



Curso de Hacking Ético Escuela de Videojuegos MasterD

Ejercicio 20

Telefonía Móvil

Alumno: Julián Gordon

Índice

Introducción	3
Descubrimiento de IP de nuestro objetivo	5
Descubriendo puertos y servicios abiertos	6
Conexión entre Kali Linux y Android	10
Descubrimos la marca y modelo del dispositivo	11
Listado de aplicaciones instaladas en el dispositivo	12
Aplicaciones en ejecución en el dispositivo	15
Servicios en ejecución en el dispositivo	17
Permisos potenciales de aplicaciones	19
Filtrar la búsqueda de permisos	21
Conclusiones	23

Introducción

En este trabajo práctico, llevaremos a cabo un ejercicio de descubrimiento e investigación sobre un dispositivo móvil Android, virtualizado en nuestro laboratorio de Virtualbox, utilizando herramientas de pentesting desde nuestra máquina de Kali Linux. El objetivo principal es obtener información detallada del dispositivo Android objetivo, incluida su dirección IP, puertos y servicios abiertos, marca y modelo, lista de aplicaciones instaladas y los permisos potenciales de estas aplicaciones.

Para lograr esto, se utilizarán herramientas como arp-scan para descubrir la dirección IP, Nmap para escanear puertos y servicios abiertos, y comandos de ADB para obtener detalles sobre el dispositivo y sus aplicaciones.

ADB (Android Debug Bridge) es una herramienta de línea de comandos que forma parte del kit de desarrollo de Android (SDK). Se utiliza principalmente para la comunicación entre un dispositivo Android y un ordenador host (Kali Linux en este caso) para realizar una variedad de tareas, como depuración, instalación de aplicaciones, acceso a la consola del dispositivo, copia de archivos entre la computadora y el dispositivo, etc.

A lo largo de este informe, se detallarán los pasos seguidos, los comandos utilizados y los resultados obtenidos, con el objetivo de proporcionar una evaluación completa y detallada del dispositivo móvil Android.

Descubrimiento de IP de nuestro objetivo

Para la realización de este ejercicio, empezaremos por descubrir la IP de nuestra máquina objetivo, en este caso la máquina de Android de nuestro laboratorio, desde nuestra máquina de Kali Linux. Para ello utilizamos el comando 'arp-scan --localnet'. Esto nos devolverá las IPs de las máquinas que tenemos en nuestra red. Podemos observar en la siguiente imagen que la IP de la máquina de Android es 10.0.2.20 y la nuestra es 10.0.2.16.

```
(root@kali)-[/home/kali]
# arp-scan --localnet

Interface: eth0, type: EN10MB, MAC: 08:00:27:1D:0A:00, IPv4: 10.0.2.16
WARNING: Cannot open MAC/Vendor file ieee-oui.txt: Permission denied
WARNING: Cannot open MAC/Vendor file mac-vendor.txt: Permission denied
Starting arp-scan 1.10.0 with 256 hosts (https://github.com/royhills/arp-scan)
10.0.2.1      (Unknown: locally administered)
10.0.2.2      (Unknown: locally administered)
10.0.2.3      (Unknown)
10.0.2.20     (Unknown)

4 packets received by filter, 0 packets dropped by kernel
Ending arp-scan 1.10.0: 256 hosts scanned in 1.884 seconds (135.88 hosts/sec). 4 responded
```

Descubriendo puertos y servicios abiertos

Ahora que ya sabemos la IP de nuestra máquina objetivo, lo que haremos será un escaneo de puertos con la herramienta NMAP. Para ello utilizamos el siguiente comando que realizará un escaneo exhaustivo de nuestro objetivo.

```
'nmap -sS -sV -sC -p 1-65535 10.0.2.20'
```

Ahora explicaremos en detalle los argumentos del comando.

-sS: Este es el indicador para realizar un escaneo de tipo SYN stealth. En este tipo de escaneo, Nmap envía un paquete SYN al puerto de destino y espera una respuesta. Si recibe una respuesta SYN/ACK, significa que el puerto está abierto. Si recibe un RST, indica que el puerto está cerrado.

-sV: Habilita la detección de versiones de servicios. Nmap intentará determinar las versiones de los servicios que se ejecutan en los puertos abiertos encontrados durante el escaneo.

-sC: Habilita el escaneo con scripts predeterminados. Nmap tiene una variedad de scripts integrados que realizan tareas específicas, como detección de vulnerabilidades, enumeración de servicios, entre otros. Este indicador ejecuta estos scripts predeterminados contra los puertos abiertos encontrados durante el escaneo.

