Practica 4: Analisis de Vulnerabilidades

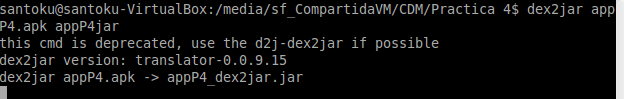
By: Adrian Tendero Lara.

# ANALISIS ESTATICO

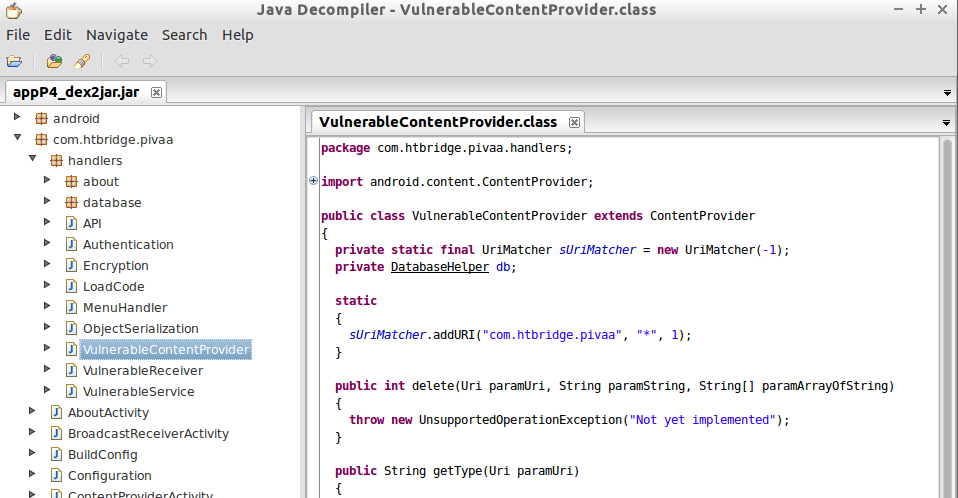
En esta practica nos piden que analizemos una app en busca de sus vulnerabilidades, lo primero que realizaremos es un análisis estático del código de la aplicación, para ello primero tenemos que desempaquetar el apk de la aplicación objetivo cosa que se hace con el APKtool

“apktool -d <APK objetivo>”

Después podemos utilizar el dex2jar para traducir el código a un lenguaje java



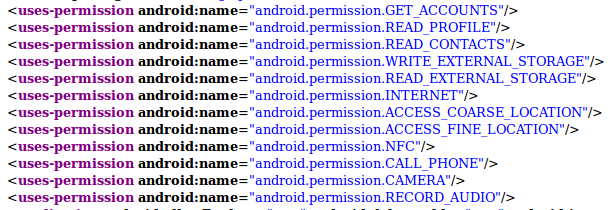
Con esto ya tenemos un código mas legible que se puede visualizar con JD-GUI



Ahora procedemos ha hechar un vistazo al manifest de la aplicación.

Lo primero que hay que comprobar son los permisos declarados en el manifest así como si hay backup y el modo debug activado.

Como se puede observar la app solicita muchos permisos.



De los cuales algunos podrían no estar siendo usados, cosa que comprobaremos mas adelante en el análisis con el androzyle.

En el manifest también podemos comprobar que la app tiene activo tanto el modo backup como el modo debug cosa que nos ayudara mas adelante con el análisis dinámico.



También hay que comprobar si hay algún elemento exportado o existen providers.

Comprobando la aplicación podemos ver que hay varios elementos exportados



Se puede comprobar que el nivel requerido de permiso de una aplicación externa para poderlo usar es “dangerous”.

