**안드로이드 쉘 스크립트를 통한 악성 앱 덤프 및 분석**

**텍스트, 디자인이(가) 표시된 사진

자동 생성된 설명**

TEAM : 케이쉴드 주니어 11기 침해사고대응반 3팀

MEMBER : 이원희, 김정완, 윤민철, 최서연, 최시온, 하주현

**목차**

[**1.** **개요** 3](#_Toc148051263)

[a. 프로젝트 배경 3](#_Toc148051264)

[b. 프로젝트 목적 3](#_Toc148051265)

[c. 기대효과 3](#_Toc148051266)

[d. 안드로이드 쉘 스크립트 설명 3](#_Toc148051267)

[**2.** **스크립트 준비** 7](#_Toc148051268)

[a. 가상기기(안드로이드 스튜디오 환경)에서 실행한 과정 및 결과 7](#_Toc148051269)

[b. 가상기기(NOX환경)에서 실행한 과정 및 결과 18](#_Toc148051270)

[**3.** **실제 기기 수행** 19](#_Toc148051271)

[a. AhMyth RAT (악성 앱) 소개 19](#_Toc148051272)

[b. AhMyth RAT (악성 앱) 설치 20](#_Toc148051273)

[c. AhMyth RAT 기능 소개 22](#_Toc148051274)

[**4.** **스크립트 실행 결과** 34](#_Toc148051275)

[a. [ Active Data ] 34](#_Toc148051276)

[b. [ InActive Data ] 46](#_Toc148051277)

[**5.** **결론** 58](#_Toc148051278)

[a. 실제 환경에서 스크립트 활용 가능성 58](#_Toc148051279)

[b. 시사점 60](#_Toc148051280)

## **개요**

## 프로젝트 배경

Windows와 Linux의 데이터 수집용 자동화 스크립트는 이미 개발되어 있으나, 우리가 매일 사용하는 스마트폰의 데이터를 자동으로 수집할 수 있는 스크립트는 찾기 어려웠다. 악성 앱을 이용한 스마트폰 해킹에 대한 사례가 증가하면서 악성 소프트웨어에 감염된 스마트폰의 데이터를 수집할 수 있으면 분석하기 편리할 것이란 생각에 프로젝트를 선정했다.

## 프로젝트 목적

안드로이드 쉘 스크립트를 작성하여 악성 앱이 설치된 실제 스마트폰에서 데이터 덤프를 얻고, 악성 소프트웨어의 탐지 및 행위를 분석한다.

## 기대효과

1. 안드로이드 시스템의 데이터 수집과 분석에 대한 기초지식 제공

2. 사용자의 안드로이드 데이터 수집의 업무효율성 증진

3. 향후 라이브 포렌식 도구 개발에 활용 가능

## 안드로이드 쉘 스크립트 설명

작성한 쉘 스크립트는 총 3개로 구성되어 있다.

1. Android\_main.sh – 활성데이터/비활성데이터 수집 스크립트 중 선택 실행
2. Android\_Active.sh – 활성데이터 수집
3. Android\_Inactive.sh – 비활성데이터 수집

**[ Main 스크립트 실행 화면 ]**



**[ Active 스크립트 실행 화면 ]**

텍스트, 스크린샷, 폰트, 포스터이(가) 표시된 사진

자동 생성된 설명

**[ Inactive 스크립트 실행 화면 ]**

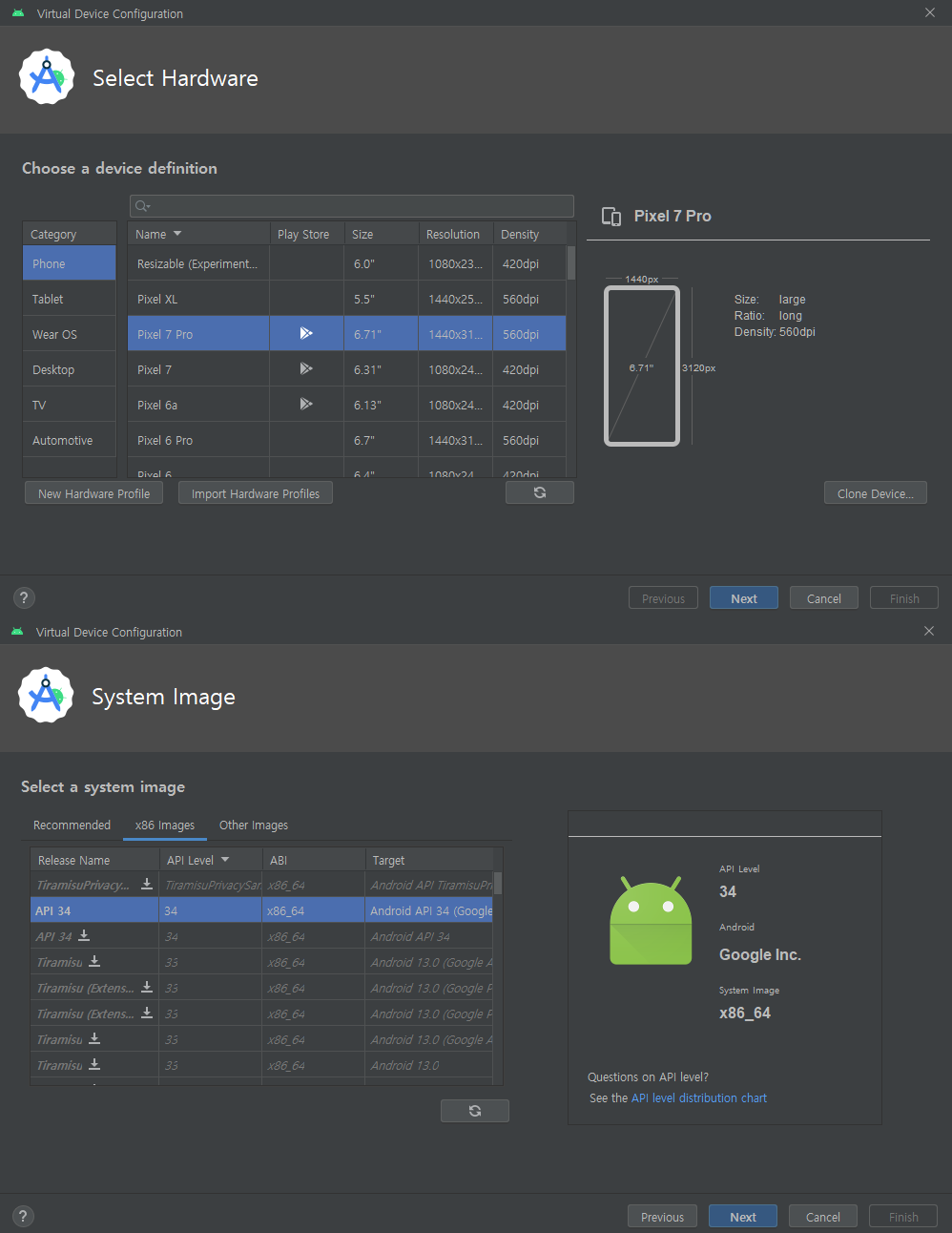


* 스크립트 제작은 안드로이드 스튜디오 및 NOX와 같은 가상 환경에서 제작했으며, 원활한 진행을 위하여 루팅을 한 상태에서 여러가지 명령을 테스트하며 제작하였다.
* 스크립트는 안드로이드 시스템에 대하여 활성 데이터/비활성 데이터를 수집할 수 있도록 수정했으며, “활성 데이터 스크립트”는 가상메모리, 네트워크 정보, 프로세스 정보, 로그온사용자 정보, 시스템 정보, 자동실행 항목, 클립보드를 수집한다. “비활성데이터 스크립트”는 파일 시스템 메타데이터, 시스템, 사용자 애플리케이션 설정 정보, 로그 파일, 휴지통, 브라우저 정보, 임시 파일, 외장 저장소 정보를 수집할 수 있도록 구성하였다.
* 스크립트는 /system/bin/sh와 함께 toybox를 활용하여 작성되었다. 프로젝트에서 사용한 가상환경과 스마트폰은 toybox 기반의 유틸리티를 제공하는데, toybox의 유틸리티 범위가 busybox보다는 제한적이다. 그러므로 toybox를 기반으로 스크립트를 작성하면 busybox 환경에서도 스크립트가 비슷한 방식으로 동작할 것이다.

## **스크립트 준비**

## 가상기기(안드로이드 스튜디오 환경)에서 실행한 과정 및 결과

# adb 프로그램이 환경변수로 설정되어 있다는 가정 하에 수행하였다.



Pixel 7 Pro, Android API 34 x86\_64 이미지를 선택하여 가상 환경을 생성했다.

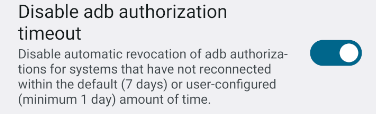
안드로이드 가상 머신 실행 후



Settings -> About emulated device -> Build number (5번 연속 터치한다)

위의 과정을 거치면 Developer options 활성화된다.





USB 디버깅을 지원하는 옵션과 adb 인증 타임아웃을 비활성화 하는 옵션을 활성화시키고

다음의 작업을 통해 adb shell에 접근한다.

adb devices를 통해 호스트에 가상 기기가 탐지되는지 확인한다.



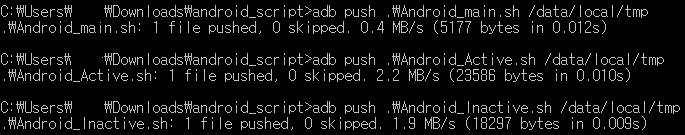
adb root 명령을 통해 루트 권한 부여한다.

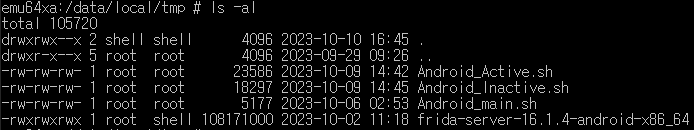


adb shell을 통하여 해당 가상머신으로 쉘을 접근한다.



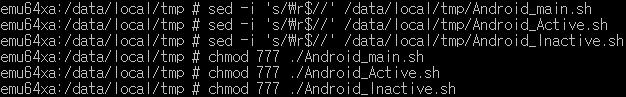
스크립트 작성하는 것 자체는 안드로이드가 아닌 바깥 호스트에서 작업을 했으며, 제작한 호스트를 테스트할 때 마다 adb push 명령을 통해 안드로이드 가상 기기에 파일을 전송했다.





[ 스크립트가 잘 push된 모습 ]

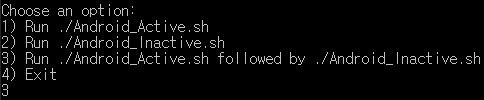
CRLF가 아닌 LF에서의 개행 문자를 처리하기 위해서 명령을 입력 및 권한 부여한다. 제작한 문서 파일 형식은 LF이다.



준비한 쉘 스크립트를 수행한다. 가급적이면 Android\_main.sh 스크립트를 실행할 것을 권장하는데, Android\_Active.sh 및 Android\_Inactive.sh 개별로 실행하면 각 스크립트를 수행한 결과물을 저장하기 위한 타임스탬프로 이루어진 명칭의 루트 디렉토리가 개별로 생성되기 때문이다.

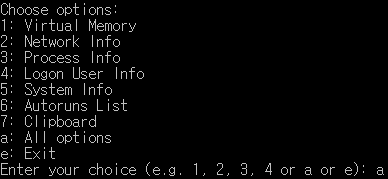


main 쉘 스크립트 수행 후 Active script, InActive script 혹은 둘 다 실행할지 선택해야 한다.

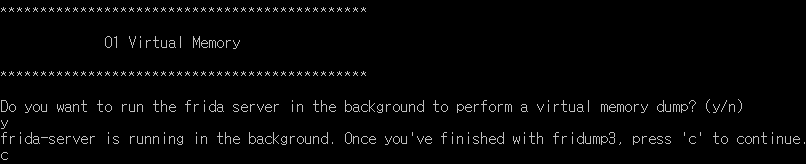


[ 예시로 3번을 선택함으로써 둘 다 선택하도록 함 ]

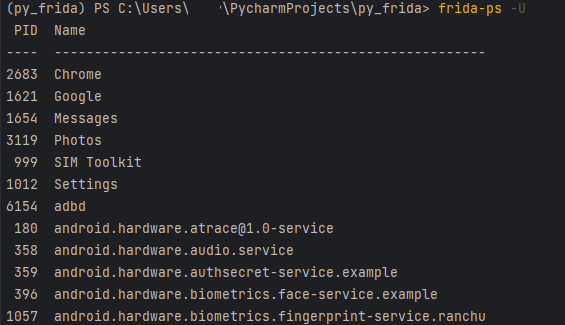
Active 스크립트를 수행했을 시 아래와 같이 어떤 옵션을 수행할지 묻는다.



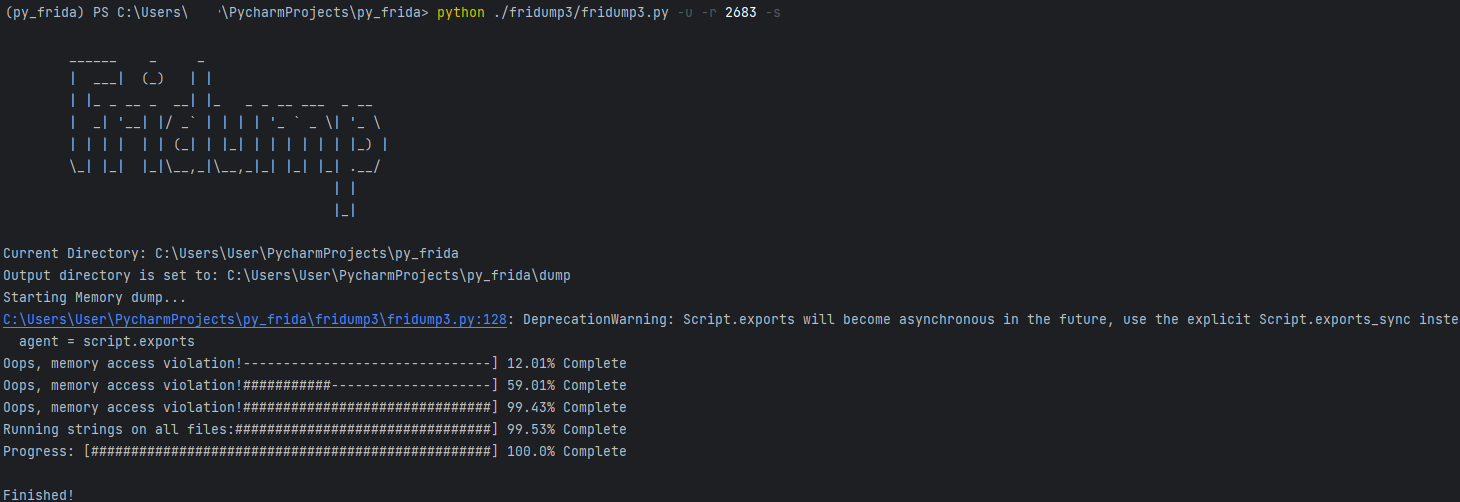
Active Script의 1단계에서 frida 서버를 백그라운드로 실행하여 대상 프로세스에 대한 가상 메모리 덤프를 뜨도록 했다. Frida 서버는 현재 x86\_64이어서 해당 아키텍처에 맞게 미리 준비한다.



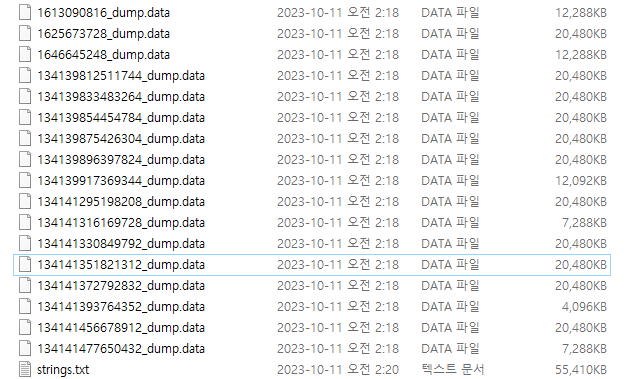
프리다에서 프로세스 목록이 무엇인지 탐지한다.



덤프 뜨고자 하는 프로세스의 PID를 지정하여 덤프 뜨는 절차를 수행한다.



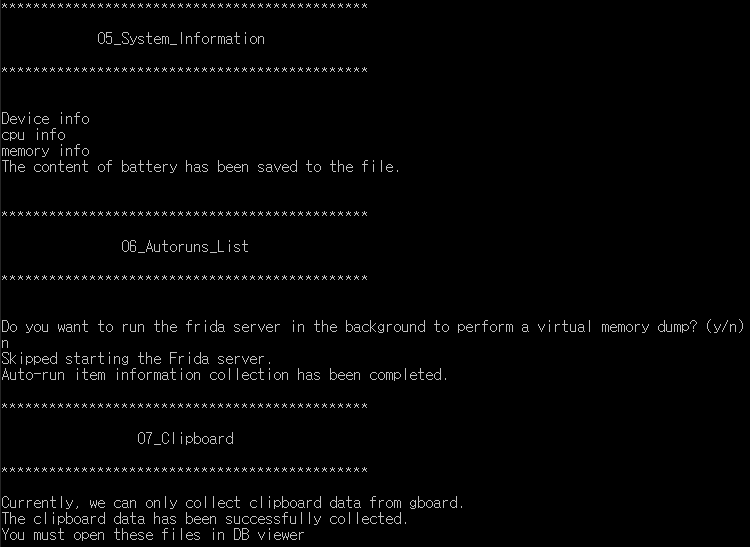
만들어진 결과물은 다음과 같이 ~~\_dump.data 파일과 해당 파일에서 의미 있는 문자열들만 출력한 strings.txt 파일이 있다.