-p 1-65535: Especifica el rango de puertos que se escanearán. En este caso, se están escaneando todos los puertos posibles, desde el puerto 1 hasta el 65535.

10.0.2.20: Esta es la dirección IP del objetivo del escaneo. Nmap realizará el escaneo en esta dirección IP específica.

```
(root@kali)-[/home/kali]
# nmap -sS -sV -sC -p 1-65535 10.0.2.20
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-04-21 21:31 CEST
Nmap scan report for 10.0.2.20
Host is up (0.00069s latency).
Not shown: 65534 closed tcp ports (reset)
PORT      STATE SERVICE VERSION
5555/tcp open  adb      Android Debug Bridge device (name: android_x86_64; model: VirtualBox; device: x86_64)
MAC Address: 08:00:27:DC:6B:2A (Oracle VirtualBox virtual NIC)
Service Info: OS: Android; CPE: /o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 27.72 seconds
```

El resultado del comando indica que el puerto TCP 5555 está abierto y ejecutando el servicio ADB (Android Debug Bridge). ADB es una herramienta de línea de comandos que permite comunicarnos con un dispositivo Android conectado.

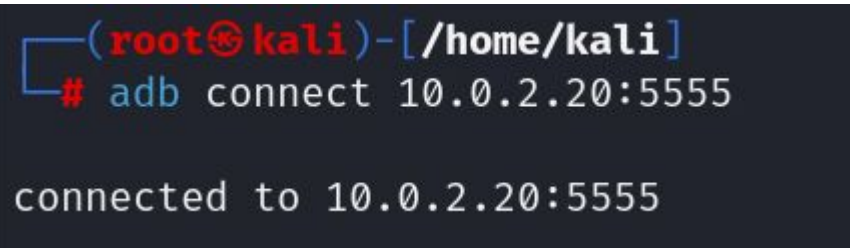
También nos proporciona información adicional sobre el dispositivo:

- "name: android_x86_64": Esto indica el nombre del dispositivo, que en este caso es "android_x86_64".
- "model: VirtualBox": Indica el modelo del dispositivo, qué es "VirtualBox".
- "device: x86_64": Indica el tipo de dispositivo, que es "x86_64".

Conexion entre Kali Linux y Android

Ahora que ya sabemos la IP, los puertos abiertos y los servicios que están corriendo, vamos a intentar conectarnos desde nuestra máquina de Kali Linux a la máquina de Android. Para ello usamos el siguiente comando:

`'adb connect 10.0.2.20:5555'`

A terminal window with a dark background. The prompt is `(root@kali)-[/home/kali]` in red and blue. The command `# adb connect 10.0.2.20:5555` is entered in blue. The output `connected to 10.0.2.20:5555` is shown in white.

```
(root@kali)-[/home/kali]  
# adb connect 10.0.2.20:5555  
  
connected to 10.0.2.20:5555
```

Podemos observar que fue exitosa la conexión.

Descubrimos la marca y modelo del dispositivo

Ahora que ya tenemos una conexión establecida con nuestro objetivo, vamos a investigar la marca y el modelo del dispositivo.

Empezaremos con el siguiente comando que nos dará la marca del dispositivo.

```
'adb shell getprop ro.product.brand '
```

Luego para obtener el modelo usamos este comando:

```
'adb shell getprop ro.product.model'
```

Podemos observar en la siguiente imagen que nuestro objetivo no es un dispositivo móvil real, ya que es una máquina virtual de VirtualBox.

Listado de aplicaciones instaladas en el dispositivo

```
(root@kali)-[/home/kali]  
# adb shell getprop ro.product.brand  
Android-x86  
  
(root@kali)-[/home/kali]  
# adb shell getprop ro.product.model  
VirtualBox
```

El siguiente comando que ejecutaremos nos dará un listado con las aplicaciones instaladas en el dispositivo. El comando será :

‘adb shell pm list packages’