Tambien podemos comprobar que en manifest hay declarado un provider en el manifest



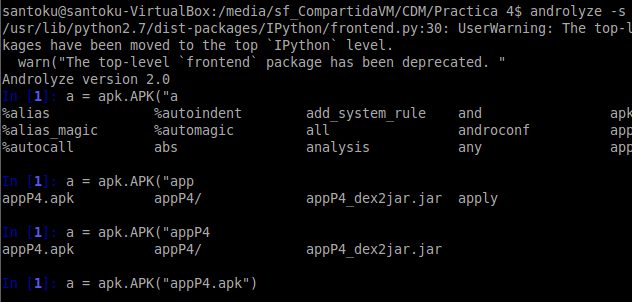


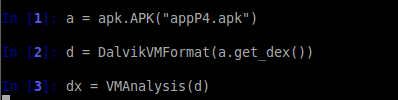
(\* Nota Es la misma línea que ha he partido en 2 capturas para que sea legible \*)

Y se puede comprobar que el provider esta exportado por lo cual puede ser utilizado por cualquier aplicación con un permiso de “dangerous”.

Ahora vamos ha proceder ha comprobrobar los permisos utilizados realmente por la aplicación mediante el androlyze

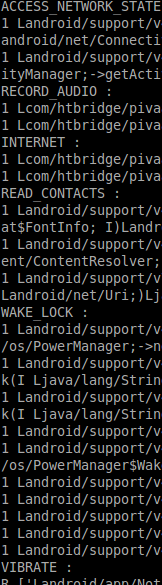
Para eso primero tenemos que abrir el programa y cargar los datos del programa





Tras esto buscaremos las llamadas que utilizan permisos y su localización en las clases de la app

Como se puede Observar en esta imagen de abajo



Hay llamadas a:

ACCES\_NEXTWORK\_STATE, RECORD\_AUDIO, INTERNET, READ\_CONTACTS, WAKE\_LOCK, VIBRATE y a ACCES\_FINE\_LOCATION



De los cuales estos 4 son los mas importantes e interesantes ya que aparecen en el manifest.

-RECORD\_AUDIO

- INTERNET

- READ\_CONTACTS

-ACCES\_FINE\_LOCATION

De estos cuatro solo RECORD\_AUDIO y INTERNET son usados por métodos y de la aplicación ya que READ\_CONTACTS y ACCESS\_FINE\_LOCATION solo son utilizados por librerias Android (fuera del análisis actual).

Esto nos indica que hay muchos permisos declarados en el manifest que no son utilizados por la aplicación siendo un posible lugar de vulnerabilidad los permisos declarados y no utilizados son:

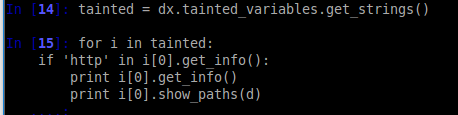
GEST\_ACCOUNTS,READ\_PROFILE,WRITE\_EXTERNAL\_STORAGE,READ\_EXTERNAL\_STORAGE;ACCESS\_COARSE\_LOCATION,NFC,CALL\_PHONE y CAMERA.

Los metodos que utilizan REDORD\_AUDIO y INTERNET que no sean de librerias son:

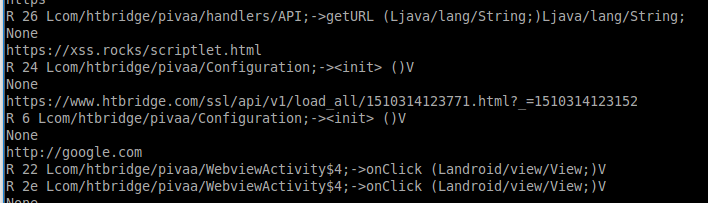




Ahora vamos ha ver donde se hacen llamadas http dentro de la aplicación (ignorando las llamadas de las librerias Google o Android) con el siguiente comando de androlyze



Esto nos da el siguiente resultado

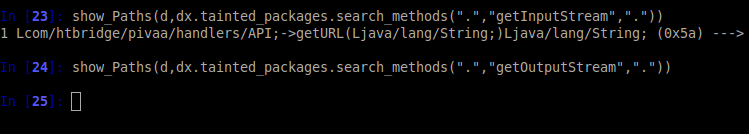


De los cuales en webviewActivity se utiliza en esta parte del código



Ahora comprobaremos las conexiones para Sockets o URL

Con una búsqueda similar obtenemos que el record\_audio se utiliza en la aplicación en el servicio de VulnerableService en los métodos StartRecording y getOutPutFile.

