CÁTEDRA DE CIBERSEGURIDAD CIBERUGR, INCIBE-UGR

Nombre	Back It Up
Categoría	Misc (Forense + Crypto)
Dificultad	Difícil
Puntos	500

DESCRIPCIÓN DEL RETO

Se ha podido recuperar un archivo de respaldo de un dispositivo Android sospechoso, con trazas muy raras de mensajería. Parece que el dueño del dispositivo fue muy vago al proteger su móvil con una contraseña muy débil. ¡Recupera la flag oculta dentro de este archivo para resolver el reto!

WRITEUP

1. El reto empieza con un fichero de *backup* de Android y un directorio con un *dump* de la partición de *system* del dispositivo.

2. Lo primero que intentamos es recuperar los datos del fichero de backup. Para ello, podemos usar un dispositivo físico o emulado de Android, u obtener un

dump de los ficheros restaurados en nuestro host. Usaremos esta última ya que resulta más cómodo y accesible que la otra forma. Haremos uso de la herramienta Android Backup Extractor (https://github.com/nelenkov/android-backup-extractor) para restaurar el fichero de backup.

```
java -jar abe-62310d4.jar unpack backup.ab backup.tar
```

```
┌──(kali㉿kali)-[/tmp/back_it_up]
└─$ java -jar abe-62310d4.jar unpack backup.ab backup.tar
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
This backup is encrypted, please provide the password
Password:
Exception in thread "main" java.lang.RuntimeException: javax.crypto.BadPaddingException: Given final
        at org.nick.abe.AndroidBackup.extractAsTar(AndroidBackup.java:239)
        at org.nick.abe.Main.main(Main.java:40)
Caused by: javax.crypto.BadPaddingException: Given final block not properly padded. Such issues can a
        at java.base/com.sun.crypto.provider.CipherCore.unpad(CipherCore.java:861)
        at java.base/com.sun.crypto.provider.CipherCore.fillOutputBuffer(CipherCore.java:941)
        at java.base/com.sun.crypto.provider.CipherCore.doFinal(CipherCore.java:734)
        at java.base/com.sun.crypto.provider.AESCipher.engineDoFinal(AESCipher.java:446)
        at java.base/javax.crypto.Cipher.doFinal(Cipher.java:2251)
        at org.nick.abe.AndroidBackup.extractAsTar(AndroidBackup.java:139)
        ... 1 more
```

3. Nos damos cuenta de que el fichero de backup está encriptado con una contraseña, que corresponde con la contraseña de acceso al dispositivo que el usuario ha establecido (Back up or restore data on your Android device - Android Help). Echamos un vistazo a los ficheros que tenemos de *system* y nos encontramos con tres ficheros clave: *device_policies.xml*, *password.key*, *locksettings.db*.

```
┌──(kali㉿kali)-[/tmp/back_it_up]
└─$ ls system
appops.xml        called_pre_boots.dat  entropy.dat             ifw              job              packages.list        password.key    recent_tasks         sync           users
batterystats.bin  device_policies.xml   framework_atlas.config  inputmethod      locksettings.db  packages.xml         procstats       registered_services  uiderrors.txt
cache             dropbox               gesture.key             install_sessions netstats         package-usage.list   recent_images   seapp_hash           usagestats
```

4. Leemos el primero de ellos que nos dice la política de seguridad de las contraseñas de acceso al dispositivo. De esta política podemos saber exactamente como es la contraseña usada por el usuario, concretamente una contraseña de 5 letras del alfabeto en minúsculas.

```
cat system/device_policies.xml
```

```
┌──(kali㉿kali)-[/tmp/back_it_up]
└─$ cat system/device_policies.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<policies setup-complete="true">
<active-password quality="262144" length="5" uppercase="0" lowercase="5" letters="5" numeric="0" symbols="0" nonletter="0" />
</policies>
```

