성

이 프로젝트의 주된 목적은 모바일 악성코드를 분석하고 이해한 과정을 토대로 MobSF와 Frida와 같은 오픈소스 도구를 활용하여 정적/동적분석 전 자동화 구축이다. 이를 통해 얻는 장점은 다음과 같다.

첫번째, 안드로이드 앱에 대한 아키텍처를 이해하여 모바일 악성코드의 형성 과정을 파악하고, 앱의 리소스 파일과 소스 코드를 분석하여 보안 취약점을 식별할 수 있다.

두번째 MobSF와 Emulator를 통해 정적/동적분석과정을 경험하여 파일 내 악성행위에 대한 이해를 높이고 Frida 오픈소스 도구를 이용하여 우회 스크립트를 작성하여 보안 탐지 우회에 대한 이해를 높일 수 있다.

마지막으로 파악한 악성코드 샘플을 MobSF API 커스터마이징을 통한 정적분석과 동적분석 자동화를 진행하여 앱의 더욱 신속하고 효율적인 보안 평가가 이루어질 수 있어 보안 검사 프로세스의 빠른 진행을 가능하게 한다.

2. 내용

2.1 모바일 샌드박스 구축 프로젝트 일정

WBS_해커잡조 (자동 저장됨).xlsx

주요 업무	세부 업무	담당	소요일	기간																				
				2월														3월						
				19	20	21	22	23	24	25	26	27	28	29	1	2	3	4	5	6	7	8	9	10
프로젝트 기획	프로젝트 계획 수립	공통	1일																					
	MobSF 구조 이해하기	공통	2일																					
분석	MobSF, Genymotion 환경 구축	공통	3일																					
	암호화된 DEX 파일이 포함된 APK 분석	공통	4일																					
	APK 복호화 후 리패키징 및 서명	공통	4일																					
	Frida 연동, Jadx 분석 및 우회코드 실행	공통	4일																					
	MobSF를 이용한 앱 정적/동적 분석	현모, 동진	7일																					
	API 커스터마이징을 통한 분석 자동화	운지, 승희	5일																					
	자동화 프로그램 개발 및 테스트	운지, 승희	5일																					
문서화 작업	WBS 및 보고서 작성	장다솜	4일																					
	프로젝트 종결, 노션 및 발표	홍영창	4일																					
	Notion & Github	공통	22일																					

2.2 단계 별 프로세스

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
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 > <u>CyberChef</u>
	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
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

2.3 시스템 환경

- windows10 , 11
- python version : 3.11.7
 - 파이썬버전이 여러종류가 있으면 인식을 하지못해서 MobSF실행 불가능
 - 운지님, 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

▼ 2.4 dex파일 복호화 및 리패키징



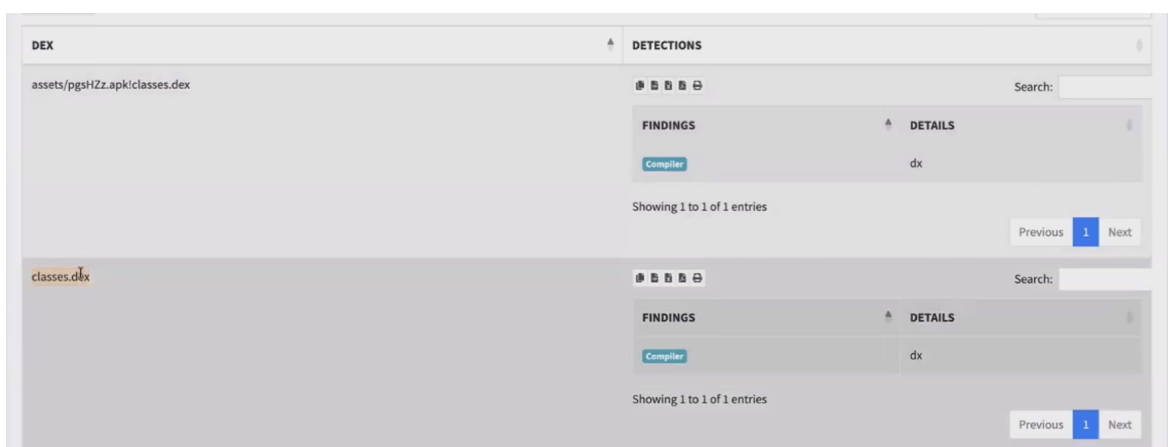
MobSF의 정적분석 기능을 이용해서 sample.apk를 분석하려 하는데 apk안에 암호화된 dex 파일들이 있어 암호화된 dex파일들(kill-classes.dex)이 정적분석에 안나온다. 이를 나오게끔 해결해야 한다.

sample.apk 리패키징하기

sample 검색				
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



kill-classes.dex 파일은 안보인다. 암호화되어 있어서 정상파일밖에 안보인다.


그러면 악성 dex파일을 어떻게 확인할 수 있냐? → apktool로 복호화 리패키징 하여 확인할 수 있다.

[Mobile] Windows Apktool 설치

설치 경로 : <https://ibotpeaches.github.io/Apktool/install/> wrapper script 우클릭하여 다른이름으로 링크 저장을 통해 apktool.bat으로 저장 find newest here에서 최신 버전의 jar 파일 설치 2에서 설치받은 파일을

 <https://cgy12306.tistory.com/393>

click, Save Link As **apktool.bat**

 to your Windows directory (Usually **C://Windows**)
you may place the two files anywhere then add that directory to

⇒ apktool 설치 방법 블로그 설명

Install Guide | Apktool

A minimum of Java 8 is required to run Apktool.