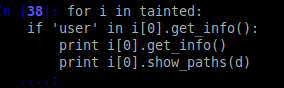


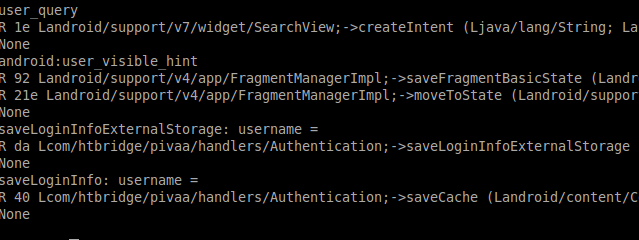


(NOTA \* LA imagen esta partida por la mitad, la línea con la flecha es una línea continua es decir “Lcom/htbridge/pivaa/handlers/API;->getURL(Ljava/lang/String;)Ljava/lang/String; (0x5a) ---> Ljava/net/HttpURLConnection;->getInputStream()Ljava/io/InputStream;” \*)

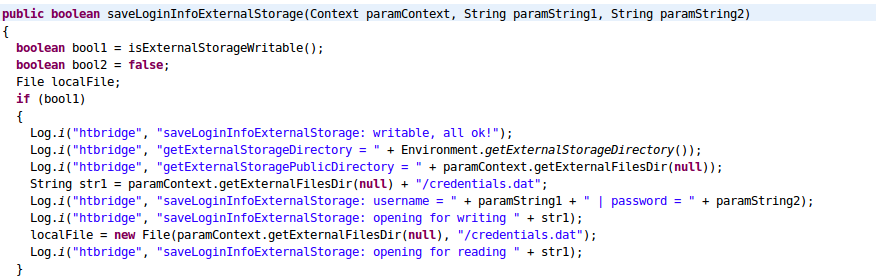
Como se puede comprobar hay otro método que utiliza internet en la clase API.

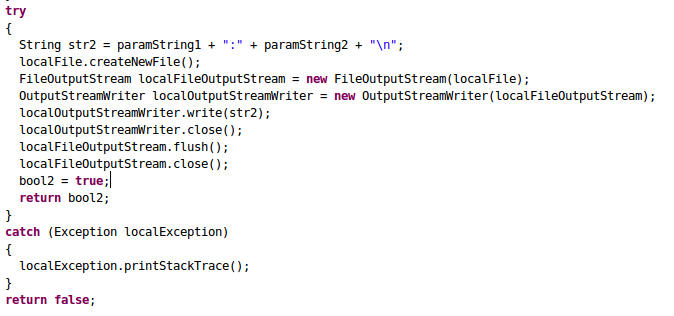
También realizando una búsqueda genérica por string para user nos encontramos con lo siguiente.





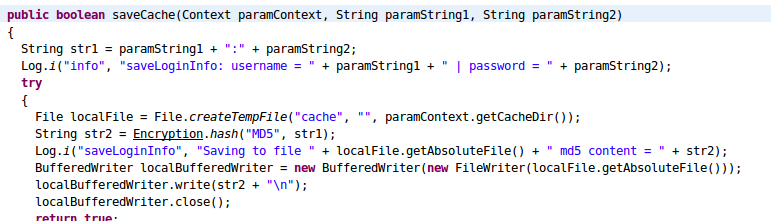
Si buscamos dichos métodos nos encontramos con lo siguiente





Con lo que se puede ver que hay un montón de log que proporcionan información y al estar el modo debug de aplicación encendido nos permitirá acceder a dichos log que están poniendo información sensible, además inspeccionando el código podemos ver que se gurdan las credenciales en un almacenamiento externo (SD) que es accesible para todas las aplicaciones siendo estos 2 riesgos de vulnerabilidades grandes.