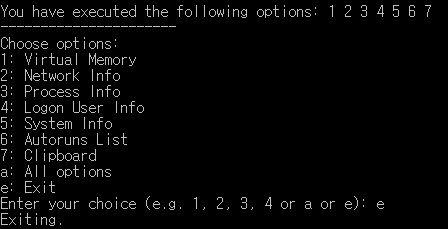
Active Script의 2, 3, 4단계에서 네트워크, 프로세스 정보와 로그온 사용자 정보를 수집하며, 특히 3, 4단계에서 어떤 프로세스 혹은 패키지를 수집해야 할지 지정해야 하기 때문에 ps –A 명령을 수행하여 전체 프로세스 리스트를 나열하도록 하였다.



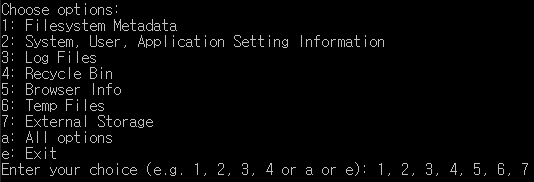
Active Script의 5, 6, 7 단계를 통해 시스템 정보, 자동 실행 항목, 클립 보드를 정보를 수집할 수 있으며, 이때 6단계에서 프리다 서버를 실행하도록 코드를 구성했는데 bixby routine과 같은 특정 앱에서 제공되는 자동실행 기능을 제공하고 있는 걸 덤프하기 위해서이다.



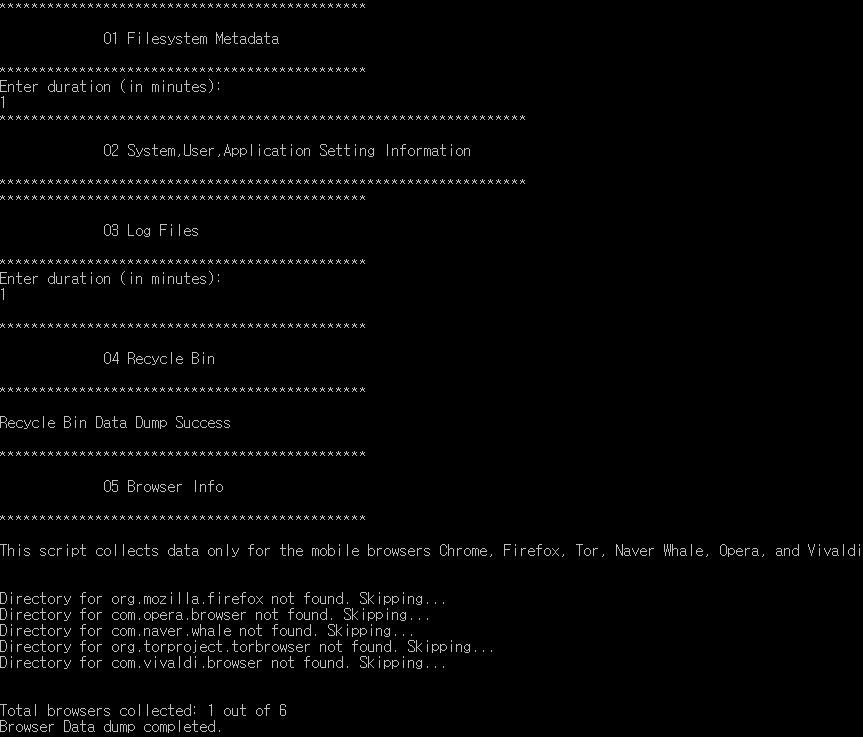
Active Script가 전부 수행되었는지 확인하기 위해서 지금까지 수행한 단계들을 오름차순으로 나열되게 스크립트를 작성했으며, e를 눌러서 Active Script를 빠져 나옴과 동시에 InActive Script를 수행한다.



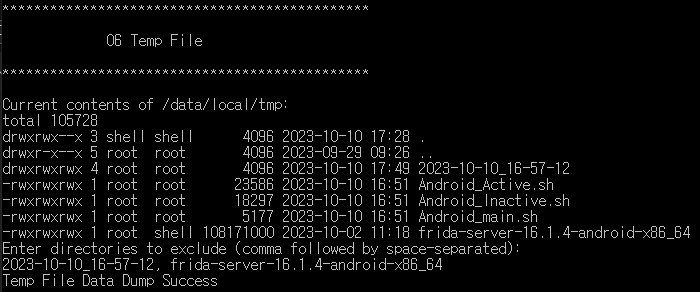
Inactive Script 또한 Active Script에서 나타냈던 바와 같이 똑같은 인터페이스를 제공한다.



Inactive Script는 1단계와 3단계에서 얼마동안 로그를 수집할지 지정해야 하기 때문에 따로 시간을 입력 받도록 구성했고 브라우저 정보를 수집하는 5단계에서는 만약 대상 시스템에 찾고자 하는 브라우저가 애초에 설치되지 않았으면 에러를 띄우게 만들고 존재하는 것만 수집하도록 구성하였다.

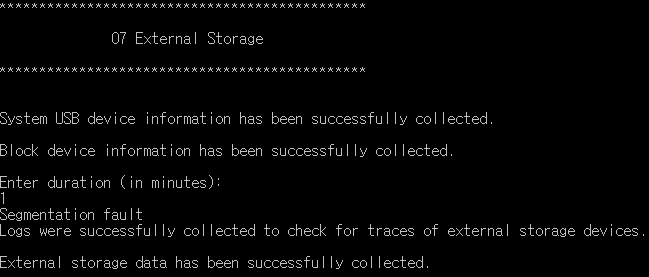


6번의 임시파일 고르기에서 현재 스크립트 파일이 존재하는 위치가 임시 디렉토리인 /data/local/tmp 경로여서 해당 파일들은 제외하도록 구성했는데 그 외에 스크립트가 수행됨으로써 생성되는 결과 디렉토리 및 프리다 서버는 굳이 수집할 필요는 없기에 아래의 사진과 같이 제외하였다.

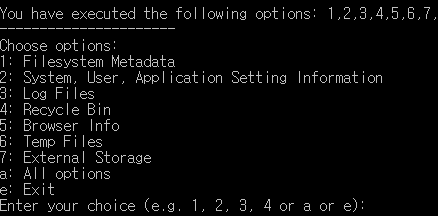


7번은 외장 저장소를 수집하는 단계다.

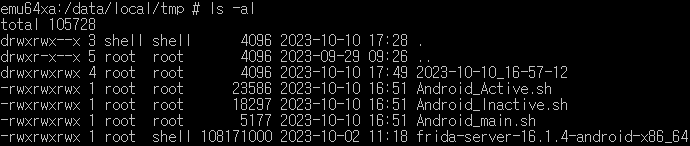
안드로이드 스튜디오 에서는 기본적으로 SD card가 생성되어 연결된 상태로 가상머신이 생성된다.



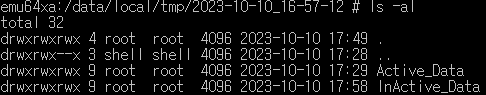
Inactive Script도 Active Script와 마찬가지로 지금까지 어떤 옵션을 수행했는지 확인할 수 있게 구성했다.



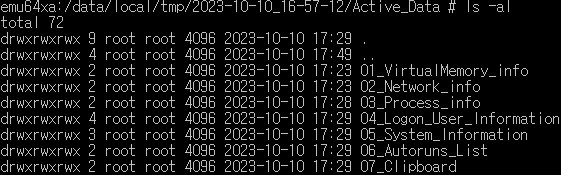
수행한 스크립트 결과물은 아래와 같으며, 결과물 디렉토리는 타임스탬프 명칭으로 생성된다.



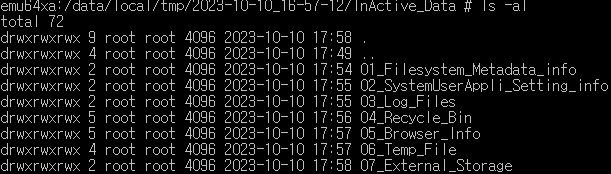
그 하위에 각 Active/Inactive 스크립트 각각에 대하여 디렉토리가 생성된다.



Active Script의 개별 단계에 대하여 다시 디렉토리가 생성된 것을 볼 수 있다.



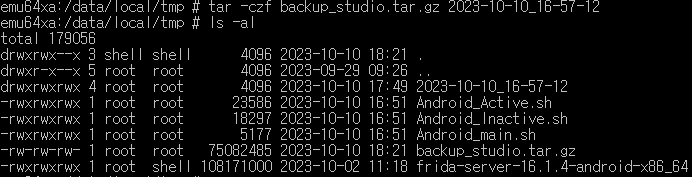
InActive Script 또한 개별 단계에 대하여 디렉토리가 생성된 것을 볼 수 있다.



예시로 Inactive Script의 1단계 결과물을 살펴본다.



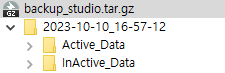
생성된 데이터를 추출하는데, 추출하기 전에 먼저 tar로 압축한다.



adb pull 명령을 이용하여 추출한다.

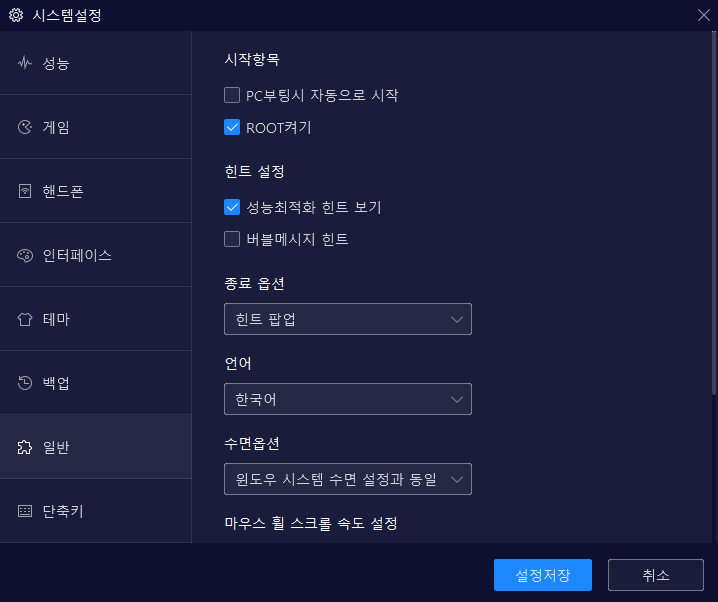


덤프 결과물 수집

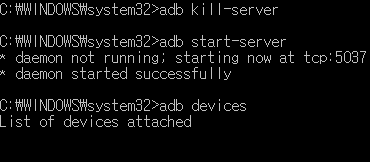


## 가상기기(NOX환경)에서 실행한 과정 및 결과

Nox에서 루팅한다. (시스템 설정 -> “ROOT 켜기” 버튼 활성화)



adb kill-server, adb start-server 후 adb devices 실행한다.



adb connect 127.0.0.1:62001 수행한다.



adb shell로 접속하면 현재 root인 것을 확인할 수 있다.

이후의 쉘 스크립트 push, pull 및 수행 과정은 안드로이드 스튜디오에서 묘사한 것과 동일하다.



## **실제 기기 수행**

## AhMyth RAT (악성 앱) 소개

AhMyth RAT은 TCP와 HTTP를 통해 휴대폰과 처음 연결한 후, WebSocket 프로토콜로 전환하여 C&C 서버와의 통신을 유지한다.

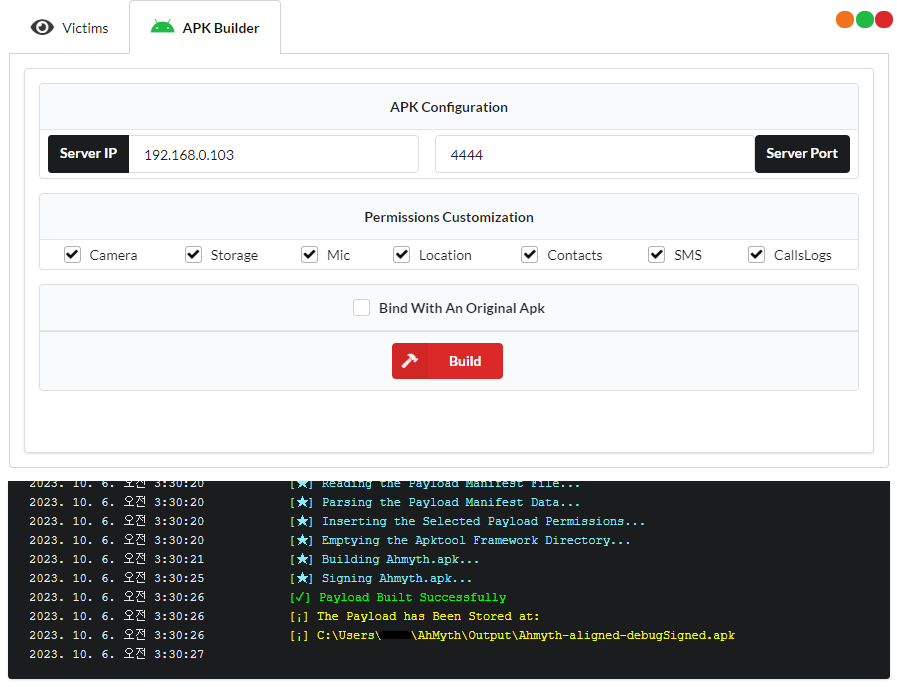
구체적으로, AhMyth RAT은 먼저 감염된 휴대폰과 TCP 연결을 설정한다. 그 후 통신을 위해 HTTP에서 WebSocket 프로토콜로 전환한다. 휴대폰으로부터의 첫 번째 HTTP 요청은 C&C 서버의 IO.sockets.on 함수에 의해 받아지며, 이 함수는 휴대폰 매개변수를 addVictim 함수로 파싱한다. 이 함수는 휴대폰 매개변수를 사전에 저장한다. 이후에 WebSocket 프로토콜을 위한 두 번째 연결이 설정되며, 통신 중에는 probe, upgrade, ping, pong과 같은 다양한 유형의 패킷이 감염된 휴대폰과 C&C 사이에 전송되어 연결을 유지한다​.

AhMyth RAT은 카메라 사진 캡처, 디렉토리 리스팅 및 다운로드, 녹음, GPS 위치 추적, 메시지 보내기 및 메시지 리스팅, 전화로그 리스팅, 연락처 리스팅 기능을 포함한다.

## AhMyth RAT (악성 앱) 설치

AhMyth RAT apk 파일을 직접 빌드하여 해당 앱을 실습 스마트폰에 설치한 후 서버로 victim 기기의 RAT을 통해 다양한 기능을 테스트 해보고 그러한 활동에 따라 생성되는 데이터들을 덤프하여 분석해본다.

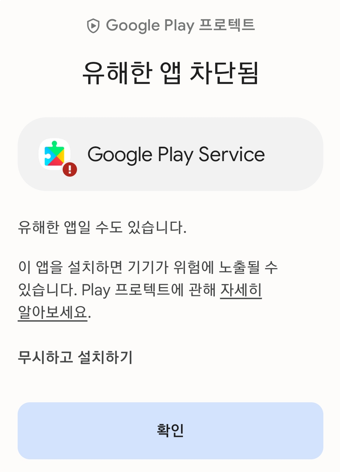
1. apk 빌드 한다.



[ 서버 주소/포트, 권한 설정 및 apk 생성 화면 ]

2. apk 설치한다.





앱 설치 시도 시 경고 출력한다.

“무시하고 설치하기” 눌러서 설치를 진행한다.

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명 텍스트, 스크린샷, 번호, 폰트이(가) 표시된 사진

자동 생성된 설명

[ 다양한 권한을 요구하는 화면 ]

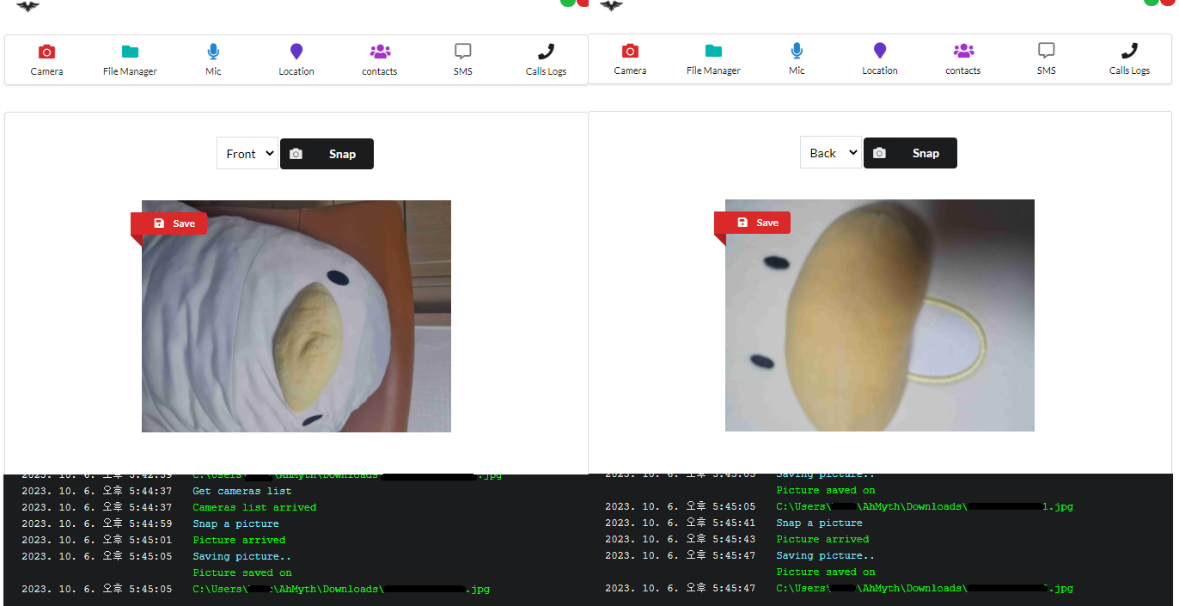
3. AhMyth RAT을 통한 victim 기기 접근 수행한다.