 <https://apktool.org/docs/install/>

Reverse engineering Android **APKs**

⇒apktool 설치 홈페이지

apktool을 사용하여 **sample.apk** 파일을 디컴파일 후 리패키징 해야된다.

sample.apk (unpack1) 복호화

```
apktool d -s sample.apk -o unpack
```

- **d** : apktool의 디코드(디컴파일) 명령어입니다. APK 파일을 디코딩하여 원시 리소스 및 소스 코드를 추출합니다.
- **s** : 이 옵션은 apktool이 리소스를 디코드하는 데 사용하는 특정 설정을 지정합니다. **s** 옵션을 사용하면 특정 리소스 ID가 변하지 않고 유지되도록 할 수 있습니다.
- **o unpack** : 디코딩된 파일을 출력할 디렉토리를 지정합니다. 여기서 ****unpack***는 디코딩된 파일이 저장될 폴더 이름입니다. 필요에 따라 이 이름을 원하는 것으로 변경할 수 있습니다.

위 명령어를 이용하여 디컴파일한다.

<div> <div> <div></div> <div>> 내 PC > 다운로드 > unpack ></div> </div> <div> <div> <div></div> <div>unpack 검색</div> </div> </div> </div>				
	이름	수정된 날짜	유형	크기
<div> <div> <div></div> <div>프로젝트</div> </div> <div> <div></div> <div>개인</div> </div> </div>	assets	2024-03-05 오후 9:09	파일 폴더	
	lib	2024-03-05 오후 9:09	파일 폴더	
	original	2024-03-05 오후 9:09	파일 폴더	
	res	2024-03-05 오후 9:09	파일 폴더	
	unknown	2024-03-05 오후 9:09	파일 폴더	
	AndroidManifest.xml	2024-03-05 오후 9:09	Microsoft Edge H...	8KB
	apktool.yml	2024-03-05 오후 9:09	Yaml 원본 파일	1KB
	classes.dex	2024-03-05 오후 9:09	DEX 파일	13KB
	kill-classes.dex	2024-03-05 오후 9:09	DEX 파일	2,550KB

unpack 폴더에 압축이 풀렸다.

이 상태에서 `kill-classes.dex` 파일을 cyber cheif에 올려줘서 복호화한다.

내 PC > 다운로드 > unpack2

unpack2 검색

이름	수정한 날짜	유형	크기
assets	2024-03-05 오후 9:23	파일 폴더	
kotlin	2024-03-05 오후 9:23	파일 폴더	
lib	2024-03-05 오후 9:23	파일 폴더	
original	2024-03-05 오후 9:23	파일 폴더	
res	2024-03-05 오후 9:23	파일 폴더	
unknown	2024-03-05 오후 9:23	파일 폴더	
AndroidManifest.xml	2024-03-05 오후 9:23	Microsoft Edge H...	20KB
apktool.yml	2024-03-05 오후 9:23	Yaml 원본 파일	1KB
classes.dex	2024-03-05 오후 9:23	DEX 파일	14KB
classes2.dex	2024-03-05 오후 9:27	DEX 파일	8,515KB
classes3.dex	2024-03-05 오후 9:25	DEX 파일	4,777KB

대체 성공!

여기서 헛갈리면 안되는데

```
apktool b unpack2 -o pgshZz.apk
```

이제 다시 `pgshZz.apk` 로 리패키징 하기 위해 위의 명령어를 사용한다.

내 PC > 다운로드


다운로드 검색

이름	수정한 날짜	유형
오늘 (2)		
unpack2	2024-03-06 오후 6:57	파일 폴더
pgsHZz.apk	2024-03-06 오후 6:57	APK 파일
어제 (2)		

리패키징 성공! 이제 복호화 파일들로 대체하고 리패키징된 `pgshZz.apk` 파일을 `sample.apk\assets` 경로에 원래 `pgshZz.apk` 파일을 삭제하고 대체한다.

내 PC > 다운로드 > unpack > assets

assets 검색

이름	수정한 날짜	유형	크기
 pgsHZz.apk	2024-03-05 오후 9:09	APK 파일	27,727KB

원래파일 지우고..

<div> <div> <div></div> <div>내 PC > 다운로드 > unpack > assets</div> </div> <div>assets 검색</div> </div>			
이름	수정한 날짜	유형	크기
pgsHZz.apk	2024-03-06 오후 6:57	APK 파일	20,243KB

대체 성공!

sample.apk (unpack1) 디컴파일 후 리패키징

이제 안의 `pgsHZz.apk` 파일 안의 dex파일들은 다 복호화 된것들이니 한꺼번에 리패키징을 해준다.
(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.

<div> <div> <div></div> <div>내 PC > 다운로드</div> </div> <div>다운로드 검색</div> </div>			
이름	수정한 날짜	유형	
<div> <div>오늘 (3)</div> <div> <div>unpack</div> <div>unpack2</div> <div>sample2.apk</div> </div> </div>			
	2024-03-06 오후 7:02	파일 폴더	
	2024-03-06 오후 6:57	파일 폴더	
	2024-03-06 오후 7:02	APK 파일	
<div> <div>어제 (1)</div> </div>			

