

그 컴퓨터들 중 하나는
매일 손에
들고 있습니다.

안드로이드 해킹 입문

김근호 · R00T-Kim
SSG 회장

JADX · Frida · Intent Redirection · WebView

질문 링크

질문은 여기로 남겨주세요.

발표 중 궁금한 점을 QR로 남기면 마지막
Q&A 시간에 같이 보겠습니다.

Android security intro · Q&A link



해킹이 뭐고, 안드로이드 해킹은 뭔가

해킹

시스템을 만든 사람이 예상하지 못한 방식으로 사용하고, 그 경계를 확인하는 일입니다.

Offensive Mindset

우리 동아리는 공격자의 관점을 중요하게 봅니다. 그래야 실제로 무엇이 위험한지 정확히 이해할 수 있습니다.

단, 실습과 연구는 허가된 대상과 책임 있는 범위 안에서 진행합니다.

왜 하필 안드로이드인가

3B+

Google이 공개한 30
억+ 활성 Android 기
기 규모

APK

앱 파일을 받아 노트북
에서 바로 열어볼 수 있
음

LOW

시작 비용이 낮고, 결과
가 눈에 잘 보임

입문 장벽이 낮습니다. 그래서 처음 보안 분석을 해보
기 좋은 분야입니다.

Android 공식 아키텍처로 보기

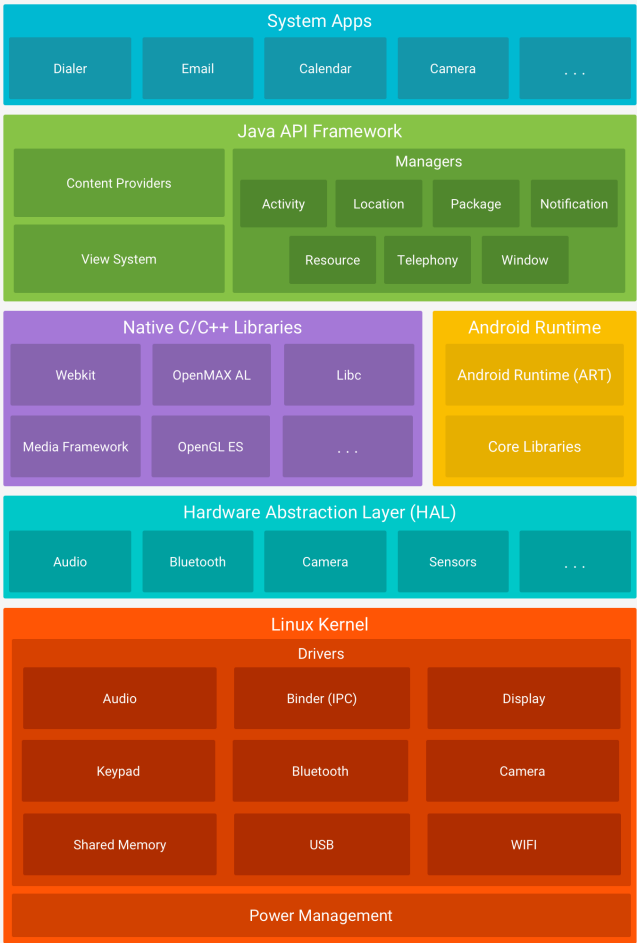


그림 1. Android 소프트웨어 스택

System Apps / User Apps ← 우리가 분석하는 APK

Java API Framework ← Activity, Intent, Permission, ContentProvider

Native C/C++ Libraries + Android Runtime ← libc, media, crypto, ART

HAL ← 카메라, 블루투스, 센서 같은 하드웨어 추상화

Linux Kernel ← process, memory, driver, SELinux

출처: Android Developers Platform Architecture

APK는 어떤 파일인가

```
app.apk // zip archive
├─ AndroidManifest.xml
├─ classes.dex
├─ resources.arsc
├─ res/
├─ lib/arm64-v8a/*.so
└─ META-INF/
```

분석의 시작점

Manifest에서 진입점을 보고, classes.dex에서
Java/Kotlin 로직을 보고, lib/에서 native 코드를 봅니다.

정적 분석: 실행하지 않고 앱 읽기

```
$ adb shell pm path com.example.app
$ adb pull /data/app/.../base.apk
$ jadx-gui base.apk
```

Search:

```
exported="true"
http://
secret / key / password
```

코드와 설정을 먼저 읽습니다.

jadx로 Manifest, Java/Kotlin 코드, 문자열을 보면서 공격 진입점과 하드코딩된 값을 찾습니다.