5. Luego, echamos un vistazo al fichero *locksettings.db* que es una base de datos SQLite que contiene una tabla *locksettings*.

```
sqlite3 locksettings.db
```

```
┌──(kali㉿kali)-[/tmp/back_it_up/system]
└─$ sqlite3 locksettings.db
SQLite version 3.46.1 2024-08-13 09:16:08
Enter ".help" for usage hints.
sqlite> .tables
android_metadata  locksettings
sqlite>
```

6. Si vemos todos los registros de la tabla conseguimos parámetros importantes del cifrado de la clave de acceso, como lo es la salt del hash usado.

```
select * from locksettings;
```

```
┌──(kali㉿kali)-[/tmp/back_it_up/system]
└─$ sqlite3 locksettings.db
SQLite version 3.46.1 2024-08-13 09:16:08
Enter ".help" for usage hints.
sqlite> .tables
android_metadata  locksettings
sqlite> select * from locksettings;
2|lockscreen.disabled|0|1
3|migrated|0|true
4|migrated_user_specific|0|true
5|lockscreen.enabledtrustagents|0|
6|lockscreen.password_salt|0|6675990079707233028
12|lock_pattern_autolock|0|0
14|lockscreen.password_type_alternate|0|0
15|lockscreen.password_type|0|262144
16|lockscreen.passwordhistory|0|
```

7. Finalmente, inspeccionamos la contraseña usada en el fichero *password.key* que vemos que es un hash.

```
┌──(kali㉿kali)-[/tmp/back_it_up/system]
└─$ cat password.key
E135432C47718760B2FD7AF5CFF7A7608A926ED6B5515B7D0DB34FF62F5C388A88B1665C
```

8. Los hashes en Android para la clave de acceso concatenan el hash de la contraseña en SHA1 con el hash de la contraseña en MD5. Por tanto, obtenemos los últimos 32 caracteres como el hash MD5 de nuestra contraseña.

```
h = “HASH”[-32:].lower()
```

```
>>> "E135432C47718760B2FD7AF5CFF7A7608A926ED6B5515B7D0DB34FF62F5C388A88B1665C"[-32:].lower()
'b5515b7d0db34ff62f5c388a88b1665c'
```

9. La salt que hemos visto antes la concatenamos con un ‘:’ y en hexadecimal a nuestro hash.

```
salt = hex(SALT_DECIMAL)[2:]

passw = “:”.join([h, salt])
```

```
>>> salt = hex(6675990079707233028)[2:]
>>> salt
'5ca5e19b48fb3b04'
>>> h
'b5515b7d0db34ff62f5c388a88b1665c'
>>> ":".join([h, salt])
'b5515b7d0db34ff62f5c388a88b1665c:5ca5e19b48fb3b04'
```

10. Lo guardamos en un fichero *passwd.hash*.

```
f = open("passwd.hash", "w")

f.write(passw)

f. close()
```

```
>>> passw = "b5515b7d0db34ff62f5c388a88b1665c:5ca5e19b48fb3b04"
>>> f = open("passwd.hash", "w")
>>> f.write(passw)
49
>>> f.close()
>>>
```

11. Utilizamos la herramienta *hashcat* para crackear la contraseña por fuerza bruta según la política que se utiliza (5 letras del alfabeto minúsculas) para obtener la contraseña.

```
hashcat -m 10 -a 3 passwd.hash ?l?l?l?l?l
```

```
b5515b7d0db34ff62f5c388a88b1665c:5ca5e19b48fb3b04:dycpr

Session..........: hashcat
Status...........: Cracked
Hash.Mode........: 10 (md5($pass.$salt))
Hash.Target......: b5515b7d0db34ff62f5c388a88b1665c:5ca5e
```

12. Intentamos de nuevo extraer los ficheros del backup con la contraseña dada.

```
java -jar abe-62310d4.jar unpack backup.ab backup.tar
```

```
┌──( kali ㉿ kali )-[ /tmp/back_it_up ]
└─$ java -jar abe-62310d4.jar unpack backup.ab backup.tar
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
This backup is encrypted, please provide the password
Password: → dycpr
Calculated MK checksum (use UTF-8: false): 35084DBA509F4120EDC815D7690F08604C45453918A40B40E3DFF382AFFD6A93
0% 1% 2% 3% 4% 5% 6% 7% 8% 9% 10% 11% 12% 13% 14% 15% 16% 17% 18% 19% 20% 21% 22% 23% 24% 25% 26% 27% 28% 29%
55% 56% 57% 58% 59% 60% 61% 62% 63% 64% 65% 66% 67% 68% 69% 70% 71% 72% 73% 74% 75% 76% 77% 78% 79% 80% 81% 8
20725760 bytes written to backup.tar.
```