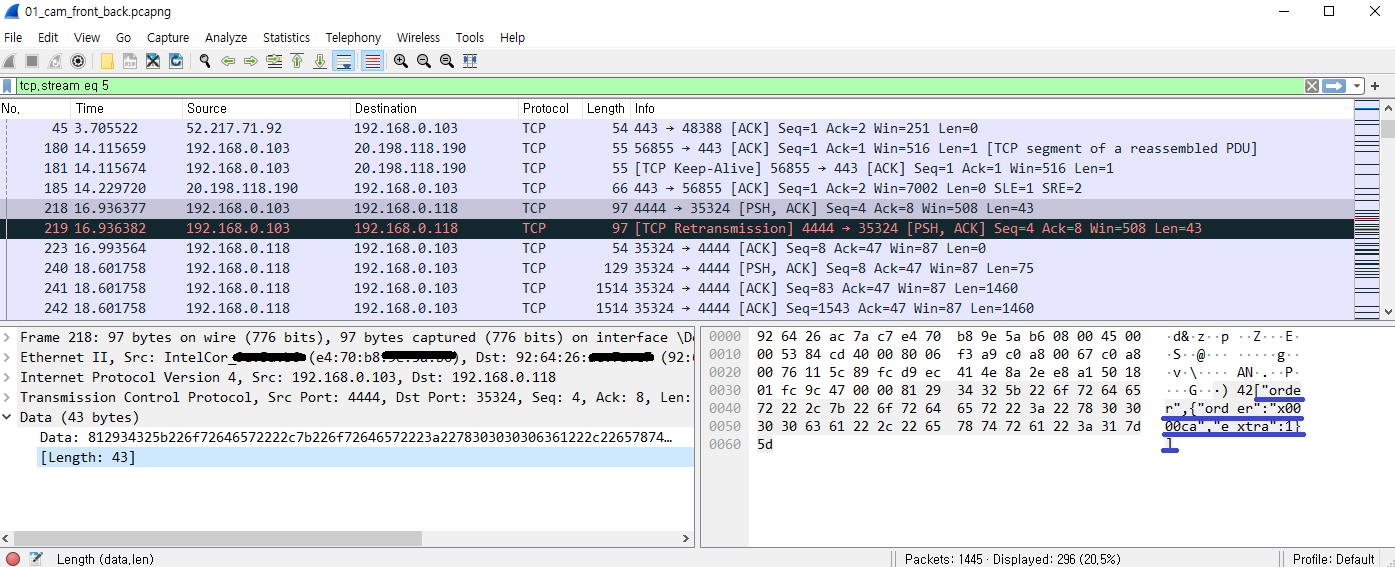
“New victim from (IP주소)” 출력됨으로써 victim 기기가 C&C 서버에 접속된 것을 확인했다.

## AhMyth RAT 기능 소개

**1. 카메라 제어**

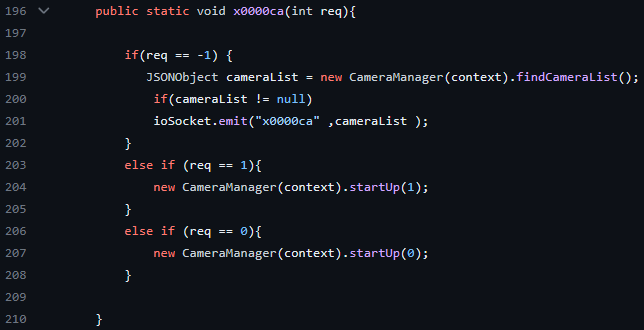
정면/후면 카메라를 통해 사진을 촬영하고 촬영한 이미지 파일을 저장한다.





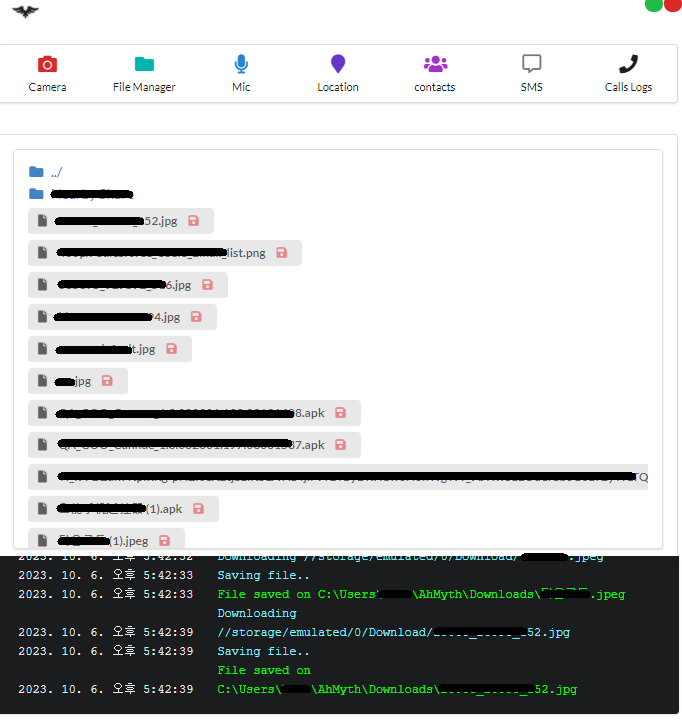
카메라 리스팅 하는 것을 확인할 수 있다.

42 [“order”, {ord er”:”x0000ca”,”e xtra” :1}] 이라고 적혀 있는 것을 볼 수 있는데 이것을 아래 나와 있는 깃허브의 소스코드 내용을 참고하여 x0000ca가 카메라 리스팅을 나타낸다는 것을 알 수 있다.

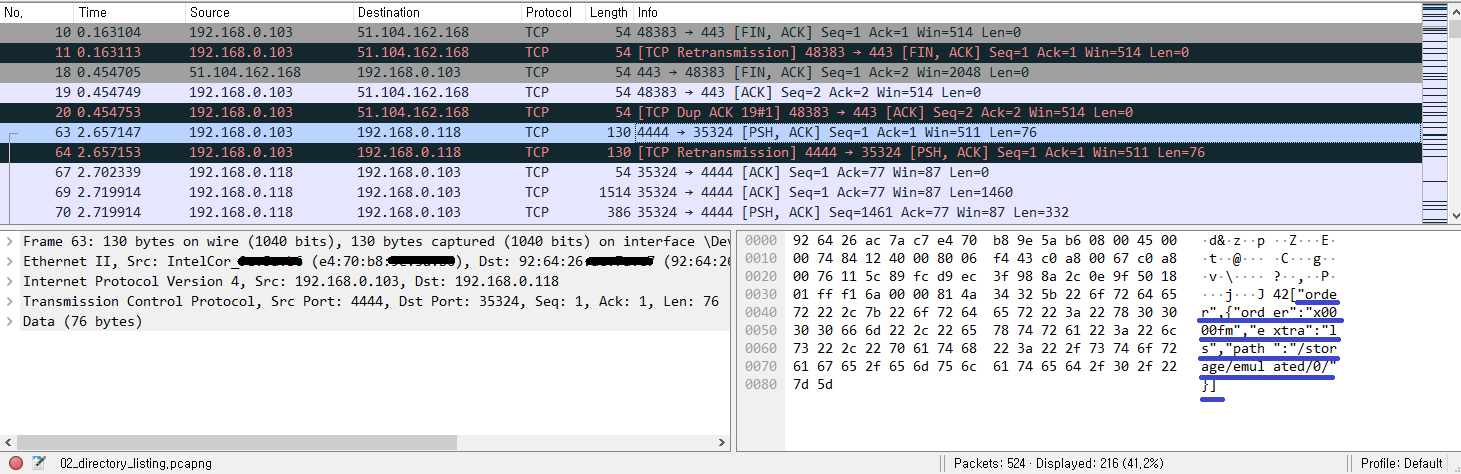


[ 카메라 리스팅 관련 소스 코드 ]

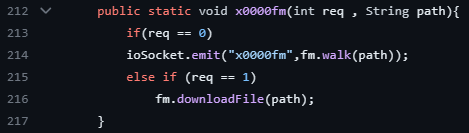
**2. 디렉토리 리스팅 및 다운로딩**



Victim 기기의 디렉토리를 리스팅 및 다운로드 받는다.

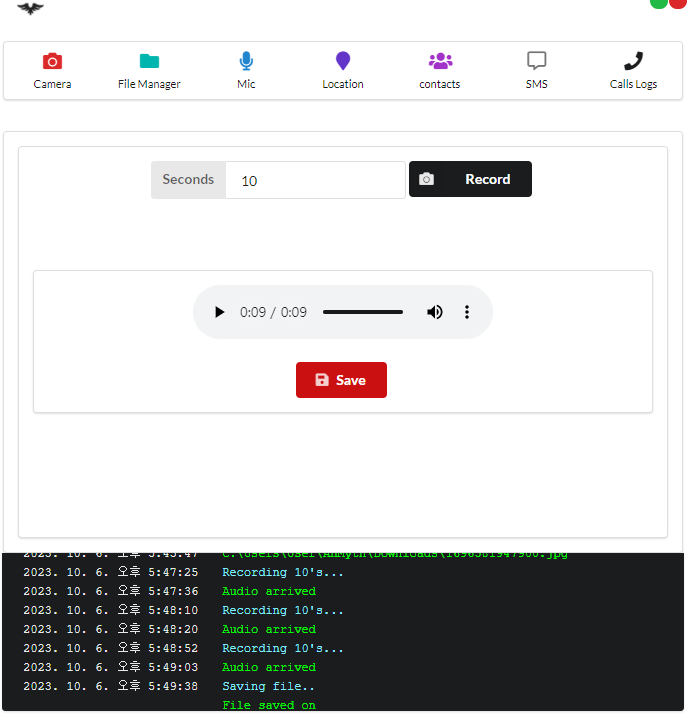


42[“order”, {ord er”:”x0000fm”,”e xtra”:”ls”,”path “:”/storage/emul ated/0/”}] 라는 문구는 아래 소스코드의 디렉토리 리스팅을 뜻하여 /storage/emulated/0/ 의 파일 항목을 출력하라는 내용이다.



[ 디렉토리 리스팅 깃허브 소스코드 ]

**3. 녹화**



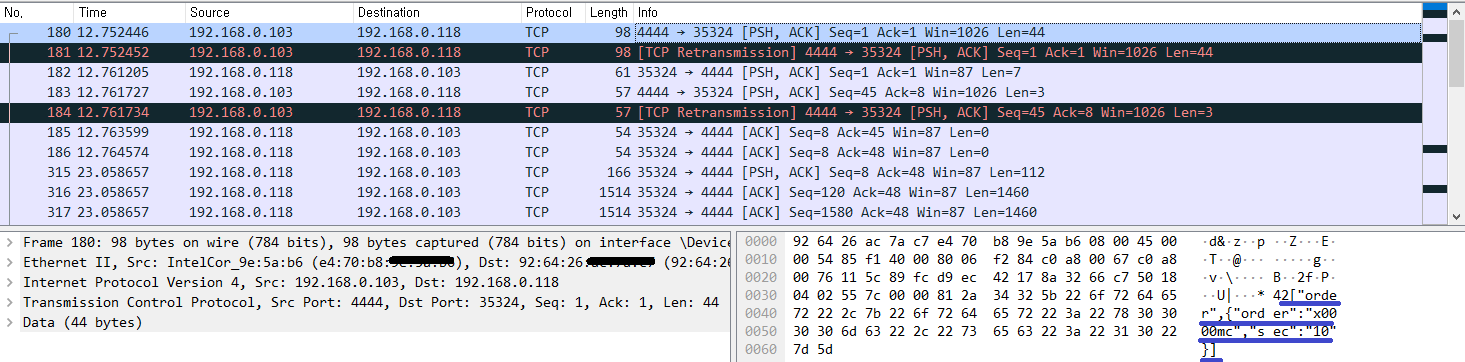
10초를 지정하여 그 시간 동안만 녹화를 수행한다.



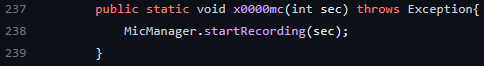


[ 녹화 활성화시 화면 변화 ]

녹화기능을 서버에서 활성화하면 victim 기기의 화면에서 녹화 한다고 알리는 녹색 마이크 버튼이 활성화된 것을 볼 수 있다.

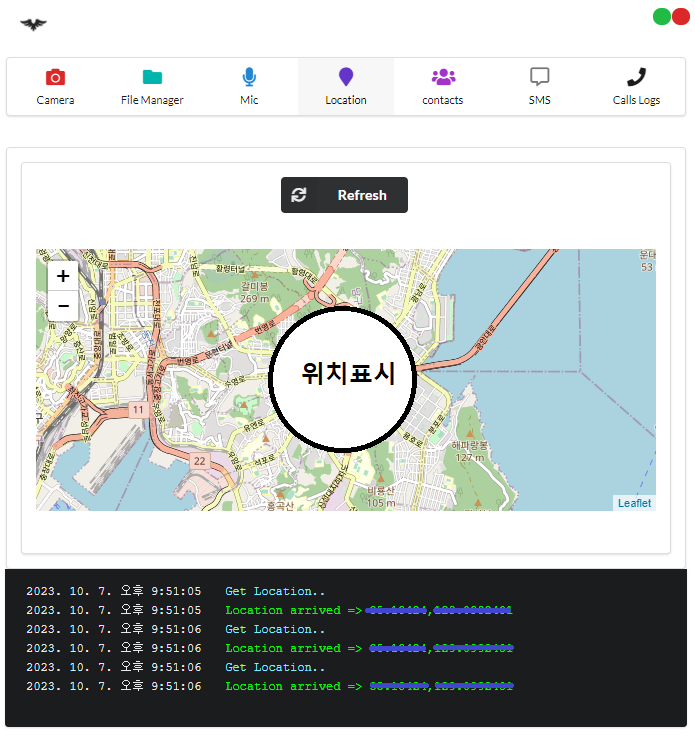


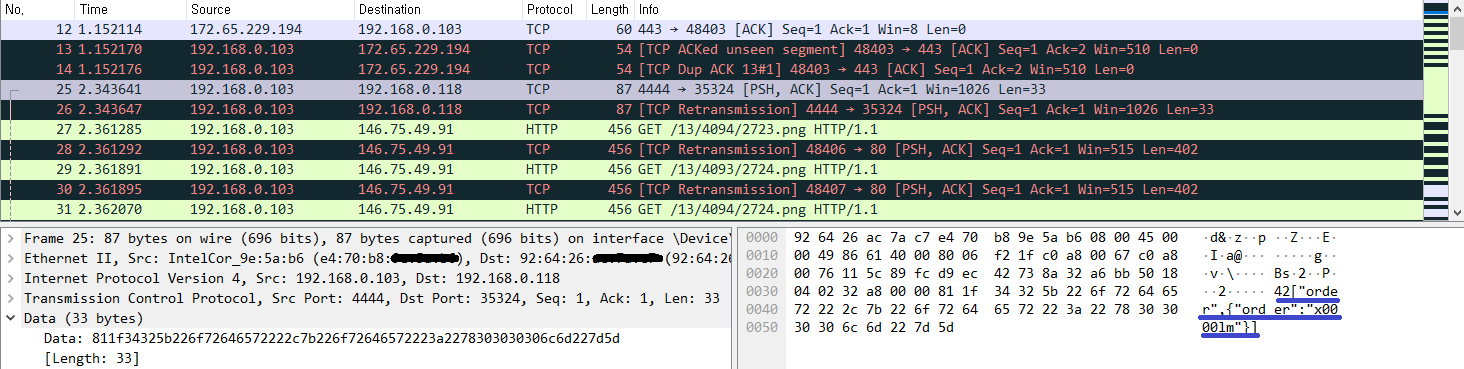
42[“order”,{“ord er”:”x0000mc”,”s ec”:”10”}] 이라고 나타나 있다. GUI 화면으로 서버에 10초 동안 녹화하겠다고 나타낸 것이 저렇게 출력되는 것으로 추정하며, x0000mc가 녹화라고 추정한 이유로 깃허브 소스코드를 근거로 볼 수 있다.