리패키징 성공! 이제 MobSF에 올려서 정적분석 해보자

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

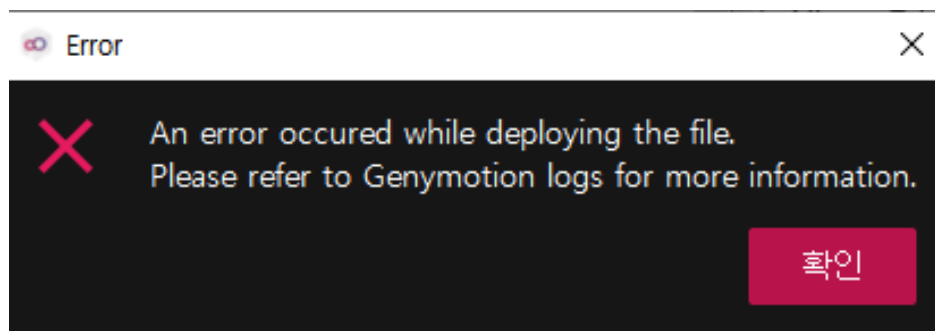
5개의 dex파일들이 MobSF에 정적분석 기능을 통해 나오게 된다. (원래는 암호화 안된 2개 파일만 나왔었다.)

목표 달성 성공!

▼ 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 신원 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] -keysize [키사이즈]
keytool -genkey -v -keystore honghong.keystore -alias honghong -keyalg RSA -keysize 2048
```

```
>keytool -genkey -v -keystore honghong.keystore -alias honghong -keyalg RSA -keysize 2048
```

키 저장소 비밀번호 입력:

새 비밀번호 다시 입력:

이름과 성을 입력하십시오.

[Unknown]: honghong

조직 단위 이름을 입력하십시오.

[Unknown]: honghong

조직 이름을 입력하십시오.

[Unknown]: honghong

구/군/시 이름을 입력하십시오?

[Unknown]: Seoul

시/도 이름을 입력하십시오.

[Unknown]: Seoul

이 조직의 두 자리 국가 코드를 입력하십시오.

[Unknown]: KO

CN=honghong, OU=honghong, O=honghong, L=Seoul, ST=Seoul, C=KOR

(가) 맞습니까?

[아니오]: y

다음에 대해 유효 기간이 90일인 2,048비트 RSA 키 쌍 및 자체 서명된 인증서(SHA256withRSA)를 생성하는 중

: CN=honghong, OU=honghong, O=honghong, L=Seoul, ST=Seoul, C=KO

<honghong>에 대한 키 비밀번호를 입력하십시오.

(키 저장소 비밀번호와 동일한 경우 Enter 키를 누름):
새 비밀번호 다시 입력:
[honghong.keystore을(를) 저장하는 중]

sample2.apk	2024-03-06 오후 7:02	APK 파일
honghong.keystore	2024-03-06 오후 7:37	KEYSTORE 파일

등록절차가 끝나면 keystore가 저장되었다고 나오고 위에 keystore가 생긴 것을 확인할 수 있다.

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] [서명할  
APK] [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_internal.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]
```

jar signed.

Warning:

The signer's certificate is self-signed.

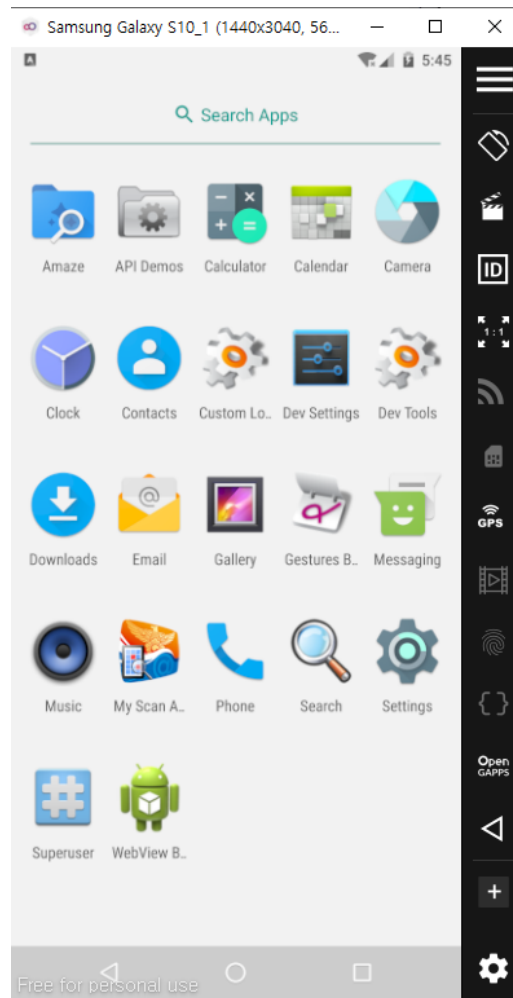
The SHA1 algorithm specified for the -digestalg option is considered a security risk and is disabled.

The SHA1withRSA algorithm specified for the -sigalg option is considered a security risk and is disabled.

처음엔 keystore 서명 파일에 대한 비밀번호를 입력하라고 나온다.

입력하고 나면 뭔가 쪽 나오면서 결국 마지막엔 `jar signed` 라고 나오면 서명이 정상적으로 완료된 것이다.

에뮬레이터에 sample2.apk 설치해보기



서명완료한 sample2.apk파일을 지니모션 에뮬레이터에 옮기니까 잘 설치된 모습입니다.

adb 명령어를 통해 설치하는 법

```
adb install [리패키징한APK파일]
```

▼ 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 목적에 있다.

[re_sample_report.pdf](#)

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.
- **Build.TAGS check:** 애플리케이션이 실행 중인 디바이스의 빌드 태그를 확인하는 코드가 포함되어 있음을 나타냅니다. 가상 머신에서의 빌드 태그는 종종 "test-keys"와 같은 특정 값들을 포함할 수 있습니다.

