evil maid on droids

or why you should never loose your android smartphone

@f0rki

2012-12-06

Agenda

evil maids

detour: the android boot process

attack scenarios

- unrooted phones
- adb access
- rooted phones

protecting yourself

Agenda

evil maids

detour: the android boot process

attack scenarios

- unrooted phones
- adb access
- rooted phones

protecting yourself

evil maids

wat?

evil maids

wat?

1. device left at hotel room

evil maids

wat?

1. device left at hotel room
2. maid comes in

evil maids

wat?

1. device left at hotel room
2. maid comes in
3. maid installs malware, fetches data, etc.

evil maids

wat?

1. device left at hotel room
2. maid comes in
3. maid installs malware, fetches data, etc.
4. ???

evil maids

wat?

1. device left at hotel room
2. maid comes in
3. maid installs malware, fetches data, etc.
4. ???
5. PROFIT!!!

targets

- laptop is classic target
- full disk encryption as mitigation

targets

- laptop is classic target
- full disk encryption as mitigation
- modify unencrypted bootloader/kernel

targets

- laptop is classic target
- full disk encryption as mitigation
- modify unencrypted bootloader/kernel
- secure boot as mitigation
  - EFI SecureBoot on x86 PCs/Notebook
  - Reduced access on embedded devices

a new victim arises

a new victim arises

picture: thx sofie <3


STORE ALL THE
12:00

Agenda

evil maids

detour: the android boot process

attack scenarios

- unrooted phones
- adb access
- rooted phones

protecting yourself

partition layout

- `/system`: OS binaries and config, android, framework
- `/data`: user-installed apps, all user data
- `boot`: kernel, fs root `/`
- `recovery`: recovery system
- `cache`: dalvik cache, other cached data
- `/sdcard /mnt/storage`: music, videos, whatever ...

Actual layout depends on device

android boot process

for HTC/Qualcomm devices:

1. baseband processor starts primary boot loader (PBL)

android boot process

for HTC/Qualcomm devices:

1. baseband processor starts primary boot loader (PBL)
2. PBL starts secondary boot loader (SBL)

android boot process

for HTC/Qualcomm devices:

1. baseband processor starts primary boot loader (PBL)
2. PBL starts secondary boot loader (SBL)
3. app processor bootup – `HBOOT` bootloader

android boot process

for HTC/Qualcomm devices:

1. baseband processor starts primary boot loader (PBL)
2. PBL starts secondary boot loader (SBL)
3. app processor bootup – `HBOOT` bootloader
4. HBOOT loads kernel/recovery

security? – locked bootloaders

for HTC/Qualcomm devices:

1. baseband processor starts primary boot loader (PBL)
   verifies signature of sbl
2. PBL starts secondary boot loader (SBL)
   verifies baseband code and HBOOT
3. app processor bootup – `HBOOT` bootloader
4. HBOOT loads kernel/recovery
   verifies signature on kernel/recovery

bootloader unlocking

- disables signature checking/verification in boot process
- allows booting of third-party code $\rightarrow$ yay, custom ROMS!

bootloader unlocking

- disables signature checking/verification in boot process
- allows booting of third-party code → yay, custom ROMS!

- bootloader unlocking
  - using fastboot tool

```
fastboot oem unlock
```

  - usually does factory reset
    - erases `/data/`
    - remove device settings (e.g. saved wifi passwords)
  - might need some proprietary tool or an exploit for unlocking

HTC S-ON/S-OFF

- system, kernel, recovery is hardware-write-protected
- “temp root” – rooted phones will be unrooted at next boot
- bootloader unlocking – S-OFF
  - submit device-specific token
  - flash signed blob
  - voids warranty
- unpublished exploit: `revolutionary`

fastboot and co

- fastboot
  - "standard" protocol from AOSP
  - implemented in app processor bootloader (e.g. HBOOT)
  - can flash images to partitions
  - can directly boot kernels
- other proprietary protocols/tools exist
  - `nvflash` for Tegra devices
  - old Motorola: SBF + miniloader
  - flash images via usb-exported-ramdisk (archos)
  - etc...

Agenda

evil maids

detour: the android boot process

attack scenarios

- unrooted phones
- adb access
- rooted phones

protecting yourself

assumptions

- device has set a PIN/password/pattern
  - else you are totally f**cked anyway
  - face-unlock also sucks
- typical smartphone usage
- google, facebook, twitter account set up
- access to storage device not possible
  - because of encryption
  - hardware protection
  - attacker can't solder ;)

Agenda

evil maids

detour: the android boot process

attack scenarios
- unrooted phones
- adb access
- rooted phones

protecting yourself

prerequisites

- stock ROM
- no adb
- no root

pull sdcard

pull sdcard

how?