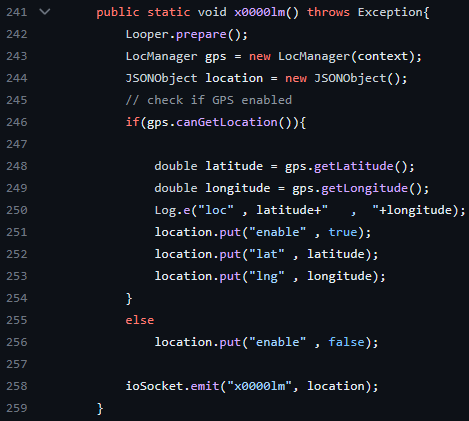
[ 깃허브 마이크 소스 코드 ]

**4. GPS 위치 추적**

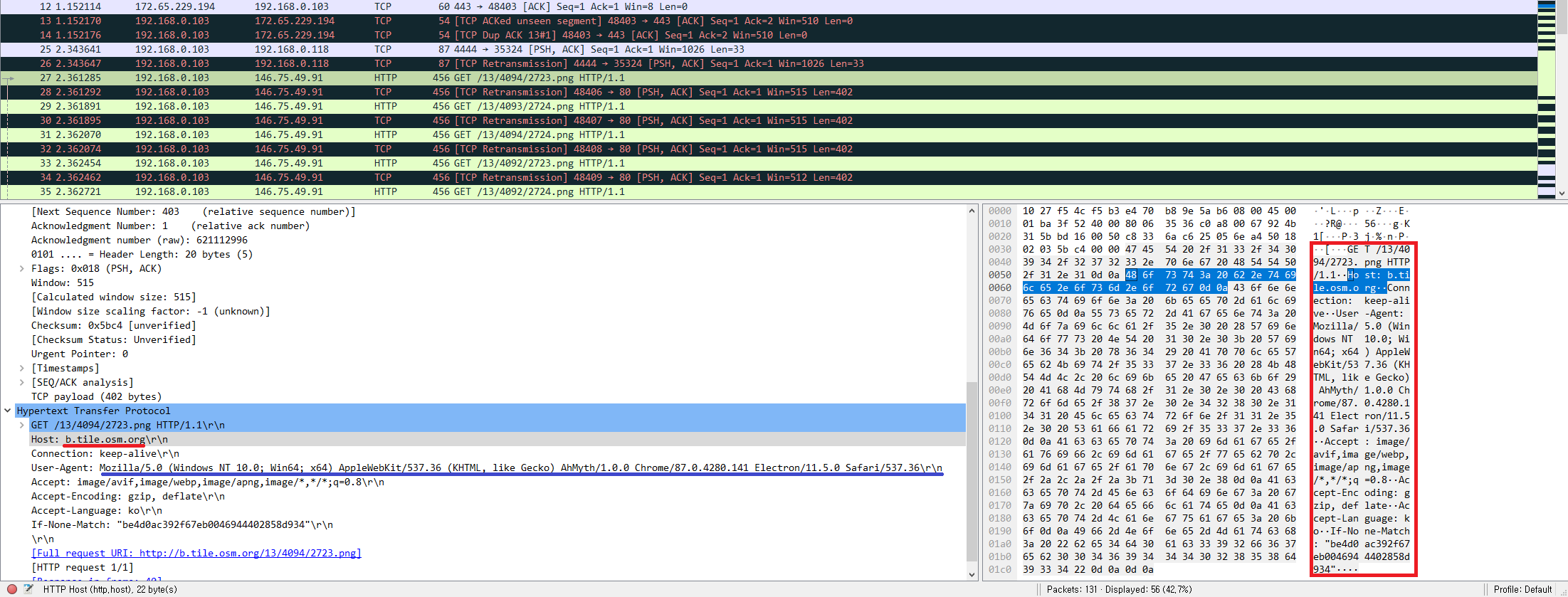




42[“order”,{“ord er”:”x0000lm”}]이 바로 GPS 위치 관련해서 C&C에서 전송하는 명령이다.



[ GPS location 관련 깃허브 소스코드 ]



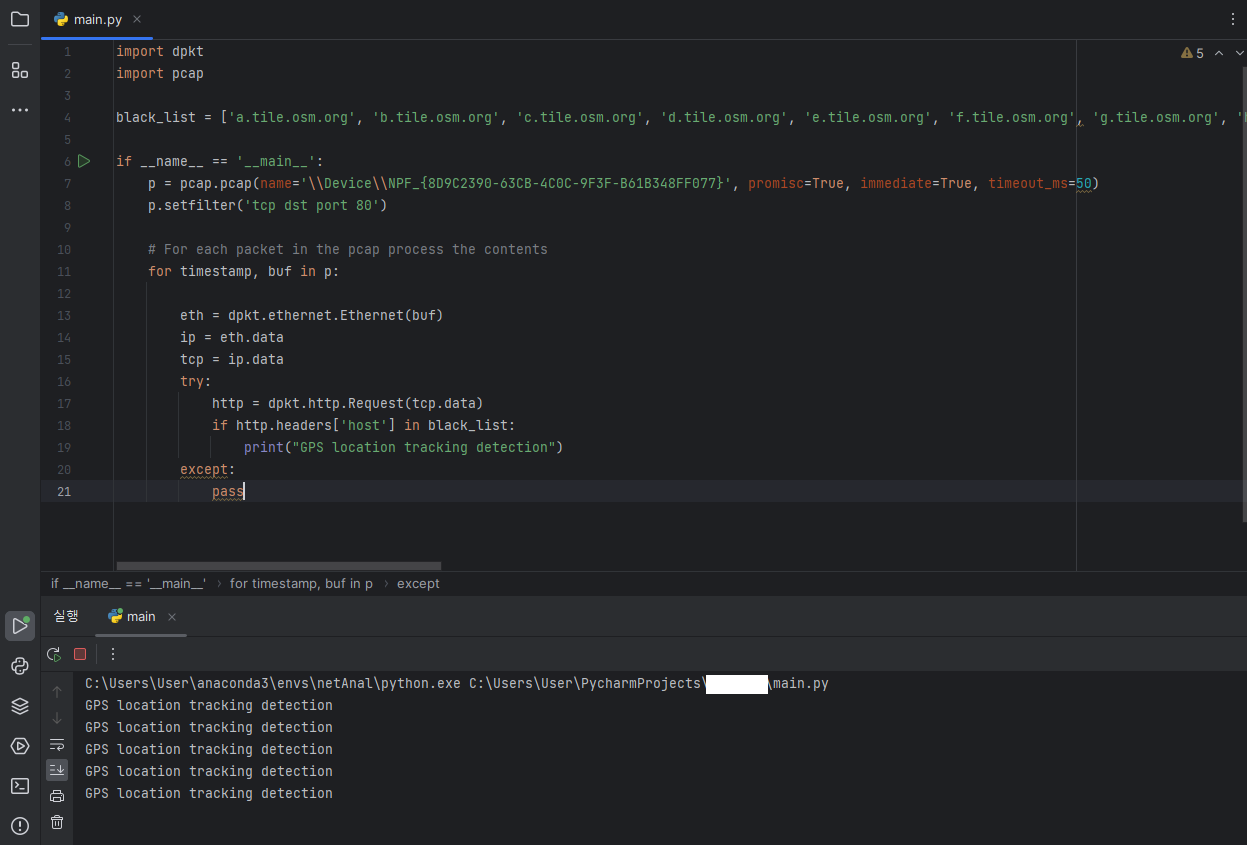
패킷을 분석해보면 b.tile.osm.org라는 사이트에서 GPS 추적했을 시 지도까지 같이 제공해주는 사이트가 있다는 것을 발견했고, 이를 발견한 위치를 여러 종류의 프로그램 혹은 브라우저에 버전 별로 제공해 주는 것을 확인했다.



해당 사이트에 접속해보면 위와 같이 나오는 것을 볼 수 있다.

비슷한 도메인 명의 사이트에서 GPS 위치 추적할 수 있도록 구성되어 있다는 것을 확인하여

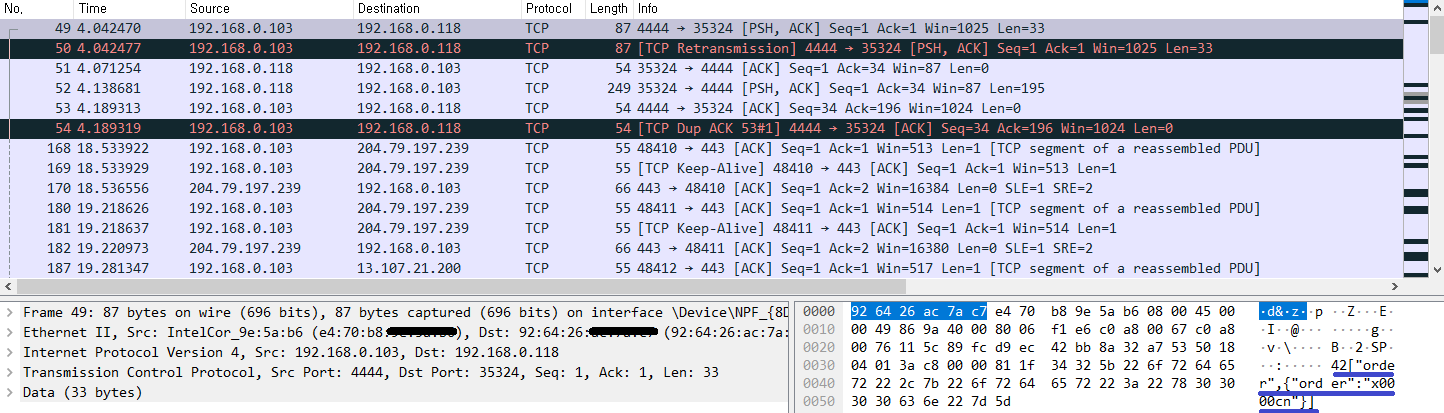
아래와 같이 파이썬 코드를 작성해서 RAT이 서버로 아까 알아낸 사이트를 통해 위치정보를 제공해줄 때 마다 GPS 추적하고 있다고 알릴 수 있게 작성했다.



**5. 연락처**



연락처를 리스팅한 내용을 csv파일로 생성한다.

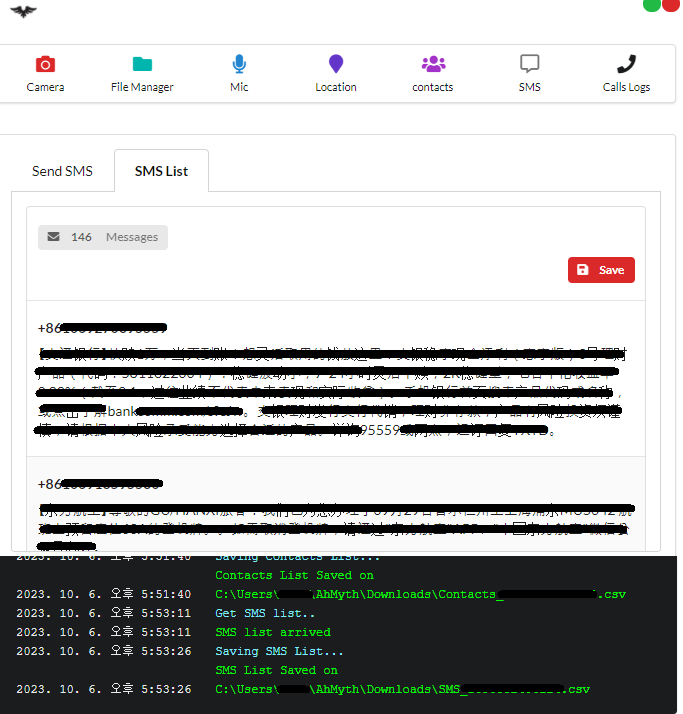


42[“order”,{“ord er”:”x0000cn”}] 이라고 적혀 있는데 깃허브 소스코드를 확인해보면 연락처 리스팅과 관련 있다는 것을 알 수 있다.

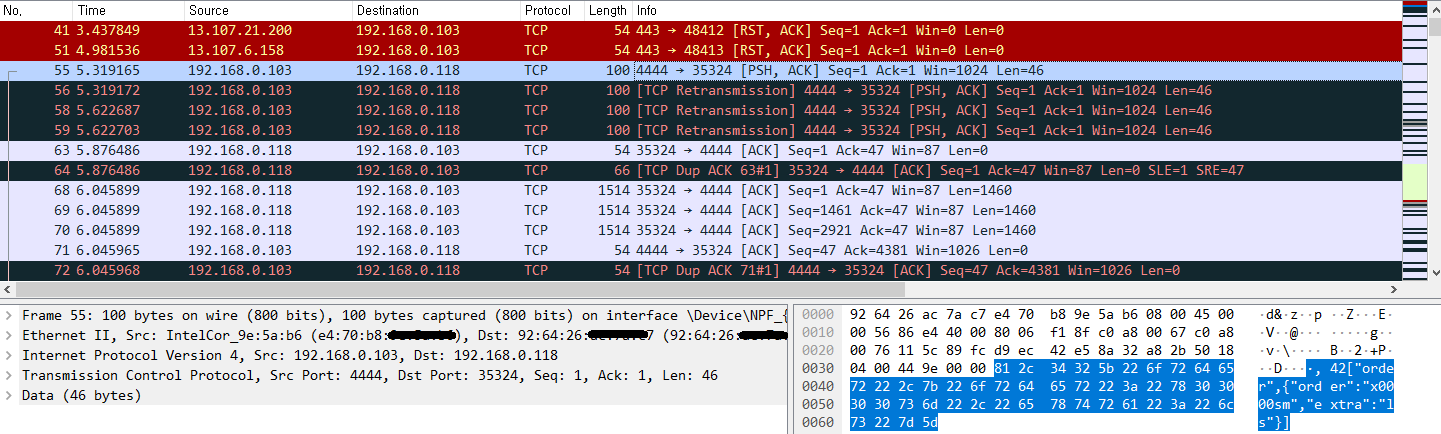


**6. sms**

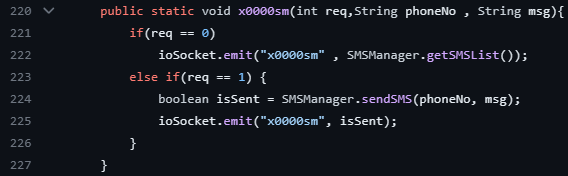
sms는 전송 기능과 리스팅 기능이 있는데 리스팅만 하였다.



SMS 리스팅 한 결과물을 csv 파일로 생성한다.

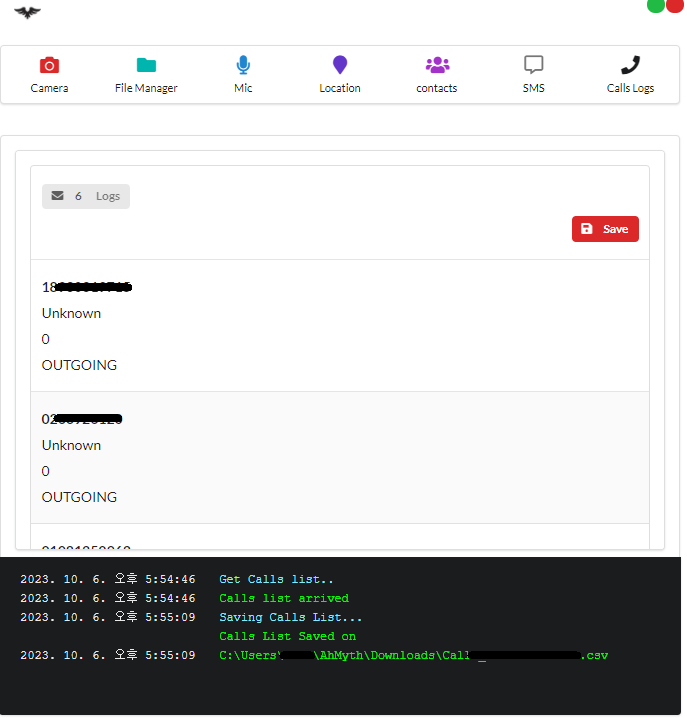


42[“order”,{“ord er”:”x0000sm”,” e xtra”:”ls”}] 이 내용이 바로 sms 리스팅을 나타낸다.

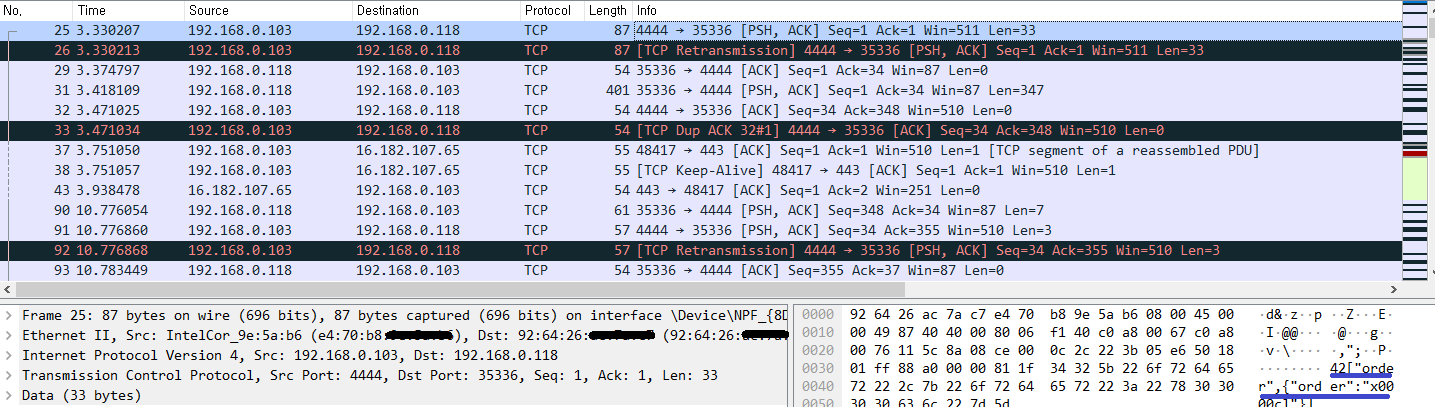


[ github의 sms 리스팅 관련 내용 ]

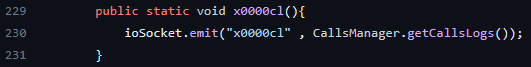
**7. 통화 기록**



통화기록을 출력한다.