Build.TAGS

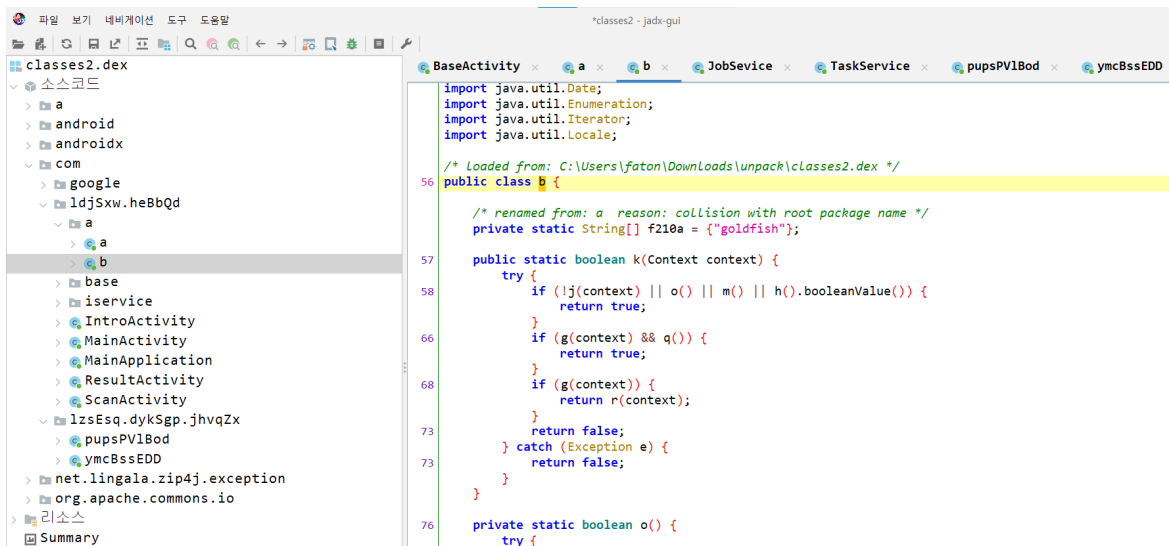
com > IdjSxw > heBbQd > p017a > C0740b



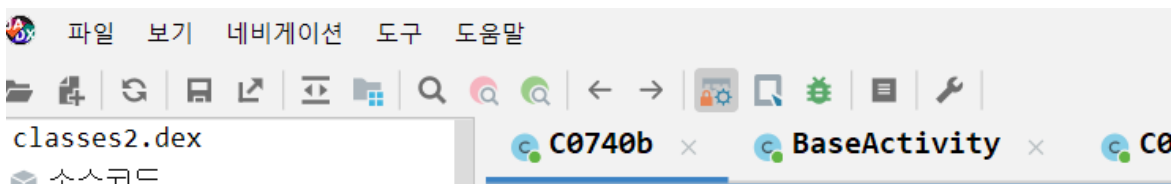
jadx도구를 통한 classes2.dex 파일 분석 과정

unpack 검색				
이름	수정된 날짜	유형	크기	
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.dex파일을 jadx로 분석한다. (classes.dex 파일에는 VM 탐지 코드가 없음을 확인함)



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

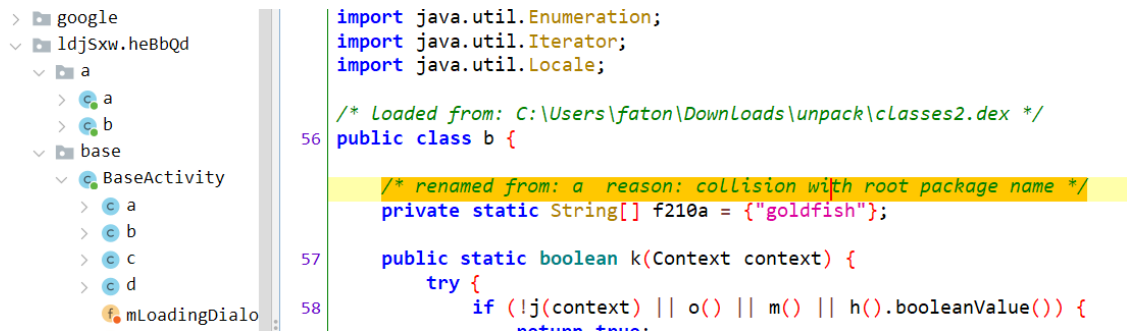


코드를 알아보기 쉽게 난독화 도구를 이용한다. b 클래스 → C0740b

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.



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.

C0740b 클래스 코드 분석하기 (난독화 진행)