ADB

JADX

Manifest

동적 분석: 실행 중인 앱 관찰하기

```
$ frida -U -f com.example.app -l hook.js

Java.perform(() => {
  Java.use("com.example.Login")
    .checkPassword.implementation = function(p) {
      console.log("password:", p);
      return true;
    };
});
```

런타임 값을 직접 확인합니다.

입력값을 보고, 반환값을 바꾸고, “항상 통과” 같은 동작을 실행할 수 있습니다.

공격 패턴: Intent Redirection

A 앱

“이 화면 좀 띄워줘” 요청

B 앱

받은 요청을 검증 없이 실행

공격자 앱

B 앱 내부 화면을 열게 만듦

권한 없는 앱이 **권한 있는 앱의 권한**을 빌려쓰는 셈입니다.

CVE-2024-26131: Intent Redirection 예시

1

exported Activity가 외부 Intent를 받음

```
exported="true"
```

2

extra 안의 내부 Intent를 꺼냄

```
getParcelableExtra()
```

3

검증 없이 startActivity 호출

```
startActivity(inner)
```

4

공격자가 내부 화면으로 redirect

```
CVE-2024-26131
```

추상 패턴 하나가 실제 CVE로 이어지는 흐름을 보여주는 사례입니다.

이런 분석의 결과는 뭔가

Bug

권한/검증/저장/통신 문제 발견

PoC

재현 절차, 공격 조건, 영향 범위 정리

CVE

공개 취약점 번호와 패치로 이어질 수 있음

핵심은 “뚫었다”에서 끝내는 게 아니라, 어떤 조건에서 어떤 영향이 나는지 정확히 증명하는 것입니다.

웹앱 / 하이브리드 앱도 분석 대상입니다

WebView

WebView 안에 웹앱이 들어가면 XSS, JS Bridge, URL Scheme이 앱 권한과 만납니다.

API

모바일 앱은 결국 서버 API와 통신합니다.

Frontend Mindset

사용자 흐름, 상태, 라우팅, 입력 검증 이해가 그대로 먹힙니다.

Job

모바일 보안, 앱 진단, 차량 인포테인먼트, 핀테크/커머스 보안

오늘 30분 안에 시작하기

1

테스트 APK 하나 받고 jadx-gui로 열기

2

AndroidManifest에서 exported activity 찾기

3

"http://", "secret", "password" 검색하기

이번 주 목표: Frida로 함수 하나 hook해서 로그 찍기.

AndroidManifest.xml은 앱의 명세서입니다

```
<activity  
  android:name=".DeepLinkActivity"  
  android:exported="true">  
  <intent-filter>  
    <data android:scheme="myapp" />  
  </intent-filter>  
</activity>
```

여기서 공격 진입점을 찾습니다.

exported activity, service, receiver, provider,
permission, deeplink, file provider 설정을 먼저 봅니
다.

jadx에서는 이렇게 추적합니다

1. Entry

Manifest에서 exported 컴포넌트
확인

2. Search

intent, uri, token, password,
api_key 검색

3. Trace

입력값이 어디로 흘러가는지 따라가기

앱 전체를 다 읽는 게 아니라, 외부 입력이 권한 있는 동작으로 이어지는 경로를 찾습니다.

앱은 데이터를 어디에 저장하나

로컬 저장소

SharedPreferences, SQLite, files/, cache/에
토큰과 개인정보가 남을 수 있습니다.

외부 저장소

권한이나 경로 설정이 약하면 다른 앱이 민감 파일에
접근할 수 있습니다.

```
$ adb shell run-as com.example.app
$ ls /data/data/com.example.app/
$ sqlite3 databases/app.db
```

```
PS C:\Users\... frida -l encrypt.js -U -F
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 ... (id=9e4f11ca)
]-->
***** Request *****
{"request":{"id":"...", "mobileNumber":"81...", "dateTime":"2...", "name":"...", "header":{"id":"..."}}}
***** Response *****
{"id":"...", "arn":"...", "errorMsg":"OK", "responseCode":"00", "id":"..."}
***** Response *****
{"id":"...", "arn":"...", "errorMsg":"OK", "responseCode":"00", "id":"..."}
***** Response *****
{"id":"...", "arn":"...", "errorMsg":"OK", "responseCode":"00", "id":"..."}
Setting the Number to : 81 8
***** Request *****
{"request":{"id":"...", "mobileNumber":"81...", "dateTime":"2...", "name":"...", "header":{"id":"..."}}}
***** Response *****
{"id":"...", "arn":"...", "errorMsg":"OK", "responseCode":"00", "activeCardsList":[{"CardNumber":"81...", "No":"..."}]}
```

모바일 앱은 결국 API 클라이언트입니다

Burp로 잡는 것

로그인, refresh token, 결제, 쿠폰, 사용자 정보, 권한 체크 API.