42[“order”,{“ord er”:”x0000cl”}]가 바로 통화기록 리스팅을 나타내는 문장이다.



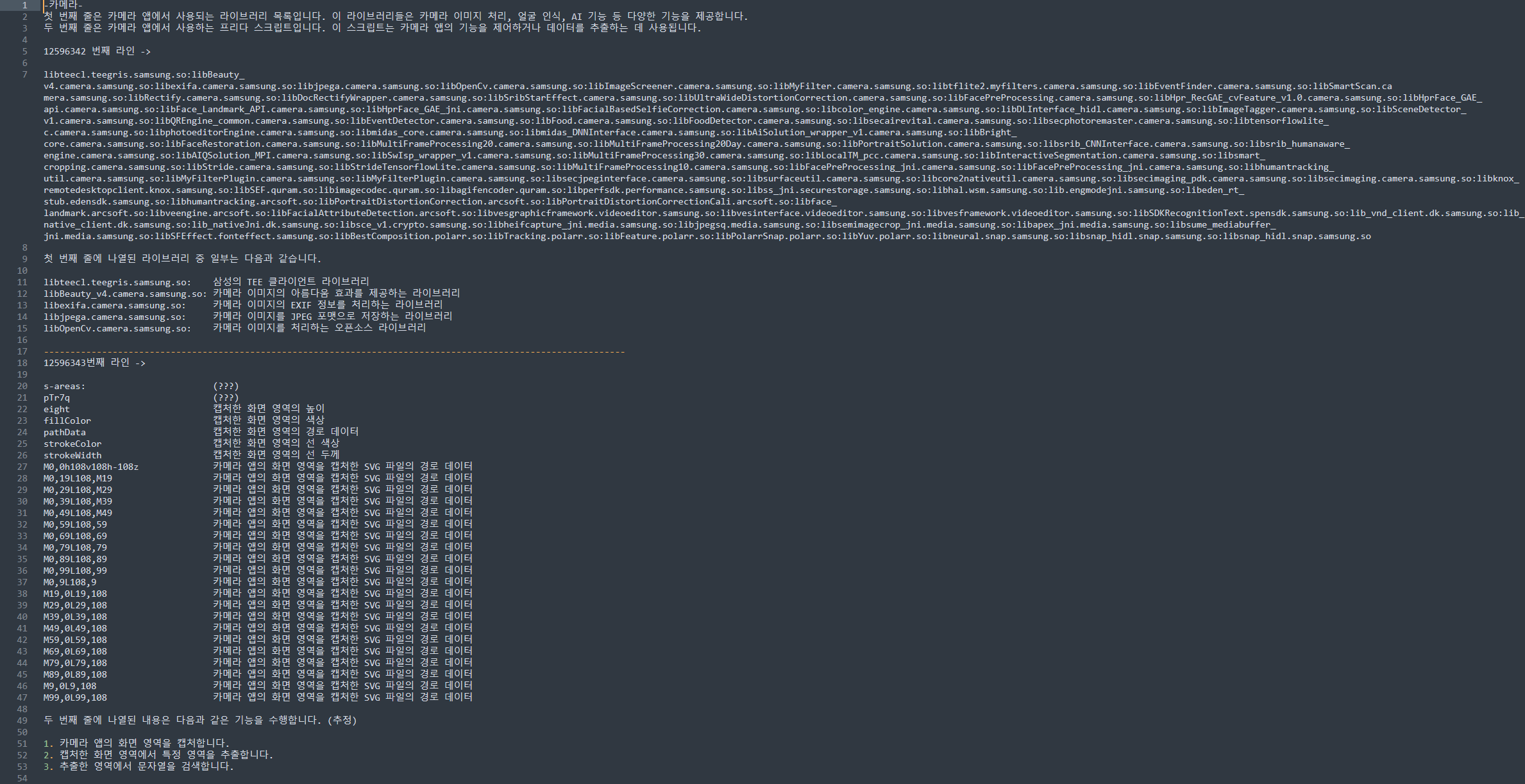
[ 통화기록을 리스팅하라는 내용의 코드 ]

## **스크립트 실행 결과**

## [ Active Data ]

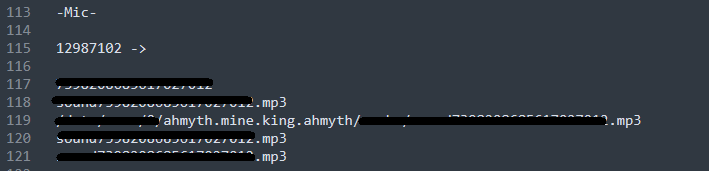
**01\_Virtual Memory**

* 카메라



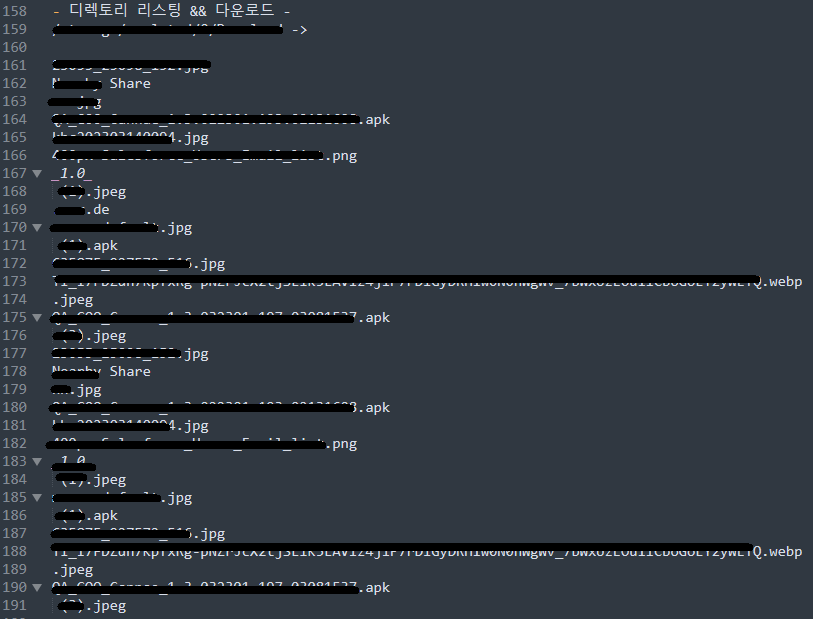
전면, 후면 카메라를 통해 사진을 찍었음에도 불구하고 촬영한 이미지 파일명을 가상 메모리 덤프 데이터에서 찾을 수 없다. 대신에 카메라 관련 메타 데이터를 찾을 수 있다. 예를 들어서 두 번째 사진의 60번째 줄에 4032x3024라고 정면 카메라가 지원하는 해상도에 대한 정보가 담겨 있었는데 이는 실제로 정면 카메라로 찍은 사진의 해상도 또한 해당 해상도 값을 가지는 것을 확인할 수 있다.

* 녹화



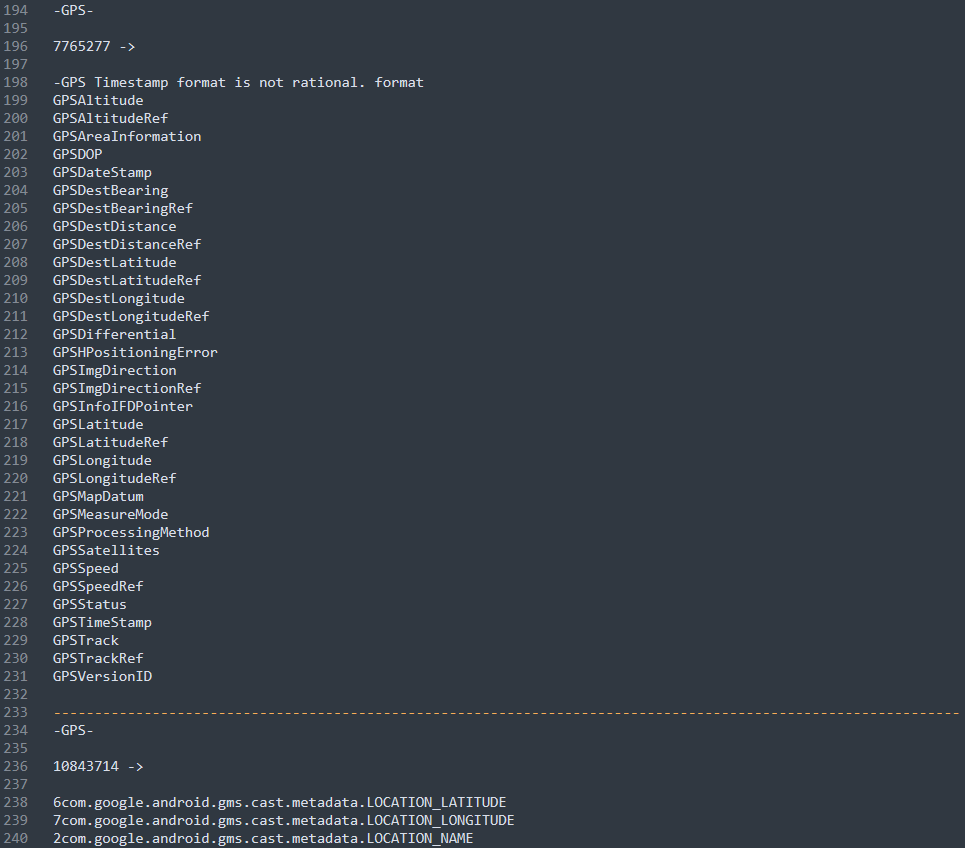
녹화를 수행했을 때, 생성되는 파일이 있었는데 해당 파일 명과 일치하는 파일명은 덤프 내에서 찾을 수 있다.

* 디렉토리 리스팅 && 다운로드



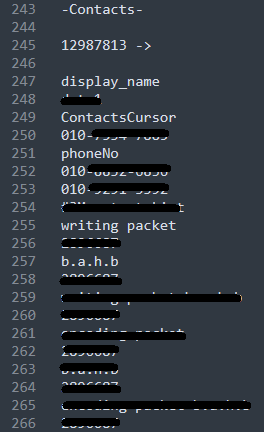
가상 메모리 덤프 내에서는 특정 디렉토리 위치 내의 데이터들이 올바르게 정렬되어서 나오지 않고 또한 정확히 어떤 데이터를 다운받았는 지까지는 확인할 수 없다. 대신에 어떤 디렉토리를 리스팅 했는지에 대해서는 확인할 수 있다.

* GPS



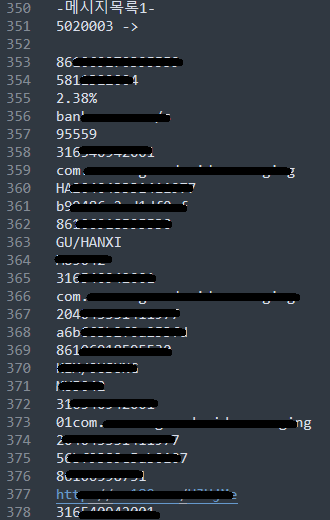
GPS로 추적한 위도, 경도 결과 값은 찾을 수 없다. 대신에 GPS 관련 데이터를 처리하거나 저장하기 위한 다양한 속성이나 메타데이터를 나열한 것으로 추정되는 부분을 찾아서 출력했다.

* 연락처



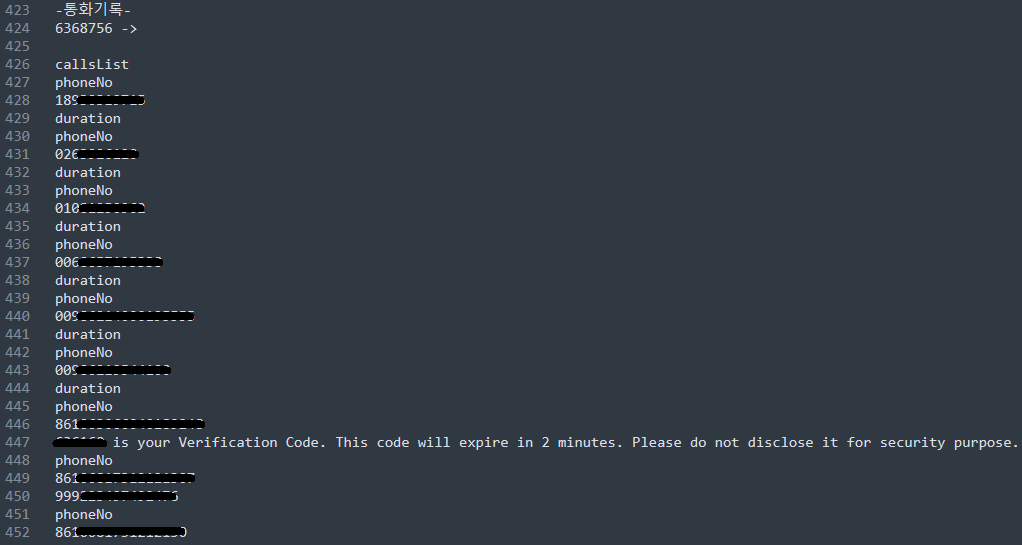
연락처 항목에서 한글로 된 이름은 제대로 출력되지 않지만 전화번호는 제대로 볼 수 있다.

* 메시지 목록



SMS 항목에서 영어와 숫자는 정상적으로 보였고 그 외 중국어나 한국어는 제대로 나오지 않는 것을 볼 수 있다.

* 통화 기록



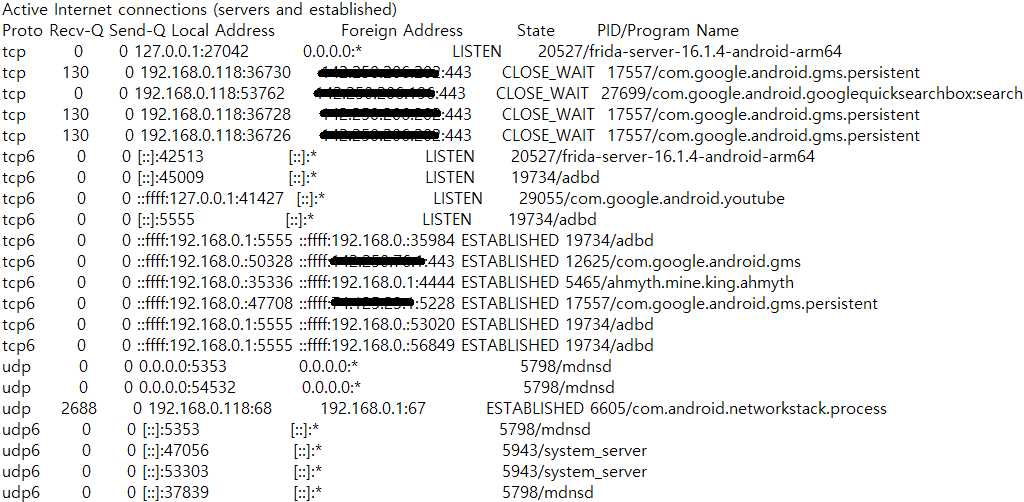
통화 기록으로 전화번호가 제대로 출력되는 것을 확인할 수 있다.

**02\_NetworkInfo**

arp: 현재 라우터와 pc 아이피가 arp 캐시에 등록된 것을 볼 수 있다.



netstat: frida-server, AhMyth가 각각 LISTEN, ESTABLISHED이며, 어느 IP 주소, 포트로 연결되었는지 볼 수 있다.

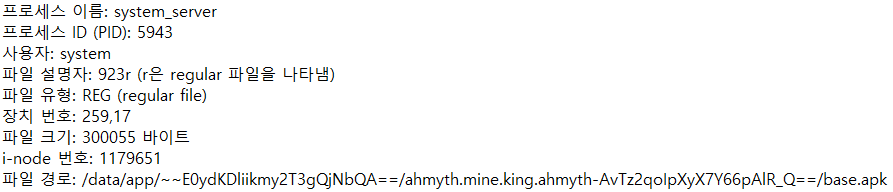


**03\_ProcessInfo**

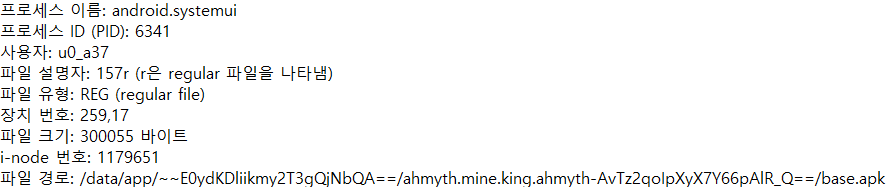
lsof: 실행 중인 프로세스가 열고 있는 파일 목록을 보여주는 리눅스/유닉스 명령어다. 각 줄은 하나의 열린 파일을 나타내며, 주요 필드에는 프로세스 이름, 프로세스 ID, 파일 설명자, 파일 유형, 파일 크기, i-node 번호 및 파일 경로 등이 포함한다.



첫 번째 줄 결과를 분석하면 아래와 같다.