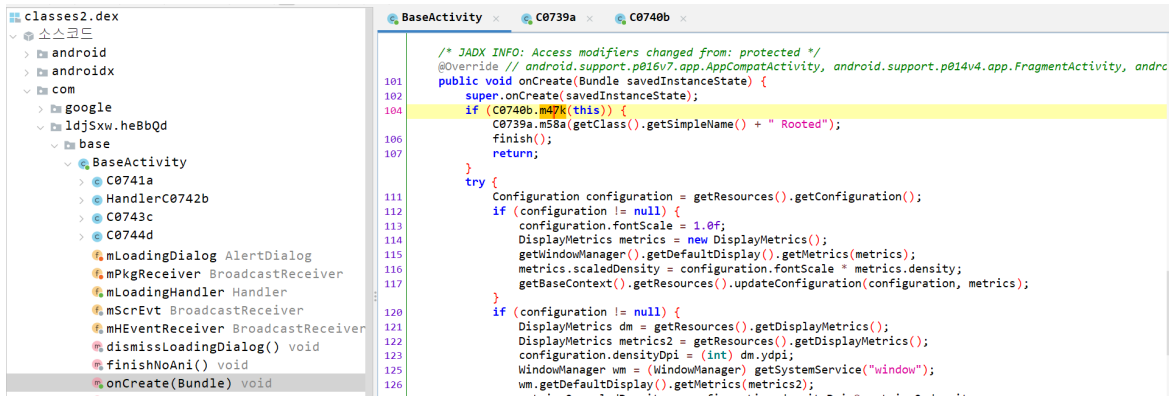
jadx.java



이 코드에서 사용되는 안드로이드 디바이스의 상태를 확인하는 메서드는 총 7개이다. VM탐지에 성공하면 True를 반환한다. 이를 우회하려면 false를 반환해야 한다.

VM을 우회하는 7개 Detection함수 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 참지 (True 반환)

2. test-keys

```

57  /* renamed from: o */
58  private static boolean m43o() {
59      try {
60          String buildTags = Build.TAGS;
61          if (buildTags != null) {
62              if (!buildTags.contains("test-keys")) {
63                  return false;
64              }
65              return true;
66          }
67          return false;
68      } catch (Exception e) {
69          return false;
70      }
71  }

```

m43o method - 디바이스가 테스트용 키(test-keys)를 사용하는지 확인

- BuildTag에 test-keys가 포함되어 있을 경우 참지 (True 반환)
- "test-key" 보통 개발자버전의 안드로이드 빌드에서 사용됨

3. Root

```
73      /* renamed from: m */
74      private static boolean m45m() {
75          try {
76              File file = new File("/system/app/Superuser.apk");
77              if (!file.exists()) {
78                  return false;
79              }
80              return true;
81          } catch (Exception e) {
82              return false;
83          }
84      }
85  }
```

m45m method - 디바이스에 Superuser 앱이 설치되어 있는지 확인

- Superuser.apk 파일 설치되어 있을 경우 참지 (True 반환) (rooting 관련 패키지)

4. Emulator

```
86      /* renamed from: h */
87      private static Boolean m50h() {
88          String[] strArr;
89          try {
90              File driver_file = new File("/proc/tty/drivers");
91              if (driver_file.exists() && driver_file.canRead()) {
92                  byte[] data = new byte[(int) driver_file.length()];
93                  try {
94                      InputStream inStream = new FileInputStream(driver_file);
95                      inStream.read(data);
96                      inStream.close();
97                  } catch (FileNotFoundException e) {
98                  } catch (IOException e2) {
99                  }
100                  String driver_data = new String(data);
101                  for (String known_qemu_driver : f1279a) {
102                      if (driver_data.indexOf(known_qemu_driver) != -1) {
103                          return true;
104                      }
105                  }
106              }
107          } catch (Exception e3) {
108          }
109          return false;
110      }
```

m50h method - 에뮬레이터인지를 확인

- /proc/tty/drivers 파일 내 **goldfish** 라는 값이 있을 경우 참지 (True 반환)
- **goldfish** 문자열은 안드로이드 에뮬레이터의 커널 이름

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 참지 (True 반환)
- 안드로이드 시스템 설정에서 **adb_enabled** 값을 가져와 1이면 ADB가 활성화, 0이면 비활성 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 참지 (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) {
131                  proxyAddress = System.getProperty("http.proxyHost");
132                  String portstr = System.getProperty("http.proxyPort");
133                  proxyPort = Integer.parseInt(portstr != null ? portstr : "-1");
134                  PrintStream printStream = System.out;
135                  printStream.println(proxyAddress + "~");
136                  PrintStream printStream2 = System.out;
137                  printStream2.println("port = " + proxyPort);
138              } else {
139                  proxyAddress = Proxy.getHost(context);
140                  proxyPort = Proxy.getPort(context);
141                  Log.e("address = ", proxyAddress + "~");
142                  Log.e("port = ", proxyPort + "~");
143              }
144              return (TextUtils.isEmpty(proxyAddress) || proxyPort == -1) ? false : true;
145          } catch (Exception e) {
146              return false;
147          }
148      }

```

m40r method - 디바이스의 프록시 설정 여부를 확인

- 디바이스에 프록시 설정이 구성되어 있는지 확인하는 함수
- API버전에 따라 다르게 동작하도록 설정
- API 14미만에서는 Proxy 클래스의 메서드를 사용해 프록시 설정을 가져옴
- proxyAddress, proxyPort == -1으로 되어 있다면 프록시가 설정되지 않았음으로 판단
- 이외의 예외가 발생하면 프록시가 설정되지 않음으로 처리

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;
    };
});

```

탐지 코드의 호출을 후킹하여 항상 **false**를 리턴하게 스크립트 작성

▼ 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/
```

3. adb shell 접속 후 전송한 frida server 파일 실행

```
cd /data/local/tmp/
chmod +x frida-server-16.2.1-android-x86
./frida-server-16.2.1-android-x86

./frida-server-16.2.1-android-x86 & #<= 추천. 백그라운드실행 (이후 동적 분석자동화시 필요)
```

#연결 프로세스 확인

```
C:\Users\HP>frida-ps -U
```

```
C:\Windows\system32\cmd.exe - adb shell
Microsoft Windows [Version 10.0.19045.4046]
(c) Microsoft Corporation. All rights reserved.