Podemos observar el resultado en las siguientes imágenes.

```
(root@kali)-[/home/kali]  
# adb shell pm list packages
```

```
package:com.google.android.youtube  
package:com.example.android.rssreader  
package:com.android.providers.telephony  
package:org.android_x86.analytics  
package:com.google.android.googlequicksearchbox  
package:com.android.providers.calendar  
package:com.android.providers.media  
package:com.google.android.onetimeinitializer  
package:com.android.wallpapercropper  
package:com.android.documentsui  
package:com.android.galaxy4  
package:com.mypermissions.mypermissions  
package:com.android.externalstorage  
package:com.android.htmlviewer  
package:com.android.mms.service  
package:com.android.providers.downloads  
package:com.android.browser  
package:com.google.android.configupdater  
package:com.android.soundrecorder  
package:com.android.defcontainer  
package:com.android.providers.downloads.ui  
package:com.android.vending  
package:com.android.pacprocessor  
package:com.joeykrim.rootcheck  
package:com.android.certinstaller  
package:com.android.carrierconfig  
package:com.belarc.securityadvisor  
package:android  
package:com.android.contacts  
package:com.android.camera2  
package:com.android.launcher3
```

```
package:com.android.backupconfirm  
package:com.android.statementservice  
package:com.google.android.gm  
package:com.overlook.android.fing  
package:com.android.wallpaper.holospiral  
package:com.android.calendar  
package:com.android.phasebeam  
package:com.google.android.setupwizard  
package:com.android.providers.settings  
package:com.android.sharedstoragebackup  
package:com.android.printspooler  
package:com.android.dreams.basic  
package:com.android.webview  
package:com.android.inputdevices  
package:com.android.server.telecom  
package:com.google.android.syncadapters.contacts  
package:com.example.android.notepad  
package:com.android.keychain  
package:com.android.chrome  
package:com.android.dialer  
package:com.android.gallery3d  
package:com.google.android.gms  
package:com.google.android.gsf  
package:com.android.calllogbackup  
package:com.google.android.partnersetup  
package:com.android.packageinstaller  
package:com.android.basicsmsreceiver  
package:com.svox.pico  
package:com.android.proxyhandler  
package:com.cyanogenmod.filemanager  
package:com.android.inputmethod.latin  
package:com.google.android.feedback  
package:com.google.android.syncadapters.calendar  
package:com.android.managedprovisioning  
package:com.google.android.gsf.login
```

```
package:com.google.android.feedback  
package:com.google.android.syncadapters.calendar  
package:com.android.managedprovisioning  
package:com.google.android.gsf.login  
package:com.android.wallpaper.livepicker  
package:org.mozilla.firefox  
package:jackpal.androidterm  
package:com.google.android.backuptransport  
package:com.android.settings  
package:com.cyanogenmod.eleven  
package:com.android.calculator2  
package:org.android_x86.hardwarecollector  
package:com.android.wallpaper  
package:com.android.vpndialogs  
package:com.android.email  
package:com.android.phone  
package:com.android.shell  
package:com.android.providers.userdictionary  
package:com.android.location.fused  
package:com.android.deskclock  
package:com.android.systemui  
package:com.android.bluetoothmidiservice  
package:com.funnycat.virustotal  
package:com.android.bluetooth  
package:com.android.development  
package:com.android.providers.contacts  
package:com.android.captiveportallogin
```

Aplicaciones en ejecución en el dispositivo

Ahora usamos este comando para ver qué aplicaciones se están utilizando actualmente en el dispositivo.

```
'adb shell dumpsys activity | grep "Proc #"'
```

Para probar esta parte, en el dispositivo, hemos abierto la aplicación de Gmail y la de Youtube. En la siguiente imagen podemos observar este proceso.


```
(root@kali)-[/home/kali]  
# adb shell dumpsys activity | grep "Proc #"
```

```
Proc # 0: fore F/A/T trm: 0 2976:com.google.android.youtube/u0a60 (top-activity)  
Proc #23: vis F/ /SB trm: 0 1630:com.google.android.googlequicksearchbox:interactor/u0a24 (service)  
Proc # 3: vis F/ /SB trm: 0 1970:com.google.android.gms/u0a10 (service)  
Proc # 1: vis F/ /T trm: 0 1650:com.google.android.gms.persistent/u0a10 (service)  
Proc #21: prcp B/ /IB trm: 0 1642:com.android.inputmethod.latin/u0a50 (service)  
Proc #22: svc B/ /S trm: 0 1784:com.cyanogenmod.eleven:main/u0a40 (started-services)  
Proc # 2: home B/ /HO trm: 0 1713:com.android.launcher3/u0a15 (home)  
Proc # 4: prev B/ /LA trm: 0 1907:com.google.android.gm/u0a45 (previous)  
Proc #11: cch B/ /Ca trm: 0 2953:com.android.chrome/u0a66 (cch-client-act)  
Proc # 9: cch B/ /CE trm: 0 1847:com.google.process.gapps/u0a10 (cch-empty)  
Proc # 8: cch B/ /CE trm: 0 3326:android.process.acore/u0a3 (cch-empty)  
Proc # 7: cch B/ /CE trm: 0 3392:com.android.gallery3d/u0a44 (cch-empty)  
Proc # 6: cch B/ /CE trm: 0 1862:com.android.vending/u0a18 (cch-empty)  
Proc # 5: cch B/ /CE trm: 0 2017:com.google.android.googlequicksearchbox:search/u0a24 (cch-empty)  
Proc #15: cch+2 B/ /CE trm: 0 2504:com.android.calendar/u0a33 (cch-empty)  
Proc #14: cch+2 B/ /CE trm: 0 2851:com.android.email/u0a41 (cch-empty)  
Proc #13: cch+2 B/ /CE trm: 0 2564:com.android.providers.calendar/u0a2 (cch-empty)  
Proc #12: cch+2 B/ /CE trm: 0 3235:com.google.android.gms.unstable/u0a10 (cch-empty)  
Proc #10: cch+2 B/ /CE trm: 0 3460:com.android.defcontainer/u0a6 (cch-empty)  
Proc #20: cch+4 B/ /CE trm: 0 3045:com.android.vending:download_service/u0a18 (cch-empty)  
Proc #19: cch+4 B/ /CE trm: 0 2302:org.mozilla.firefox:gpu/u0a68 (service)  
Proc #18: cch+4 B/ /CE trm: 0 2128:org.mozilla.firefox:tab13/u0a68 (service)  
Proc #17: cch+4 B/ /CE trm: 0 1888:org.mozilla.firefox/u0a68 (cch-empty)  
Proc #16: cch+4 B/ /CE trm: 0 3123:com.overlook.android.fing/u0a65 (cch-empty)
```