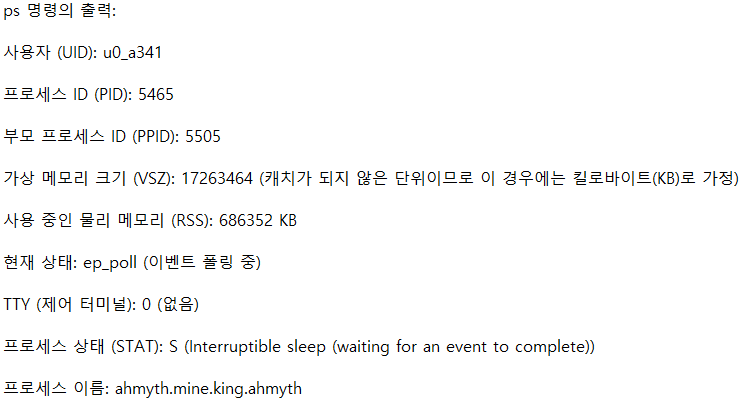
두 번째 줄 결과를 분석하면 아래와 같다.



ps: AhMyth RAT 프로세스의 스냅샷을 제공받을 수 있다. 그래서 해당 프로세스에 대한 프로세스 ID, 사용자, CPU 사용량, 메모리 사용량, 실행 중인 명령, 프로세스 상태 등의 정보를 제공할 수 있다.



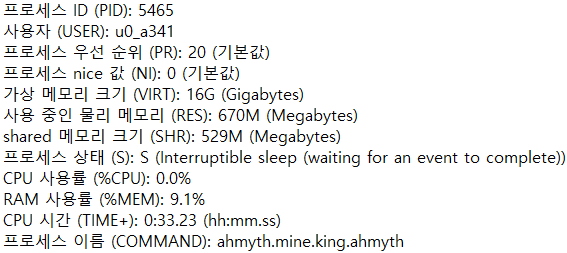
위 ps 명령의 결과를 분석하면 아래와 같다:



top: 실시간으로 AhMyth RAT 프로세스를 모니터링 할 수 있다. top은 시스템의 상태, 실행 중인 프로세스, cpu와 메모리 사용량 등의 다양한 정보를 실시간으로 업데이트하며 표시한다.



위 top 명령 결과를 분석하면 아래와 같다:



**04\_Logon\_User\_Information**

**1. Image\_video**

수집 경로 : /sdcard/DCIM

수집된 폴더 중 Camera에는 테스트용으로 넣어둔 사진을 확인할 수 있다.

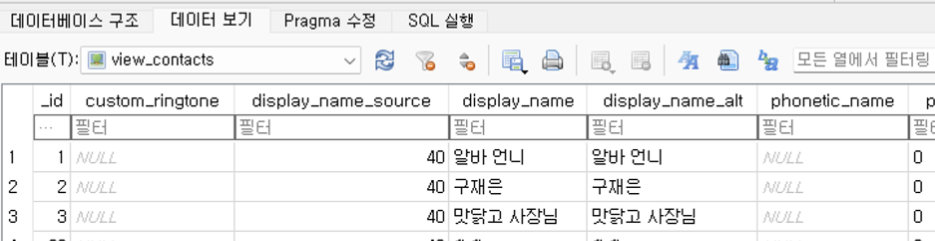
악성 앱의 기능으로 피해자 휴대폰의 카메라를 제어해 사진을 찍더라도 피해자 휴대폰에 저장되는 것이 아니라 공격자의 서버에 저장되는 것으로 생각된다. 따라서, Image\_video에 수집되지 않는다.

**2. Contact & mms**

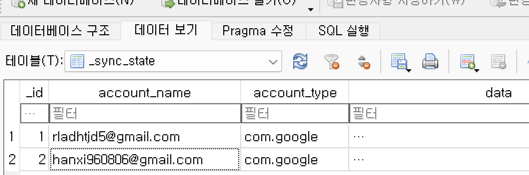
/data/data/com.samsung.android.providers.contacts/databases/contacts2.db" "$LogonUser\_Dir/contacts2\_data\_data.db

/data/user/0/com.samsung.android.providers.contacts/databases/contacts2.db" "$LogonUser\_Dir/contacts2\_data\_user.db

위의 2개 경로에서 수집하였다.



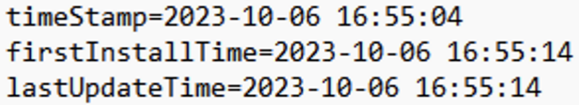
DB 확인 결과 저장된 contact의 리스트를 확인할 수 있었다.



휴대폰에 로그인된 메일 주소도 확인 가능했다.

**3. AppData**

스크립트 실행 중 Specify packages 옵션을 이용해 수집할 패키지를 지정한다. 악성 앱의 패키지 이름인 AhMyth.mine.king.AhMyth를 지정해 AhMyth.mine.king.AhMyth.txt를 수집하였다.

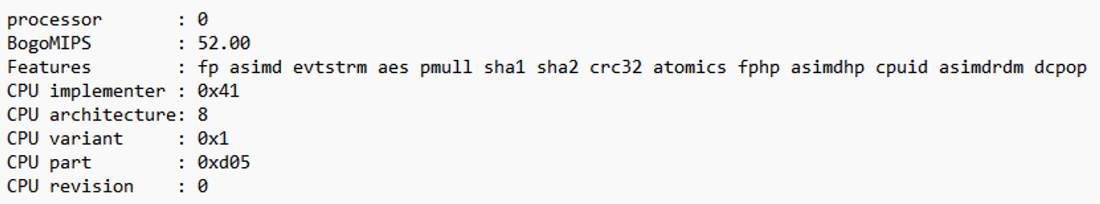


수집 결과로 악성 앱의 설치시간인 2023-10-06 16:55:14를 찾을 수 있었다.

또한, requested permissions, install permissions, runtime permissions 필드에 악성 앱이 동작하기 위해 필요한 것들을 확인할 수 있다.

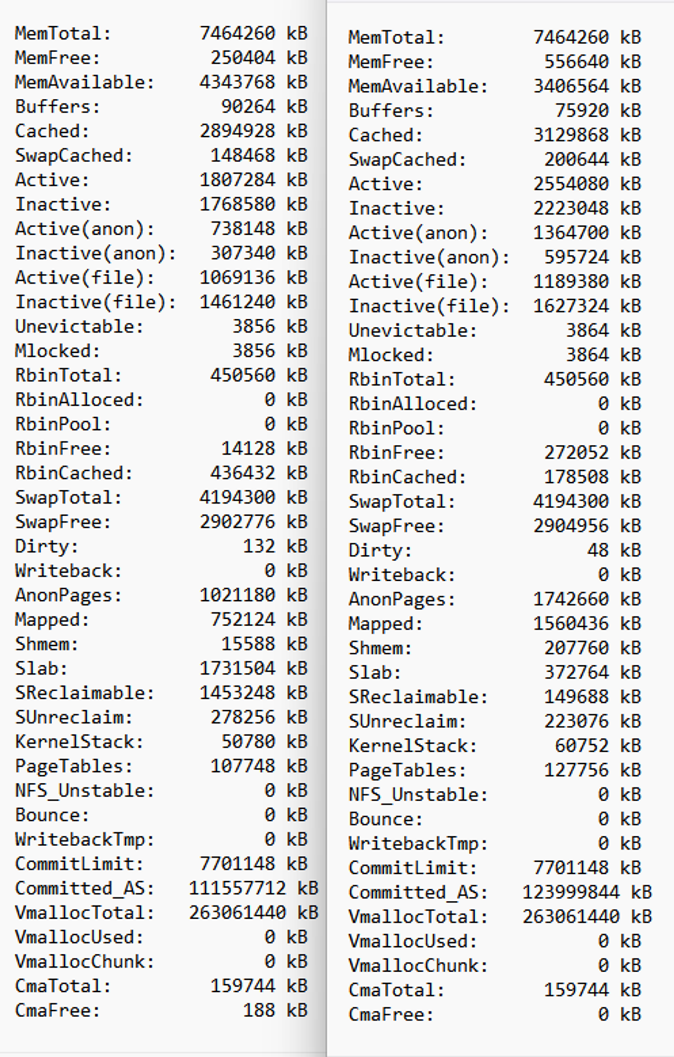
**05\_System\_Information**

**1. Cpu\_info**



악성 앱 설치 전후의 변화는 없지만, Features에서 CPU가 지원하는 다양 명령어 집합과 기능을 나타낸다.

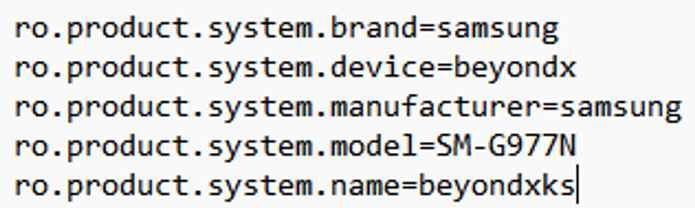
**2. Memory\_info**



위는 수집된 memory info의 내용 중 일부를 가져온 것이다. 왼쪽은 설치 전의 수집 결과이고, 오른쪽은 설치 후의 수집 결과이다.

악성 앱 설치 후에 Active, Inactive의 데이터양이 많아져서 사용 가능한 메모리 양이 줄어들었다.

**3. Device\_info**



기기의 구성 설정 및 빌드 속성을 나타내는 파일이다. 순서대로 브랜드, 디바이스 코드 이름, 제조사, 모델 번호, 시스템 이름을 나타낸다.

대상 휴대폰은 모델 번호가 SM-G977N인 beyondxks 이름의 삼성 갤럭시 S10 5G 시리즈이다.

**4. Battery**

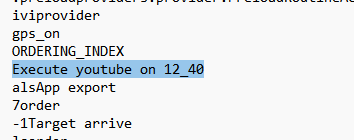
수집된 항목들 중 중요한 정보를 가지는 일부만 해석하였다.

capacity : 수집 당시 기기의 배터리 90%  
status : 충전 중이 아님  
health : 배터리 상태 좋음  
technology : 배터리 기술 리튬 이온(Li-ion)

**06\_Autoruns**

대상 기기에서 빅스비 루틴으로 자동 실행 항목을 지정하였다.

가상머신 2단계에서 수행했던 Frida 덤프를 이용해 com.samsung.android.app.routines를 덤프하여 string.txt를 얻을 수 있었다.



대상 기기에 지정했던 12시 40분에 유투브 실행 작업을 확인할 수 있었다.

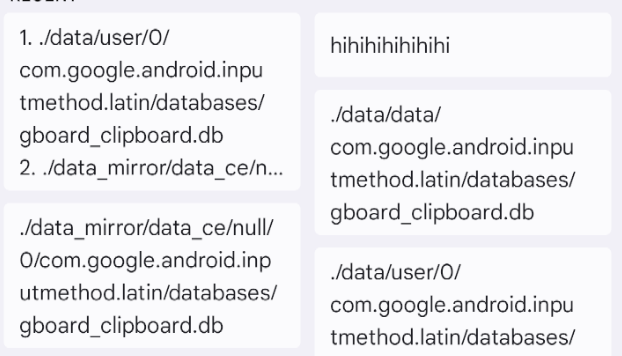
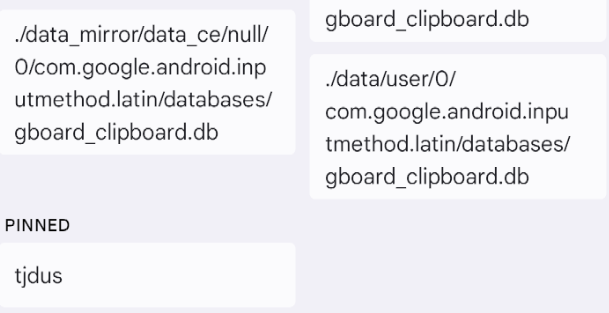
**07\_Clipboard**

1. Clipboard는 하나의 파일에서 다른 파일로 옮길 자료를 저장하기 위한 주기억장치의 부분을 의미한다.

2. Clipboard 데이터를 저장하고 있는 경로 : Gboard를 사용할 경우에만 수집 가능한 파일이다.

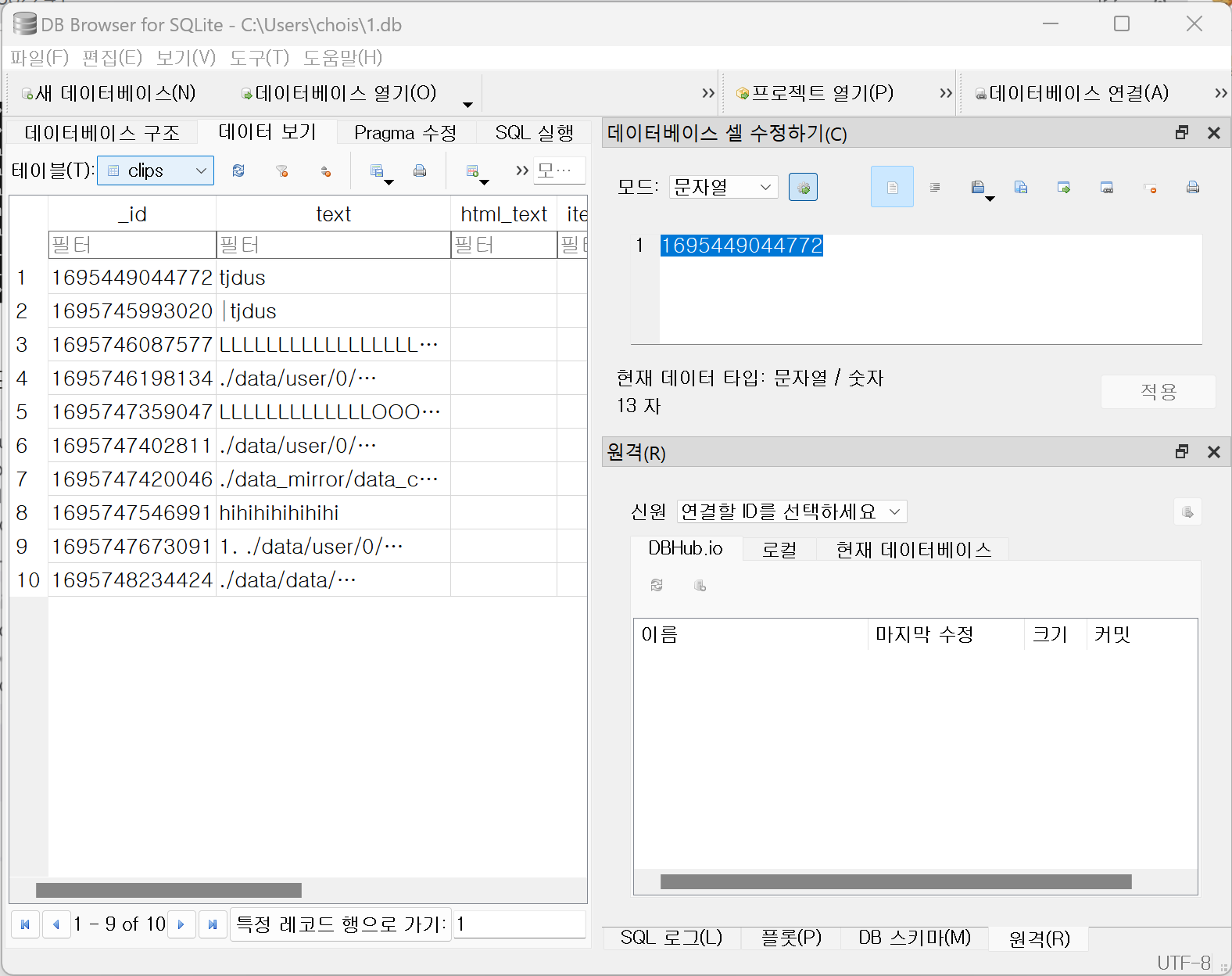
* 1. ./data/user/0/com.google.android.inputmethod.latin/databases/gboard\_clipboard.db
  2. ./data\_mirror/data\_ce/null/0/com.google.android.inputmethod.latin/databases/gboard\_clipboard.db : 해당 파일은 복제가 불가능하다.
  3. ./data/data/com.google.android.inputmethod.latin/databases/gboard\_clipboard.db

3. 클립보드

4. 추출한 파일

- SQLite 프로그램으로 열고 clips라는 Table을 보면 clipboard data를 확인할 수 있다.



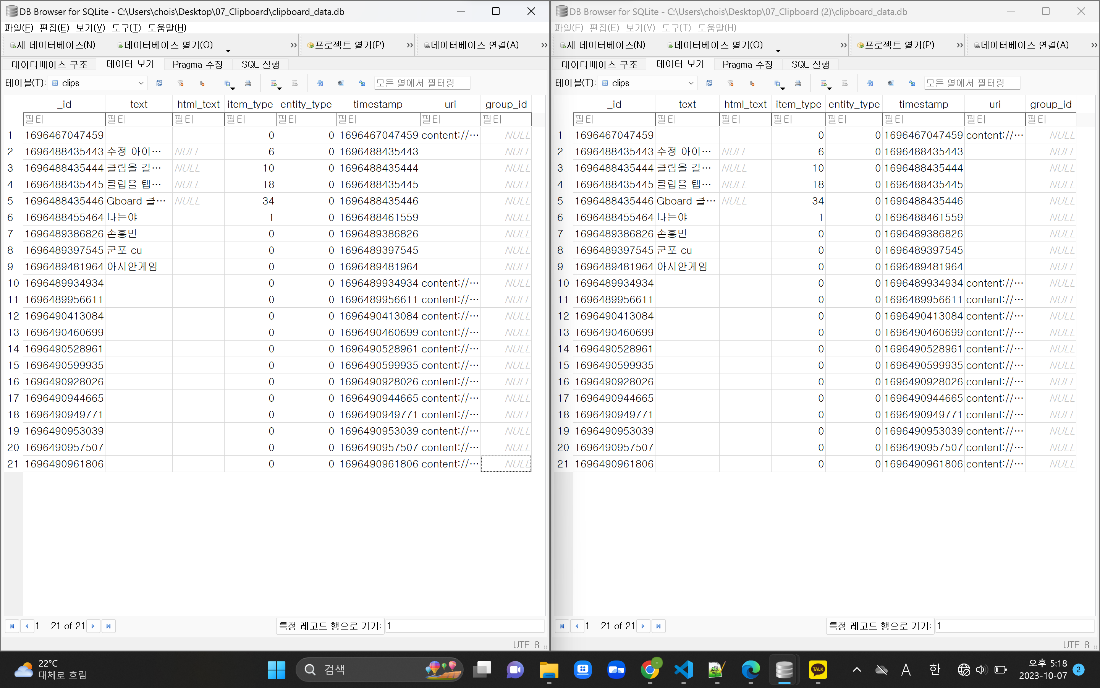
- text: 클립보드에 복사한 값

- timestamp: text 값을 복사한 시간

5. 실제 폰에서 Data 수집 성공 여부: 성공

- 삼성 키보드에서 Gboard로 설정을 바꾸고 수집했을 때 수집에 성공했다.