C:\Users\fatton>adb shell
vbox86p:/ # cd /data/local/tmp
vbox86p:/data/local/tmp # ./frida-server-16.2.1-android-x86
```

이렇게 frida 서버를 켜놓고(실행중이어야 함) 다른 cmd창을 켜서 클라를 실행한다.
\$하면 켜놓지 않아도 된다.

- `frida-ls-devices` : 연결 가능한 기기 목록 출력
 - 출력 결과에 있는 device id를 이용하여 해당 기기와 상호작용 가능

```
C:\Users\fatton\Downloads>frida-ls-devices
Id                Type      Name                OS
-----
local            local    DESKTOP-CG1D271    Windows 10.0.19045
192.168.56.102:5555  usb      Galaxy S10          Android 7.1.1
barebone         remote   GDB Remote Stub
socket           remote   Local Socket
```

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

▼ frida_script.js

```
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;
    };
});
```

```
C:\Users\fatoni\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
2919	My Scan APP	com.lджSxw.heBbQd
1543	Superuser	com.genymotion.superuser
-	API Demos	com.example.android.apis
-	Amaze	com.amaze.filemanager
-	Calculator	com.android.calculator2
-	Calendar	com.android.calendar
-	Camera	com.android.camera2
-	Contacts	com.android.contacts
-	Custom Locale	com.android.customlocale2
-	Dev Tools	com.android.development
-	Development Settings	com.android.development_settings
-	Email	com.android.email
-	Files	com.android.documentsui
-	Messaging	com.android.messaging
-	Music	com.android.music
-	Phone	com.android.dialer
-	Search	com.android.quicksearchbox
-	Settings	com.android.settings
-	WebView Shell	org.chromium.webview_shell
-	com.android.gesture.builder	com.android.gesture.builder

위의 My Scan APP이 실행중인 것을 확인한다.

```
명령 프롬프트 - frida -U -n "My Scan APP" -l 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)

[Galaxy S10::My Scan APP ]-> _

```

다시 `frida -U -n "My Scan APP" -l frida_script.js` 명령어를 실행하면 붙는다.

이제 genymotion에서 `My Scan APP` 앱을 실행하면 된다.

▼ 2.8 MobSF API를 이용한 분석 자동화

정적분석

```
import subprocess
import os
import requests
from Cryptodome.Cipher import AES #crypto나 cryptography로도 해봤으나 원
from Cryptodome.Util.Padding import unpad #패딩과정 넣으면 38점 안하면 2
import glob
import time
import threading
import json

def decompile_apk(apk_path, output_dir):
    # APK 파일 디컴파일
    try:
        subprocess.run(['C:\\\\Windows\\apktool.bat', 'd', '-r', '-s'
        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.
```

```

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

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

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

    return True

except Exception as e:
    print(f"Decrypting Error: {e}")
    return False

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) == os.path.basename(encrypted_dex_path)])

        # 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, key_alias):
    # 디컴파일된 APK 재컴파일
    try:
        subprocess.run(['C:\\Windows\\apktool.bat', 'b', '-f', decompiled_dir, output_apk_path])
        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}")

except subprocess.CalledProcessError as e:
    print(f"APK 재컴파일 또는 서명 실패: {e}")

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)}
            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

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)
    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.")

```

동적분석

```

def get_APPS(self):
    headers = {'Authorization': self.api_key}
    response = requests.get(f'{self.server}/api/v1/dynamic/get_apps')
    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', data=data, headers=headers)
    if response.status_code == 200:
        print("mobsfy: ", response.json())
    else:
        print("mobsfy Error: ", response.status_code, response.json())

```



```

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/sta
    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
    def run_script():
        print("안티 VM 우회를 위해 Frida 스크립트 실행 중...")
        self.process = subprocess.Popen(['frida', '-U', '-l', b
        stdout=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
        if response.status_code == 200:
            print("SSL/TLS Success: ", response.json())
        else:
            print("SSL/TLS Error: ", response.status_code, resp
    except requests.exceptions.RequestException as e:
        print("HTTP Request Failed: ", e)
    except json.decoder.JSONDecodeError:
        print("JSON decoding failed")

```

```

# 윤지님 ssl tls test
# def dynamic_tls_ssl_test(self, file_hash): #이것도 비동기 백그라운드 실행
#     """TLS/SSL 보안 테스트를 백그라운드에서 수행하고 결과를 출력"""

#     def run_test():
#         if not file_hash:
#             print("파일 해시가 제공되지 않았습니다.")
#             return

#         print("TLS/SSL 보안 테스트 수행 중...")
#         headers = {'Authorization': self.api_key}
#         data = {'hash': file_hash}
#         response = requests.post(f'{self.server}/api/v1/android/ssl/tls/test',
#                                  data=data, headers=headers)

#         if response.status_code == 200:
#             print("TLS/SSL 보안 테스트가 성공적으로 완료되었습니다.")
#             results = response.json()
#             print(json.dumps(results, indent=4))
#         else:
#             print("TLS/SSL 보안 테스트 수행에 실패했습니다.")

#     # 백그라운드 스레드에서 TLS/SSL 보안 테스트 실행
#     test_thread = threading.Thread(target=run_test)
#     test_thread.start()

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/static-analysis/start',
                             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)
    except:
        pass