자주 나오는 취약점

IDOR, 인증 누락, 과도한 정보 노출, TLS 검증 우회, debug endpoint.

웹 보안 지식이 모바일 앱 분석에서 그대로 재사용됩니다.

WebView는 웹과 앱 권한이 만나는 곳

```
webView.settings.javaScriptEnabled = true  
webView.addJavascriptInterface(  
    Bridge(), "Android"  
)
```

위험한 조합

외부 URL 로드, JavaScript bridge, 약한 origin
검증이 합쳐지면 웹 취약점이 앱 권한으로 번질 수 있습니다.

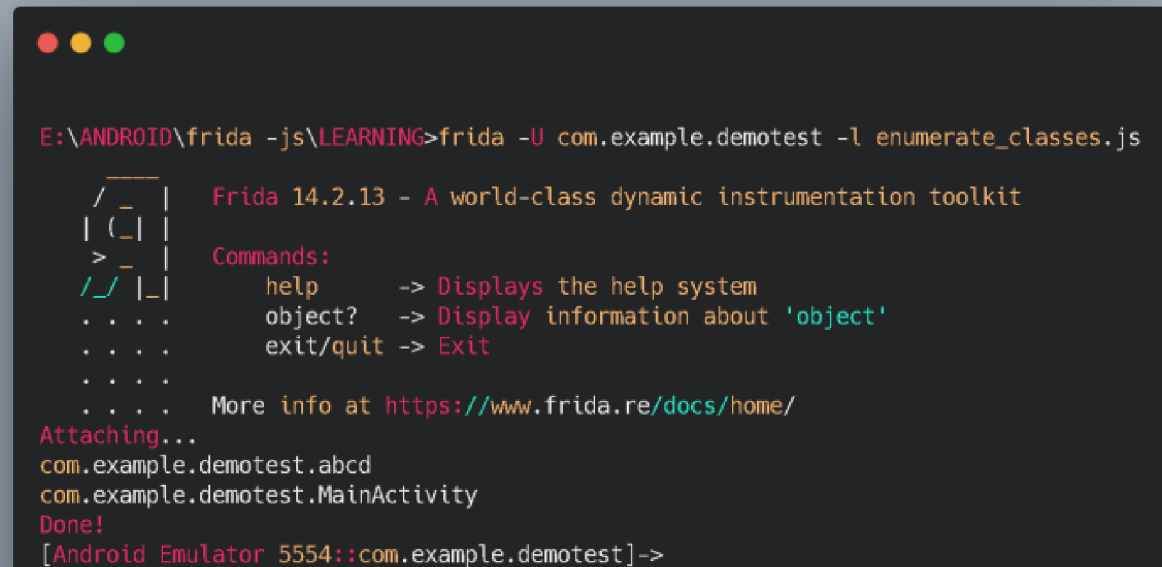


Frida 실습은 이렇게 시작합니다

```
$ adb devices
$ adb push frida-server /data/local/tmp
$ adb shell chmod +x /data/local/tmp/frida-server
$ frida-ps -U
```

정적 분석에서 동적 분석으로

코드에서 의심 지점을 찾고, Frida로 실제 런타임 값과 분기 결과를 확인합니다.



```
E:\ANDROID\frida -js\LEARNING>frida -U com.example.demotest -l enumerate_classes.js

/ _ |
| ( |
> _ |
/_/ |
...
...
...
...
More info at https://www.frida.re/docs/home/

Attaching...
com.example.demotest.abcd
com.example.demotest.MainActivity
Done!
[Android Emulator 5554::com.example.demotest]->
```

계속 공부하려면 무엇을 보면 되나

Tools

jadx, apktool, Frida, Objection, Burp, MobSF.

Targets

테스트 APK, CTF 앱, WebView 앱, IoT 연동 앱, 차량 인포테인먼트 앱.

Keywords

exported, deeplink, WebView bridge, token storage, API authorization.

보고서 얘기보다 지금은 “다음에 무엇을 잡고 공부하면 되는지”가 더 중요합니다.

여러분 손에 든 그 컴퓨터가 첫 분석 대상입니다.

앱 하나를 열어보면
공격 표면이 보이기 시작합니다.