6. 악성 앱 실행 전과 후: 변화 없음



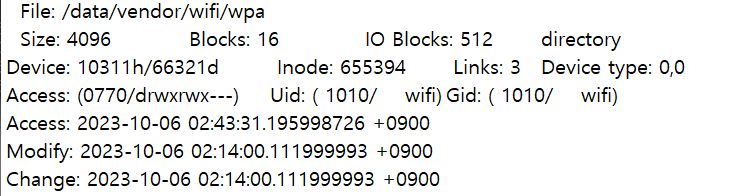
[ 왼쪽: 악성 앱 실행 전, 오른쪽: 악성 앱 실행 후 ]

⇒ 악성 앱에서 clipboard 기능을 사용하지 않았기 때문에 변화가 없었던 것으로 판단된다.

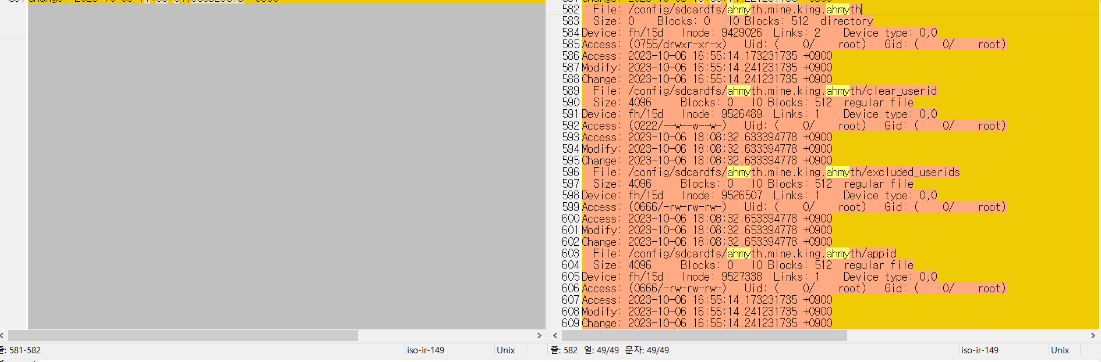
## [ InActive Data ]

**01\_Filesystem\_Metadata\_info**

스크립트 실행을 통해서 파일 시스템 전체의 메타데이터를 수집한다.



수집되는 메타데이터의 내용에는 파일 크기, 블록, 파일의 종류(directory), 파일이 존재하는 장치의 번호(Device), 권한정보(상단 Access), 접근 시각(atime) 정보(Modify), 변경 시각(ctime) 정보(Change) 등을 알 수 있다.

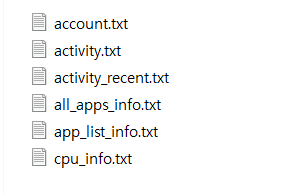


[ 좌-악성 앱 설치 전 / 우- 악성 앱 설치 후 ]

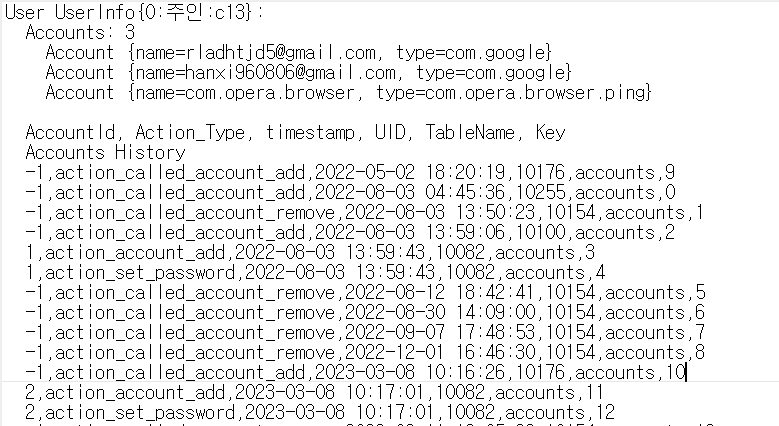
sdcardfs는 안드로이드 장치에서 외부 저장소인 SD 카드에 대한 접근과 관련된 작업을 처리하는 중요한 구성 요소이다. 이 외부 저장소에 AhMyth 앱이 설치되었음을 확인하였다.

**02\_SystemUserAppli\_Setting\_info**

시스템, 사용자, 애플리케이션 설정 정보를 수집한다. 2단계 실행 시 6개의 파일을 생성한다.

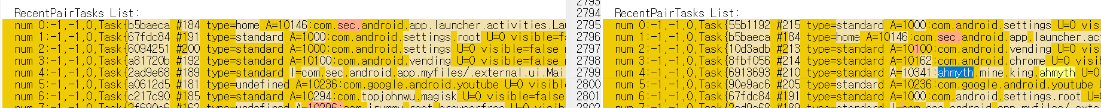


**1. account.txt :** 모든 계정에 대한 정보가 저장한다. 악성 앱으로 인한 변화는 없다.



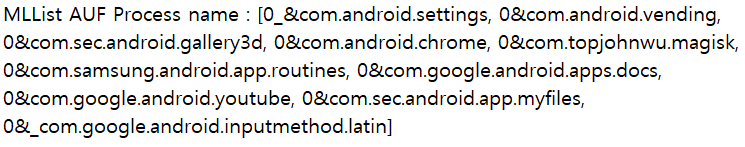
**2. activity.txt :** activity 시스템 상태 확인 정보가 저장된다.

안드로이드에서 activity란 앱의 전반적인 활동을 담당하는 것이다.



AhMyth 동작 내역들을 확인해볼 수 있다.

[ 악성 앱 설치 전 ]



[ 악성 앱 설치 후 ]



다음과 같이 실행파일에 ***AhMyth.mine.king.AhMyth*** 가 추가됨을 볼 수 있다.

**3. activity\_recent.txt :** 최근에 사용한 activity 시스템 상태 확인 정보가 저장된다.



[ 악성앱 실행후 파일 기준으로 activity.txt와 activity\_recent.txt 크기 비교 ]

**4. app\_list\_info.txt :** 설치된 모든 앱의 리스트가 저장된다. (간단한 버전)



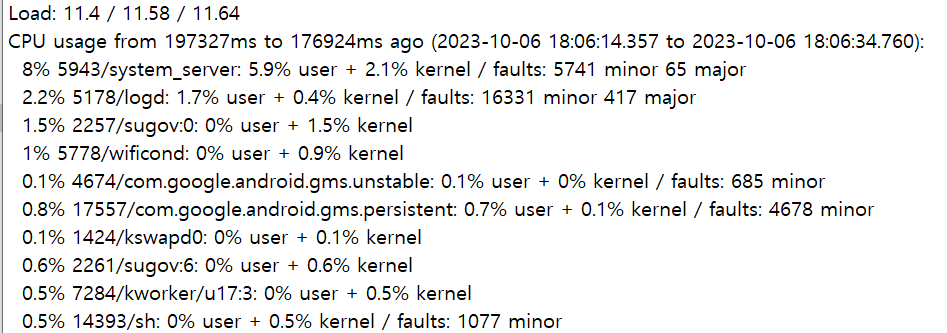
AhMyth 앱이 설치 된 것을 확인할 수 있다.

**5. all\_apps\_info.txt** : 설치된 모든 앱의 정보가 저장된다. (자세한 버전)



AhMyth 앱의 UID, 소스경로 등을 확인할 수 있다.

**6. cpu\_info.txt :** CPU 프로세서 사용 정보가 저장된다. 악성 앱으로 인한 변화는 없다.



처음 실행 줄은 Load : 최근 1분 / 5분 / 15분 로드량을 표시한다.

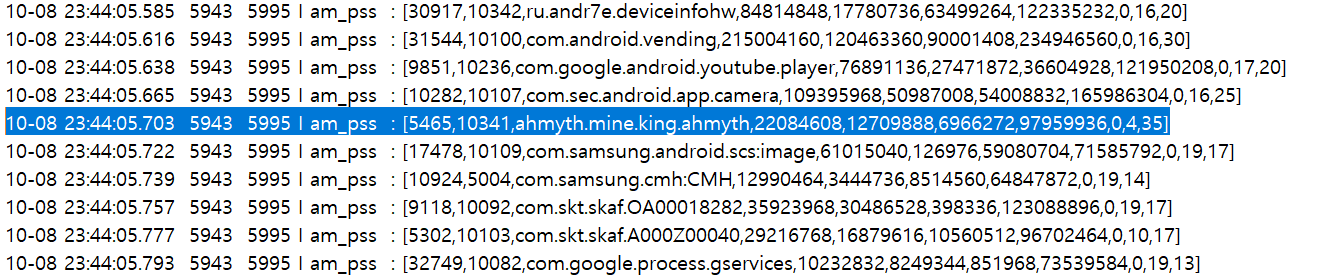
PID(Process ID) 리스트가 cpu 사용량 순으로 정렬되어 저장된다.

**03\_Log\_Files**

**1. event\_log\_info.txt :** 이벤트 로그 수집한다.

안드로이드 운영 체제의 동작, 앱 실행, 하드웨어 상태 등과 관련된 다양한 이벤트가 기록된다.

am-kill, power\_partial\_wake\_state 등 TagName이 분류되어 있어 동작의 의미를 파악할 수 있다.

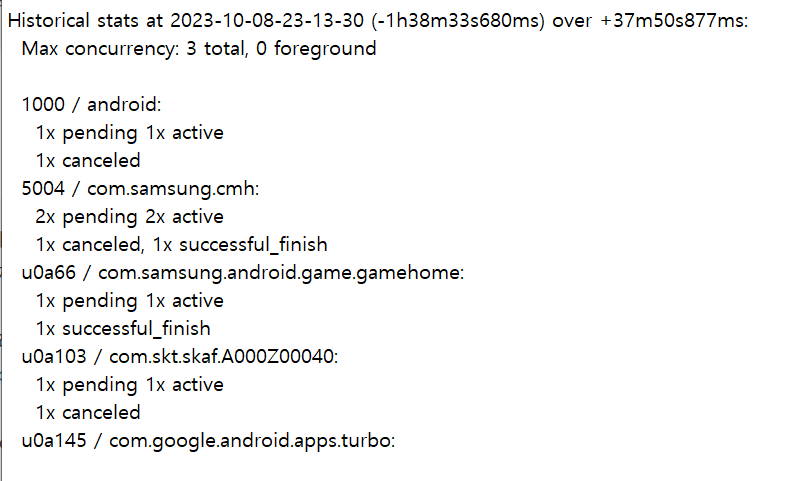


AhMyth 동작 흔적이 로그로 남아 있다.

**2. jobscheduler\_info.txt**

안드로이드 운영체제에서 제공하는 기능으로 애플리케이션 작업을 예약하고 관리하는데 사용된다.

결과물은 주로 작업의 실행 상태, 성공 또는 실패 여부, 작업 완료 시간 등을 제공한다.



**04\_Recycle Bin**

일반적인 휴지통 기능과 유사하다.

* 생성 폴더
* .Trash
* Lost+found
* trash

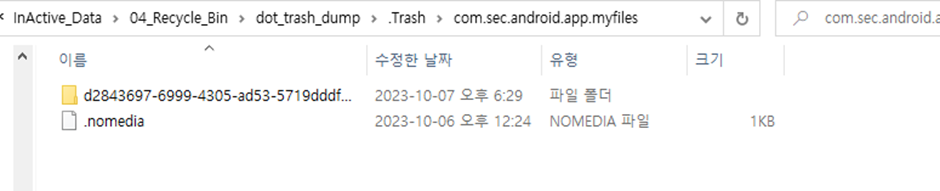
실행 시 .Trash 폴더에 2개의 폴더 생성된다.

* 생성 폴더
* com.sec.android.app.myfiles
* com.sec.android.gallery3d

myfile은 휴대폰의 내파일 앱의 휴지통이며 gallery3d는 갤러리 앱의 휴지통이다.

같은 이미지 파일을 삭제해도 삭제하는 앱에 따라서 저장 위치가 달라진다.

* .nomedia 파일
* 역할 : 폴더의 내용을 숨기거나 음악, 비디오, 벨소리 등 숨기기 위해 사용된다.
* 덤프 파일에서는 com.sec.android.app.myfiles (안드로이드에서 내파일 앱) 안에 존재하며 해당 폴더가 앱 등의 검색 결과에 나타나지 않게 한다.



악성 앱 실행 후 차이점은 없다.

**05\_Browser Info**

브라우저의 캐쉬, 쿠키, 히스토리 파일을 통해 브라우저 사용 흔적을 수집한다.

* **Chrome**
* **cache**

형식은 data\_0 이다.

실행 시 저장되는 구성 폴더는 아래와 같다.

**1. .com.google.Chrome.’defalut’**

**2. Cache**

일반적인 브라우저 관련 cache 파일이다.

**3. Code Cache**

프로그래밍 언어의 코드를 저장하고 실행하는 데 사용되는 메모리 공간이다.

프로그램의 소스 코드나 중간 단계의 컴파일 된 코드를 저장하여 반복적인 컴파일 작업을 피하고 실행 속도를 향상시킨다.

**4. Crash Reports**

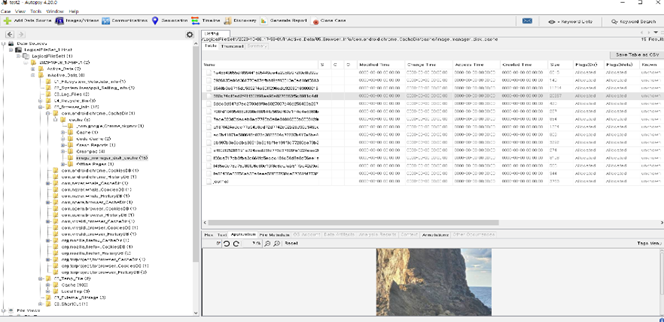
비정상적인 종료 혹은 충돌 시 발생하는 오류 정보 기록한다.

**5. Crashpad**

구글에서 개발한 크래시 리포팅 시스템 어플의 충돌 정보 수집 서버로 보고한다.

**6. image\_manager\_disk\_cache**

웹 페이지에서 로드한 이미지를 디스크에 임시로 저장하여 다음에 동일한 이미지가 필요할 때 다시 다운로드하지 않고 바로 불러올 수 있다.



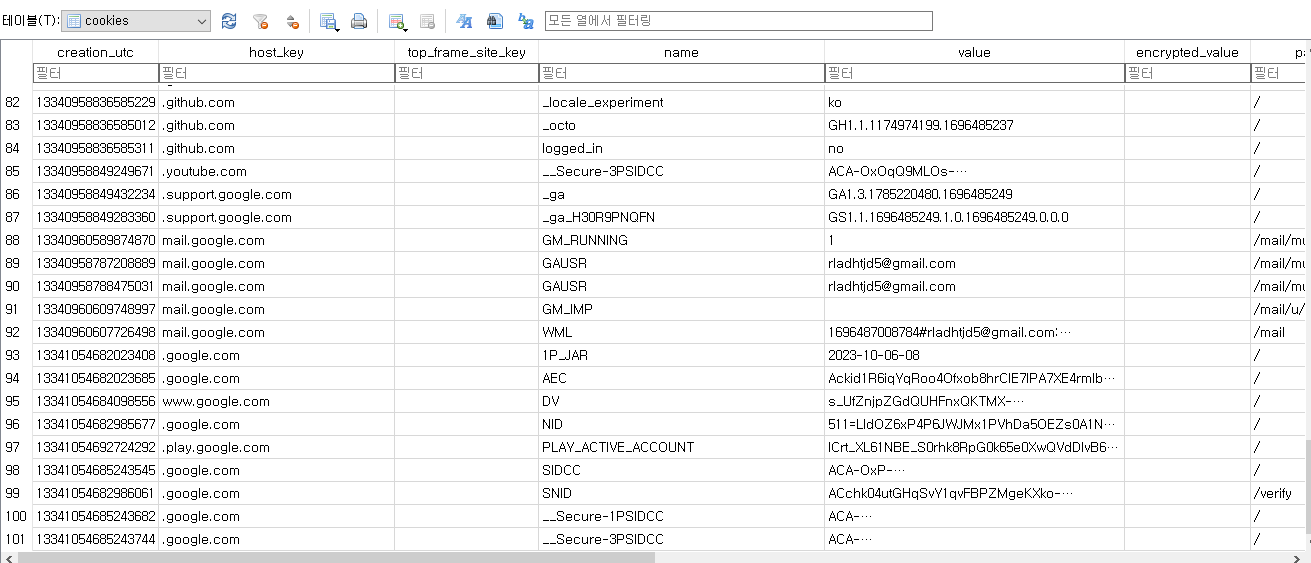
**7. Offline pages**

사용자가 인터넷에 연결되어 있지 않을 때도 이전에 방문한 웹 페이지를 볼 수 있게 해준다.