```

```

        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.")

# def get_dynamic_analysis_status(self, file_hash):
#     """동적 분석 상태 확인"""
#     print("동적 분석 상태 확인 중...")
#     headers = {'Authorization': self.api_key}
#     response = requests.get(f'{self.server}/api/v1/dynamic/status/{file_hash}', headers=headers)
#     result = response.json()
#     if response.status_code == 200:
#         print("동적 분석 상태: ", result['status'])
#         print("동적 분석 결과", result)
#         return result
#     else:
#         print("동적 분석 상태 확인 실패.")
#         return None

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/status/{file_hash}', headers=headers, json=data)
    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/repo
    response = requests.post(f'{self.server}/api/v1/dynamic/repo
    if response.status_code == 200:
        report_path = f'{os.path.splitext(self.file_path)[0]}_c
        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.")

if __name__ == "__main__":
    server_url = 'http://127.0.0.1:8000' # MobSF 서버 URL
    api_key = '9eca7005f71ab259f2f7b0a51c672e381b7949180474b6382769
    apk_path = 'C:\\Users\\faton\\Downloads\\test\\sandbox\\sample.
    aes_key = b'dbcdfghijklmaop'
    decompiled_dir = 'C:\\Users\\faton\\Downloads\\test\\sandbox\\c
    output_dir = 'C:\\Users\\faton\\Downloads\\test\\sandbox'
    nested_decompiled_dir = 'C:\\Users\\faton\\Downloads\\test\\sar

    keystore_path = 'C:\\Users\\faton\\Downloads\\test\\sandbox\\hc
    keystore_password = "honghong" # Keystore 비밀번호
    key_alias = 'honghong' # 키 별칭

    bypass_script_path = 'C:\\Users\\faton\\Downloads\\test\\sandbc
    package_name = 'com.ldjSxw.heBbQd'

    # APK 디컴파일, 중첩된 APK 처리, DEX 파일 복호화
    decompile_apk(apk_path, decompiled_dir)
    find_nested_apks_and_decompile(decompiled_dir)
    process_dex_files(decompiled_dir)

    # APK 재컴파일 및 서명
    recompiled_apk_path = os.path.join(output_dir, 're_sample.apk')
    recompile_apk(decompiled_dir, recompiled_apk_path, keystore_pat
    re_nested_apk_path = os.path.join(output_dir, 're_pgshZz.apk')
    recompile_apk(nested_decompiled_dir, re_nested_apk_path, keysto

```

```

# MobSF 작업 시작
mobSF = MobSF_API(server_url, api_key, recompiled_apk_path)
file_hash = mobSF.upload()
mobSF_nested = MobSF_API(server_url, api_key, re_nested_apk_path)
nested_file_hash = mobSF_nested.upload()

if nested_file_hash:
    # 스캔 및 PDF 리포트 다운로드
    mobSF_nested.scan(nested_file_hash)
    mobSF_nested.download_pdf_report(nested_file_hash)

if file_hash:
    # 스캔, PDF 리포트 다운로드
    mobSF.scan(file_hash)
    mobSF.download_pdf_report(file_hash)

    # 동적 분석 시작
    identifier = mobSF.get_APPS()
    mobSF.mobsfying(identifier)
    dynamic_results = mobSF.start_dynamic_analysis(file_hash)
    mobSF.tls_test(file_hash)

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

        mobSF.start_activity_analysis(file_hash, 'com.ldjSxw.he
        # 동적 분석 중 TLS/SSL 보안 테스트 수행
        # mobSF.dynamic_tls_ssl_test(file_hash) #비동기 백그라운드
        mobSF.start_activity_analysis(file_hash, 'com.ldjSxw.he
        mobSF.start_activity_analysis(file_hash, 'com.ldjSxw.he
        mobSF.start_activity_analysis(file_hash, 'com.ldjSxw.he

        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)

mobSF.tap_screen(177, 515)# accessibility
time.sleep(5)

mobSF.tap_screen(90, 242)# sonqLgOT
time.sleep(5)

mobSF.tap_screen(464, 159)# sonqLgOT
time.sleep(5)

mobSF.tap_screen(434, 814)# ok
time.sleep(5)

mobSF.tap_screen(418, 920)# □
time.sleep(5)

mobSF.tap_screen(298, 168)# 새창
time.sleep(5)

mobSF.tap_screen(279, 721)# 검사시작
time.sleep(10)

mobSF.tap_screen(287, 699)# 치료하기
time.sleep(5)

mobSF.stop_frida_script()

```

```
# 동적 분석 멈춤
mobSF.stop_dynamic_analysis(file_hash)

# 동적 분석 리포트 다운로드
mobSF.download_dynamic_report(file_hash)
```

2.9 최종 결과

2.9.1 소스코드 압축파일

[mobsfAPI.py](#)

[2024_mobile_3회차_2조_해커잡조.zip](#)

2.9.2 시연 동영상

MobSF 커스텀 API로 자동화 / 업로드 - 정적분석 - 동적분석 - 결과물

<https://prod-files-secure.s3.us-west-2.amazonaws.com/601f866a-fc0b-4e79-9803-30a518d349f4/c0947656-883b-4efd-bfc9-d5578bef6f88/final4.mp4>

+VirusTotal_API로 악성코드 패밀리 식별

https://prod-files-secure.s3.us-west-2.amazonaws.com/601f866a-fc0b-4e79-9803-30a518d349f4/6d8364f2-54d0-4743-bf5a-eadf061aff46/Identifying_Malicious_Code_Families.mp4

3. 회고

프로젝트 자체 평가 및 느낀 점

홍영창

팀장으로서 프로젝트에 대한 이해를 빨리하고 방향을 잡아야 했지만 프로젝트에 대한 경험과 이해가 부족하여 어떻게 진행해야 될지 막막했으나, 멘토님의 질의응답 시간을 통해 목표와 방향성을 잡고 조금씩 진행해 나갔다.