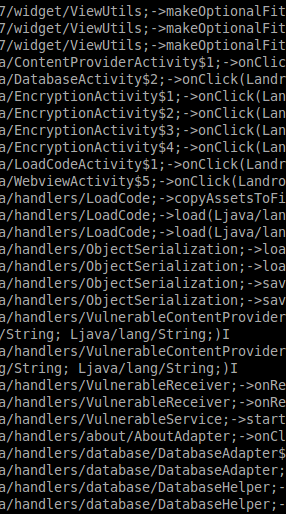
También vemos en el método saveCache que aparte de haber mas logs printando información sensible, la aplicación también guarda las credenciales en la chache como se puede ver abajo.



Ahora vamos ha comprobar los Log que tiene la aplicación con el siguiente método



Esto nos da una cantidad enorme de Log (entre los que se encuentran varios log de las librerias) dada la cantidad de log es imposible (por tiempo) comprobar todos los log



Esta es una lista “reducida e incompleta” de todos los log de la aplicación que recordemos al estar el modo debug encendido la aplicación al ejercutarla en el análisis dinámico podremos ver lo que dicen dichos logs, también me gustaría destacar varias clases interesantes con logs como son EncryptionActivity, ContentProviderActivity, VulnerableReceiver y VulnerableService, a parte del visto anteriormente de Athentification.

## RESUMEN ANALISIS ESTATICO

Con Esto podemos Concluir el análisis Estático, que resumiendo, aparecen actividades exportadas aunque con un nivel de seguridad de “dangerous” puede ser fuente de problemas, así mismo también la aplicación solicita muchos permisos que no son utilizados, dando lugar a un posible foco de vulnerabilidad de acceso a recursos sensibles, también hay que comentar que la mayoría de conexiones ha internet son http, es decir vulnerables a un ataque man in the middle, la aplicación guarda información sensible tanto en el cache como en la SD como pueden ser las credenciales y la gran cantidad de log activos que se encuentran en la aplicación y recordamos con el modod debug activado, que nos permite leer la información de los logs en el análisis dinámico.

# ANALISIS DINAMICO

Lo primero que hemos hecho es Leer los log con el comando adb logcat y hemos probado ha poner unas credenciales “random” para ver como se comportaba los log y que información daban, esto ha dado lugar ha varios fallos de seguridad.

El primero es que no ha comprobado si mis credenciales son validas y me ha autorizado sin ningún problema



Segundo como se puede ver mi usuario es “SoyUnUsuarioCabroncete” y la contraseña es “PeroMuyCabron” siendo una fuente de filtración de datos importante.

Tercero ha guardado mis credenciales tanto en cache como en almacenamiento Interno

(IE)



(CACHE)



Probando ha analizar la fila del cache obtenemos el siguiente resultado



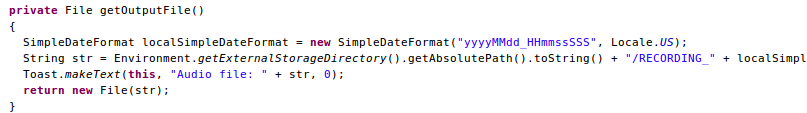
(\*NOTA EL comando es CAT cache…\*)

Como se puede ver el contenido esta cifrado

Ahora vamos ha comprobar el VulnerableService que localizamos en el análisis estático que es una actividad exportada y accede al RECORD\_AUDIO .



Como se puede comprobar podemosacceder al servicio y tal como vimos en el análisis estático el servicio esta grabado un audio que lo deposita en la raiz del almacenamiento externo



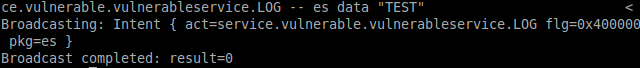
Y tal como lo refleja los log





Esta grabando.

Ahora vamos ha intentar con el VulnerableReceiver



Como podemos comprobar también lo podemos ejecutar externamente gracias ha que esta exportado en el manifest

Estoy teniendo problemas con el whiresarck que no me detecta ninguna interfaz y por tanto no puedo comprobar si las peticiones http son en texto plano o cifradas.