* **cookies**

형식은 db이다.

저장 정보는 호스트, path , 변수, 값, 방문 회수, 마지막 접근 시간, 쿠키 만료 시간이다.



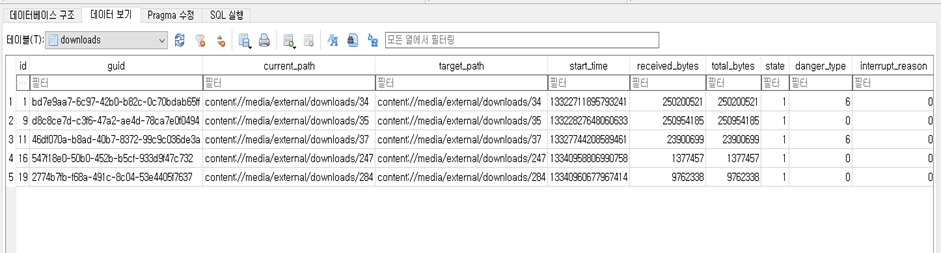
<cookies>

* **History**

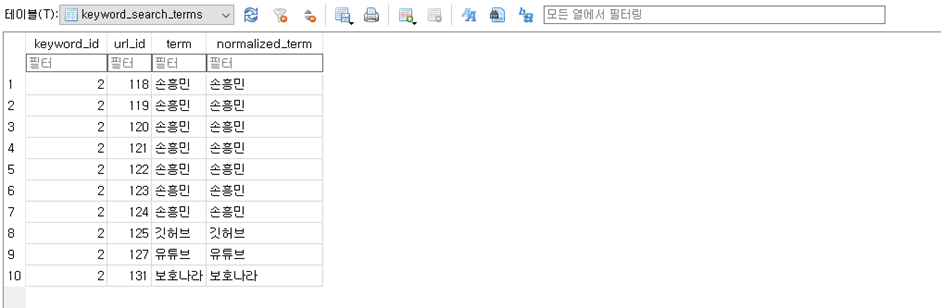
형식은 db이다.

저장 정보는 URL, Title, 방문 횟수, 방문 타입, 방문 시간이다.

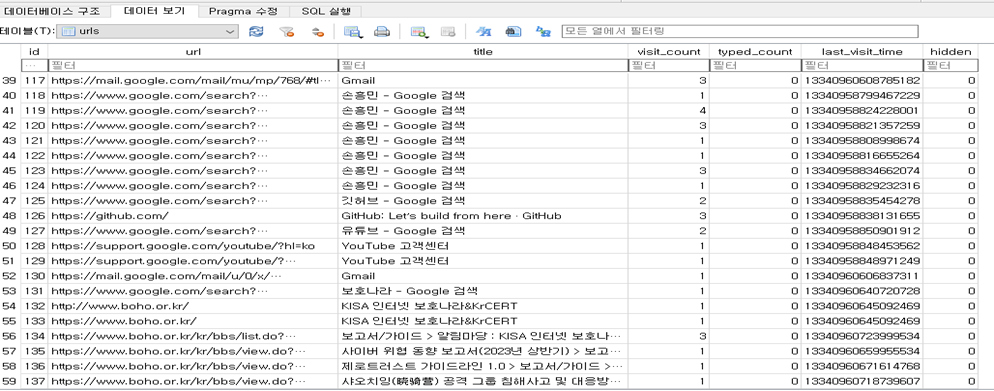
주요 테이블 : downloads, keyword\_search\_terms, urls, visits



<downloads>



<keword\_search\_terms>

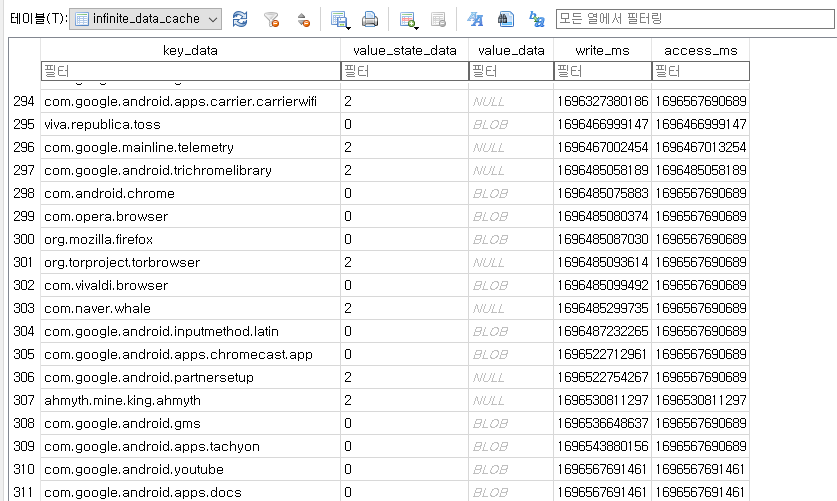


<urls>

악성앱 실행 후 차이점은 없다.

**06\_Temp File**

임시로 저장된 파일 중 cache와 local tmp 파일을 수집한다.

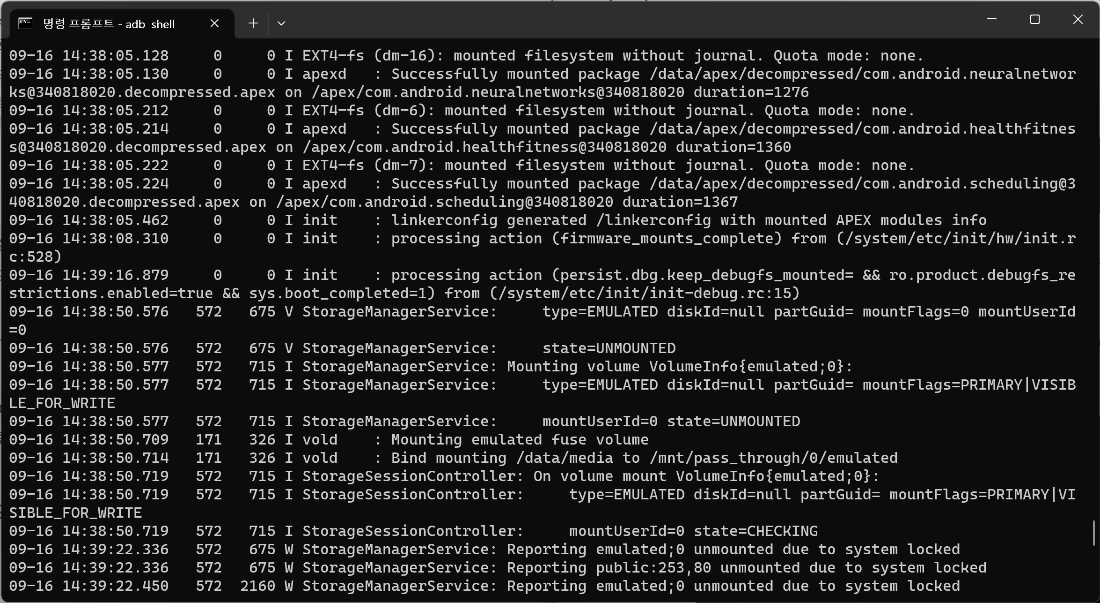


수집된 Cache 데이터 중 autofill\_infinite\_data\_cache 부분에서 AhMyth.mine.king.AhMyth(악성앱)의 흔적을 발견했다.

**07\_External Storage**

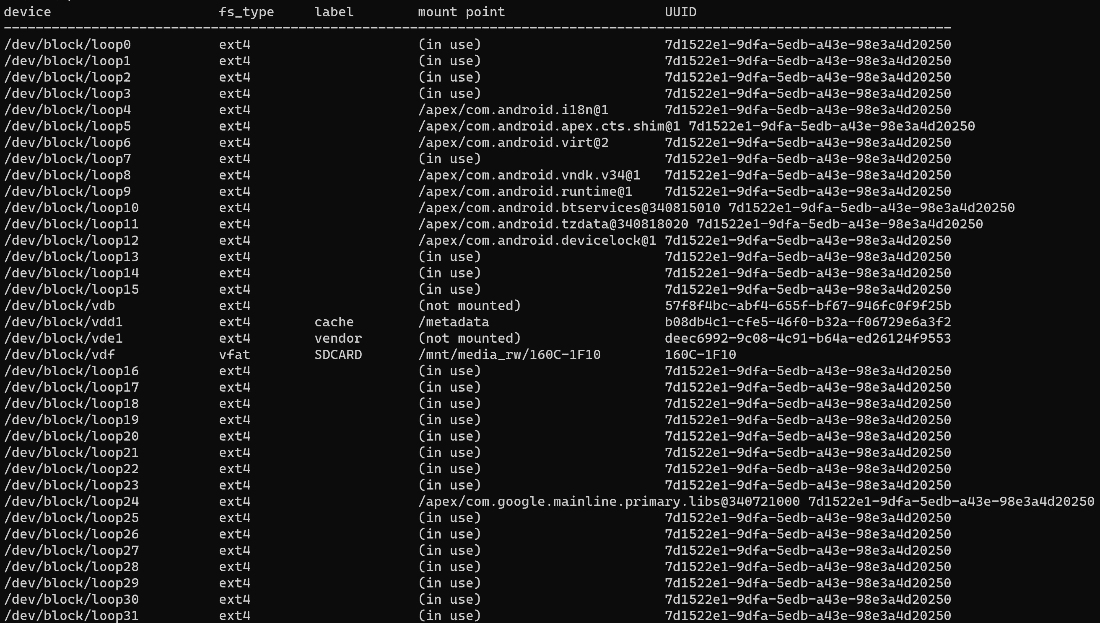
1. 수집 방법

a. logcat | grep -i “mount” 명령어



외부저장장치 흔적 로그에는 mount라는 키워드가 포함되어 있었기 때문에 mount 키워드를 가지고 있는 로그만 추출했다.

b. blkid -o list 명령어



파티션, SSD, 하드 디스크 등과 같은 블록디바이스 정보를 출력한다. 세부 정보는 아래와 같다.

- device: 블록 다바이스 경로

- fs\_type: 파일 시스템 유형

- label: 파일 시스템에 지정된 레이블(이름)

- mount point: 블록 디바이스가 파일 시스템으로 마운트되는 디렉토리 경로

- uuid: 파일 시스템에 할당된 UUID 값

2. 실제 폰에서 Data 수집 성공 여부

a. logcat | grep -i “mount”: 실패

- 해당 폰에 sdcard가 없었기 때문에 수집되지 않은 것으로 판단된다.

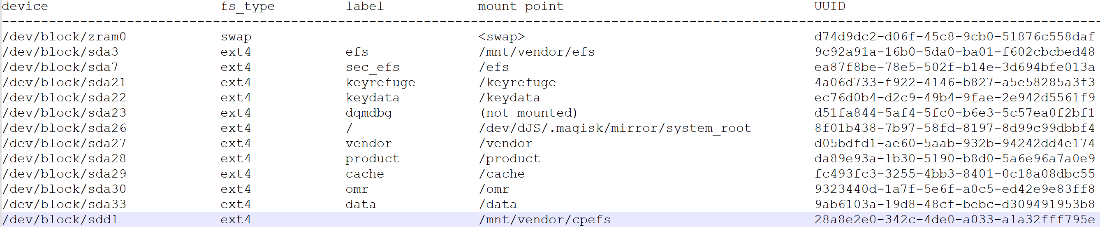
b. blkid -o list: 성공

3. 악성 앱 실행 전과 후: 변화 없음

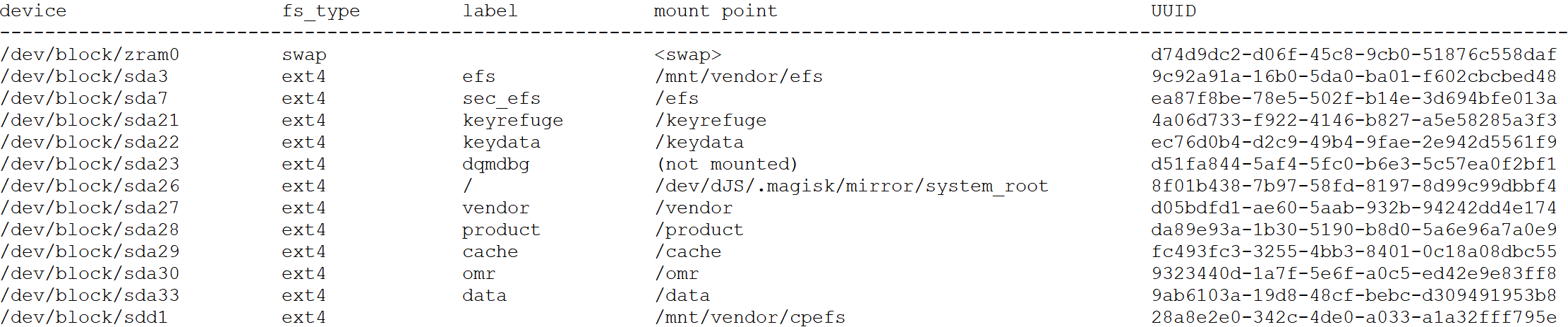
a. logcat | grep -i “mount”: 악성앱 실행 전후 둘 다 Data가 없다.

b. blkid -o list: 값이 동일하다.

- 악성 앱 실행 전



- 악성 앱 실행 후



⇒ 블록 디바이스에 영향을 주는 악성 앱이 아니었기 때문으로 판단된다.

## **결론**

## 실제 환경에서 스크립트 활용 가능성

**# 활성데이터**

|  |  |  |  |
| --- | --- | --- | --- |
| **단계** | **세부항목** | **가상기기** | **실기기** |
| 1단계 | 가상메모리 | 성공 | 성공 |
| 2단계 | arp | 성공 | 성공 |
| netstat | 성공 | 성공 |
| ifconfig | 성공 | 성공 |
| wifi | 성공 | 성공 |
| network\_interface | 성공 | 성공 |
| route | 성공 | 성공 |
| network properties | 성공 | 성공 |
| iptables | 성공 | 성공 |
| sysctl | 성공 | 성공 |
| tcp 소켓 | 성공 | 성공 |
| udp 소켓 | 성공 | 성공 |
| 3단계 | top | 성공 | 성공 |
| ps | 성공 | 성공 |
| lsof | 성공 | 성공 |
| activity process | 성공 | 성공 |
| meminfo | 성공 | 성공 |
| strace | 성공 | 실패 |
| 4단계 | 연락처 | 성공 | 성공 |
| DCIM | 성공 | 성공 |
| 앱 데이터 | 성공 | 성공 |
| 5단계 | 장치 정보 | 성공 | 성공 |
| Cpu 정보 | 성공 | 성공 |
| 메모리 정보 | 성공 | 성공 |
| 배터리 정보 | 성공 | 성공 |
| 6단계 | 자동실행 항목 덤프 | 성공 | 성공 |
| 7단계 | 클립 보드 덤프 | 성공 | 성공 |

**# 비활성데이터**

|  |  |  |  |
| --- | --- | --- | --- |
| **단계** | **세부항목** | **가상기기** | **실기기** |
| 1단계 | 파일시스템 메타데이터 | 성공 | 성공 |
| 2단계 | 모든 계정 정보 | 성공 | 성공 |
| Activity 상태 확인 | 성공 | 성공 |
| 최근 activity 상태 확인 | 성공 | 성공 |
| 설치된 앱 리스트 | 성공 | 성공 |
| 설치된 앱 상세 정보 | 성공 | 성공 |
| CPU 프로세서 정보 | 성공 | 성공 |
| 3단계 | 이벤트 로그 | 성공 | 성공 |
| 작업 예약 관리 | 성공 | 성공 |
| 4단계 | trash | 성공 | 성공 |
| .Trash | 성공 | 성공 |
| lost+found | 성공 | 성공 |
| 5단계 | chrome | 성공 | 성공 |
| firefox | 성공 | 성공 |
| opera | 성공 | 성공 |
| whale | 성공 | 성공 |
| tor | 성공 | 성공 |
| vivaldi) | 성공 | 성공 |
| 6단계 | cache | 성공 | 성공 |
| LocalTmp | 성공 | 성공 |
| 7단계 | blkid | 성공 | 성공 |
| logcat | 성공 | 성공 |

## 시사점

프로젝트를 통해 안드로이드 가상머신에서 실행한 쉘 스크립트를 실제 악성 앱에 감염된 스마트폰에 실행했을 때, 스크립트의 98%의 성공 결과를 얻을 수 있었다.

안드로이드 시스템의 기본 정보를 수집하고 분석하는 이 과정을 통해, 쉘 스크립트에 대한 이해도를 높일 수 있었으며, 안드로이드 시스템의 구성 환경에 대한 인식을 갖게 되었다. 또한, 이를 통해 침해사고 발생 시 필요한 접근 방식에 대한 시야를 확장할 수 있었다.

향후에는 직접 제작한 쉘 스크립트가 실제 안드로이드 포렌식으로 활용될 수 있도록 각 생성된 데이터에 대한 타임스탬프 및 해시 값을 생성하는 기능을 추가할 예정이다. 뿐만 아니라 네트워크 상에서 원격으로 수집될 수 있도록 기능을 구현할 계획이다.