윈도우 실행파일 악성코드 분석은 해본 경험이 있지만 안드로이드 악성코드 앱 분석은 경험이 없어서 안드로이드 아키텍처에 대한 개념과 파일 구조에 대해 공부해야 했다. 그리고 jadx 분석도구를 통해 앱 동작 과정을 100% 이해할 수 없었지만 어느 정도는 이해할 수 있어서 재밌었고 기존에는 c언어와 python언어 공부를 해왔었다면 DEX 파일을 분석하면서 APK는 디컴파일시 Java소스코드로 되어 있기 때문에 Java언어에 대한 공부 필요성도 느끼게 되었다.

내부 Nested apk가 핵심적으로 동작하는 악성코드여서 나중에 이 부분도 추가로 공부해서 동작과정을 이해하고 싶다.

또 MobSF api를 통한 전 자동화 과정을 직접 만들어보진 못했지만 경험해 봄으로써 api에 관한 공부도 할 수 있어서 좋았다.

구현모

앱 관련하여 간단한 지식 밖에 없었는데 이번 계기를 통해 앱 구조 및 암호 복호화, 서명 등 개념을 배웠다.

진행하면서 오류가 많았는데 팀원들에게 도움을 받아 해결해나가면서 감사함을 많이 느꼈다...

작업 후반으로 갈수록 도움이 되지 못해 아쉬웠고 개인적으로 부족한 역량을 키워야겠다는 생각이 많이 들곤 했다.

다음 프로젝트 때는 더 열심히 해서 팀원들에게 도움이 되도록 노력해야겠다.

김동진

악성 어플리케이션을 분석을 MobSF를 활용하여 정적 분석 및 동적 분석을 해보면서 apk 파일 구조와 apk가 실행파일이 되는 과정을 배울 수 있었다. 악성 어플리케이션이 어떤 식으로 사용자를 속이고 설치 및 실행되는지도 경험할 수 있었다. 악성 어플리케이션 분석을 위한 샌드박스 구축에 시간을 많이 쓰거나 경험이 없어 처음에 감을 빨리 잡지 못하여 시간을 많이 보냈게 아쉬웠다. API를 활용한 자동화를 하면서 생각보다 많이 어려웠는데 팀원들이 많이 도와줘서 잘 진행되었다.

장다솜

이번 프로젝트를 통해 정적분석과 동적분석을 통한 APK 파일의 구조와 분석 방법에 대한 소중한 경험을 쌓을 수 있었다.

애플리케이션 보안에 대한 취약점과 경각심을 느꼈으며, 환경 구축과 연동의 어려움을 겪을 때마다 팀원들의 도움으로 매끄럽게 진행될 수 있었다. 하지만 자동화 소스코드와 에러 수정 부분에서 도움이 되지 못해 아쉬웠다.

이를 통해 팀원과의 소통과 협력의 중요성을 한 번 더 깨달았다.

조운지

악성코드 분석을 제대로 못해봐서 아쉬긴 했지만, 안드로이드와 에뮬에 대한 이해도가 생겼고 특히 mobsf API자동화를 코딩하는 과정에서 프로그래밍에 대한 재미를 느끼게 되어서 개인적으로 의미 있는 프로젝트였다.

추가적으로 악성코드 패밀리 식별을 Virustotal API로 해보면 어떨까하는 아이디어가 떠올라서 그냥 재미로 시도해보았는데, 짧은 시간 안에 결과물을 만들어서 스스로도 놀랐다.

저번 포트스캐너 프로젝트보다 더 낫설었던 프로젝트였는데 얻어가는게 많아서 구성을 잘해주신 멘토님께 감사하다는 말을 전하고 싶다.

최승희

어플리케이션은 처음 다뤄봐서 낫설었지만 색다른 느낌이어서 재밌었다. mobsf 구조 분석, sample.apk, dex파일 분석 모두 대략적으로 해보았는데 처음에는 너무 난해했지만 보다보니 이해가 가면서 즐거움을 느꼈다. 분석 과정에서 가장 큰 재미를 느꼈었던 것 같다. 이후에 이해한 것을 바탕으로 코드를 구현하는데 어려움을 느꼈지만 다른 팀원들과 함께하며 하나씩 돌파해나가는 재미도 있었다. 이번 프로젝트는 하나하나 이해하면서 해낸 느낌이라 의미있는 것 같다.

4. 참고문헌

ANDITER를 활용한 안드로이드 위협 탐지 및 우회 방안 : PART 3 (프리다, 피닝)

01. 프리다(Frida) 개요 및 우회방안 1) 프리다의 개요 'ANDITER를 활용한 안드로이드 위협 탐지 및 우회 방안 : PART 2'에서 동적 분석에 사용되는 디버그와 애플리케이션 분석 환경인 에뮬레이터의 탐지 및 우회방법에 대해 살펴보았다. PART3에서는 데이터 위·변조 및 분석

📄 <https://www.igloo.co.kr/security-information/anditer를-활용한-안드로이드-위협-탐지-및-우회-방안-part-3-프/>



⇒ frida 우회 방법

[Android] APK 파일 리패키징(repackaging)하는 방법

PC 환경 : Windows 10 x64 1. apktool 을 다운로드 받습니다. 아래 URL에 들어가서 우선 wrapper script 부터 다운로드 받습니다.

<https://ibotpeaches.github.io/Apktool/install/> Apktool - How to Install

👤 <https://domdom.tistory.com/287>



⇒ 서명 방법

https://mobsf.live/api_docs

⇒ API Docs