13. Esto nos da un fichero *backup.tar* que podemos extraer y que nos dará los ficheros restaurados.

```
tar xvf backup.tar
```

```
┌──(kali㉿kali)-[/tmp/back_it_up]
└─$ tar xvf backup.tar
apps/android/_manifest
apps/android/r/wallpaper_info.xml
apps/com.android.browser/_manifest
apps/com.android.browser/r/app_appcache/
apps/com.android.browser/db/browser2.db-shm
apps/com.android.browser/db/browser2.db-wal
apps/com.android.browser/db/browser2.db
apps/com.android.browser/sp/WebViewChromiumPrefs.xml
apps/com.android.browser/sp/com.android.browser_preferences.xml
apps/com.android.calculator2/_manifest
apps/com.android.calendar/_manifest
apps/com.android.calendar/sp/calendar_alerts.xml
apps/com.android.calendar/sp/_has_set_default_values.xml
apps/com.android.calendar/sp/com.android.calendar_preferences.xml
apps/com.android.camera2/_manifest
apps/com.android.captiveportallogin/_manifest
```

14. Los ficheros extraídos son dos directorios *apps* y *shared* que contienen información de distintas apps.

```
┌──(kali㉿kali)-[/tmp/back_it_up]
└─$ ls
abe-62310d4.jar  apps  backup.ab  backup.tar  shared  system
```

15. Entre todas las apps nos interesa la app de Whatsapp. Si nos vamos a *shared/0/Whatsapp/Databases* nos encontramos con una base de datos de los mensajes de Whatsapp encriptados por *crypt14*.

```
┌──(kali㉿kali)-[/tmp/…/shared/0/WhatsApp/Databases]
└─$ ls
msgstore.db.crypt14

┌──(kali㉿kali)-[/tmp/…/shared/0/WhatsApp/Databases]
└─$ file msgstore.db.crypt14
msgstore.db.crypt14: data

┌──(kali㉿kali)-[/tmp/…/shared/0/WhatsApp/Databases]
└─$
```

16. Crypt14 es un cifrado usado por Whatsapp para encriptar ficheros sensibles como lo son los mensajes de chats. La clave de este fichero encriptado se encuentra en *apps/com.whatsapp/f/key*.

```
┌──(kali㉿kali)-[/tmp/…/shared/0/WhatsApp/Databases]
└─$ ls ../../../../apps/com.whatsapp/f
app_state   backup_token   cldr_strings_1663099321.pack   downloadable   key   me                      statistics.json   ViewOnce      wamprivatestats.wam   wam.wam          'WhatsApp Video'
Avatars     biz_directory  decompressed                    invalid_numbers  Logs  network_statistics.json  Stickers          wamdit3.wam   wamrealtime.wam       'WhatsApp Images'
```

17. Para poder desencriptar la base de datos usamos la herramienta wadecrypt de wa-crypt-tools (https://github.com/ElDavoo/wa-crypt-tools).

```
wadecrypt ../../../../apps/com.whatsapp/f/key msgstore.db.crypt14 msgstore.db
```

```
┌──(.tmp)─(kali㉿kali)-[/tmp/…/shared/0/WhatsApp/Databases]
└─$ wadecrypt ../../../../apps/com.whatsapp/f/key msgstore.db.crypt14 msgstore.db
key14.py:128    : [I] Crypt12/14 key loaded
wadecrypt.py:273        : [I] Done
```

18. Accedemos con SQLite a la base de datos y consultamos los registros de la tabla *message*, donde se encontrará la flag.

```
sqlite3 msgstore.db “select * from message;”
```