Servicios en ejecución en el dispositivo

Ahora usamos este comando para ver qué servicios se están utilizando actualmente en el dispositivo.

```
'adb shell dumpsys activity services'
```

Nos devolverá un listado muy extenso con todos los servicios en ejecución, para filtrar por servicios esenciales que estén en ejecución, podemos usar este comando:

```
'adb shell dumpsys activity services | grep  
"WindowManagerService\|KeyguardService\|PackageInstallerService\|Conne  
ctivityService"'
```

Podemos observar el resultado en la siguiente imagen.

```
(root@kali)-[/home/kali]
# adb shell dumpsys activity services | grep "WindowManagerService\|KeyguardService\|PackageInstallerService\|ConnectivityService"

* ServiceRecord{84d0a83 u0 com.android.systemui/.keyguard.KeyguardService}
  intent={cmp=com.android.systemui/.keyguard.KeyguardService}
  intent={cmp=com.android.systemui/.keyguard.KeyguardService}
    ConnectionRecord{e2e0d32 u0 CR com.android.systemui/.keyguard.KeyguardService:@fe1c13d}
    ConnectionRecord{e2e0d32 u0 CR com.android.systemui/.keyguard.KeyguardService:@fe1c13d}
* ConnectionRecord{e2e0d32 u0 CR com.android.systemui/.keyguard.KeyguardService:@fe1c13d}
  binding=AppBindRecord{7f1f15a com.android.systemui/.keyguard.KeyguardService:system}
```

Permisos potenciales de aplicaciones

Por último, para obtener información sobre los permisos concedidos a las aplicaciones instaladas, usaremos nuevamente un comando de ADB.

En este caso usamos:

`'adb shell dumpsys package packages'`

Nos devolverá un resultado muy extenso, con todos los permisos sobre todas las aplicaciones, podemos observar el comienzo del resultado en la siguiente imagen.

(root@kali)~[/home/kali]

adb shell dumpsys package packages

Packages:

Package [com.google.android.youtube] (453fdb2):

userId=10060

pkg=Package{8bb7103 com.google.android.youtube}

codePath=/data/app/com.google.android.youtube-2

resourcePath=/data/app/com.google.android.youtube-2

legacyNativeLibraryDir=/data/app/com.google.android.youtube-2/lib

primaryCpuAbi=x86_64

secondaryCpuAbi=null

versionCode=1531305408 targetSdk=33

versionName=17.34.35

splits=[base, config.en, config.mdpi, config.x86_64]

applicationInfo=ApplicationInfo{f903880 com.google.android.youtube}

flags=[SYSTEM HAS_CODE ALLOW_CLEAR_USER_DATA UPDATED_SYSTEM_APP ALLOW_BACKUP KILL_AFTER_RESTORE RESTORE_ANY_VERSION LARGE_HEAP]

privateFlags=[]

dataDir=/data/user/0/com.google.android.youtube

supportsScreens=[small, medium, large, xlarge, resizeable, anyDensity]

usesOptionalLibraries:

androidx.window.extensions

androidx.window.sidecar

timeStamp=2024-04-01 11:47:46

firstInstallTime=2017-04-27 10:58:34

lastUpdateTime=2024-04-01 11:48:15

installerPackageName=com.android.vending

signatures=PackageSignatures{49f3eb9 [90042fe]}

installPermissionsFixed=true installStatus=1

pkgFlags=[SYSTEM HAS_CODE ALLOW_CLEAR_USER_DATA UPDATED_SYSTEM_APP ALLOW_BACKUP KILL_AFTER_RESTORE RESTORE_ANY_VERSION LARGE_HEAP]

declared permissions:

com.google.android.youtube.permission.C2D_MESSAGE: prot=signature, INSTALLED

install permissions:

com.google.android.c2dm.permission.RECEIVE: granted=true

android.permission.USE_CREDENTIALS: granted=true

Filtrar la búsqueda de permisos

La herramienta ADB cuenta con distintos comandos que nos pueden ayudar a encontrar información importante. Estos comandos nos permitirán obtener información detallada sobre los permisos de las aplicaciones instaladas en el dispositivo Android de una manera más automatizada y eficiente

El comando que ejecutamos anteriormente, nos devolvió un resultado muy extenso y difícil de leer e identificar. Para ayudarnos con esta tarea, vamos a ver algunos parámetros que podemos utilizar.

El comando: `'adb shell pm list permissions -d -g'`, enumera los permisos peligrosos (Dangerous Permissions) agrupados por categorías de permisos (permission-groups).

```
(root@kali)-[/home/kali]  
# adb shell pm list permissions -d -g
```

Dangerous Permissions:

```
group:com.google.android.gms.permission.CAR_INFORMATION  
permission:com.google.android.gms.permission.CAR_VENDOR_EXTENSION  
permission:com.google.android.gms.permission.CAR_MILEAGE  
permission:com.google.android.gms.permission.CAR_FUEL
```

```
group:android.permission-group.CONTACTS  
permission:android.permission.WRITE_CONTACTS  
permission:android.permission.GET_ACCOUNTS  
permission:android.permission.READ_CONTACTS
```

```
group:android.permission-group.PHONE  
permission:android.permission.READ_CALL_LOG  
permission:android.permission.READ_PHONE_STATE  
permission:android.permission.CALL_PHONE  
permission:android.permission.WRITE_CALL_LOG  
permission:android.permission.USE_SIP  
permission:android.permission.PROCESS_OUTGOING_CALLS  
permission:com.android.voicemail.permission.ADD_VOICEMAIL
```

```
group:android.permission-group.CALENDAR  
permission:android.permission.READ_CALENDAR  
permission:android.permission.WRITE_CALENDAR
```

```
group:android.permission-group.CAMERA  
permission:android.permission.CAMERA
```

```
group:android.permission-group.SENSORS  
permission:android.permission.BODY_SENSORS
```

```
group:android.permission-group.SUPERUSER  
permission:android.permission.ACCESS_SUPERUSER
```

```
group:android.permission-group.LOCATION  
permission:android.permission.ACCESS_FINE_LOCATION  
permission:com.google.android.gms.permission.CAR_SPEED
```

Conclusiones

Durante el desarrollo de este ejercicio, llevamos a cabo una exploración exhaustiva del dispositivo Android objetivo desde nuestra máquina de Kali Linux. A través de una serie de pasos cuidadosamente planificados, se logró obtener información detallada sobre la configuración y la seguridad del dispositivo, así como sobre las aplicaciones instaladas en él.

Inicialmente, descubrimos la dirección IP del dispositivo mediante el uso del comando `'arp-scan --localnet'`, lo que nos permitió identificar su ubicación en la red local. Posteriormente, realizamos un escaneo de puertos utilizando Nmap, revelando que el puerto TCP 5555 estaba abierto y ejecutando el servicio ADB (Android Debug Bridge). Esta información proporcionó una entrada crucial para la conexión exitosa entre la máquina de Kali Linux y el dispositivo Android utilizando el comando `'adb connect.adb shell pm list packages'`, lo que nos brindó una visión general de su entorno de software.

Una vez establecimos la conexión, investigamos la marca y el modelo del dispositivo utilizando comandos de ADB como `adb shell getprop ro.product.brand` y `adb shell getprop ro.product.model`. Además, obtuvimos un listado de las aplicaciones instaladas en el dispositivo y cuales estaban en ejecución actualmente.

Finalmente, exploramos los permisos concedidos a las aplicaciones instaladas utilizando el comando `'adb shell dumpsys package packages'`. Aunque el resultado fue extenso y difícil de analizar, explicamos la utilidad de algunas herramientas de ADB para obtener información detallada sobre la seguridad de las aplicaciones y cómo filtrar la búsqueda de estos resultados.