- pull sdcard
- dump everything

pull sdcard

how?

- pull sdcard
- dump everything

what?

- personal data (pictures, music)
- apps2sd
  - e.g. `/sdcard/Android/data/`
- app backups

- probably nothing really critical
- company phone – company data???

what about nexus s?

- there's no sdcard!

what about nexus s?

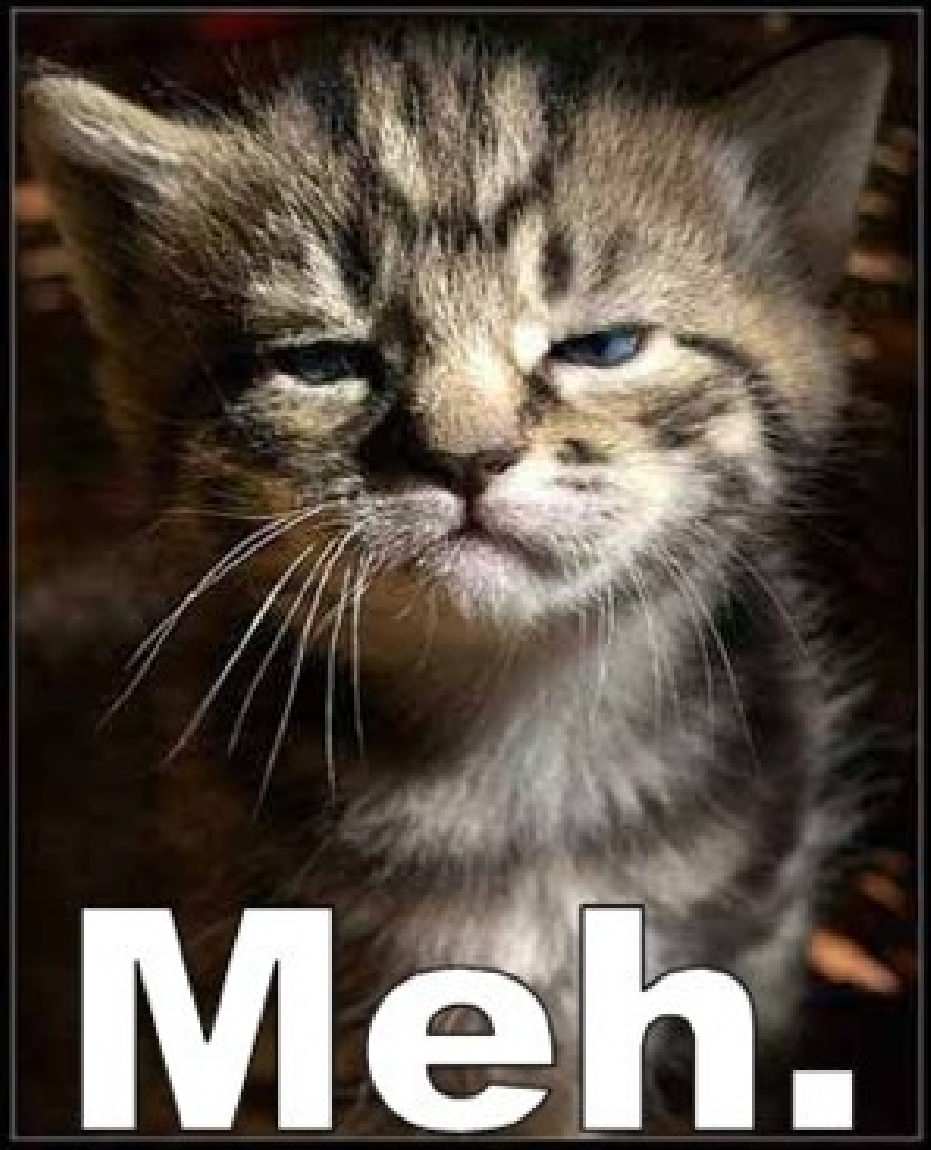
- there's no sdcard!

- only internal storage
- accessible via media transfer protocol (mtp)
  - access only when unlocked
  - restricted access to data

smudge patterns I

smudge patterns II

old news. . . boring stuff. . .



Agenda

evil maids

detour: the android boot process

attack scenarios

- unrooted phones
- adb access
- rooted phones

protecting yourself

prerequisites

- phone used personally and for development
- stock ROM
- no root
- adb enabled

install malware

- create and install malicious app pulling all possible data

```
adb install com.example.AngryBirdsStarTrek.apk
```

install malware

- create and install malicious app pulling all possible data

```
adb install com.example.AngryBirdsStarTrek.apk
```

- still restricted access
  - give malware every possible android permission
  - still no access to most of `/data/`
  - no `system` or `systemOrSignature` level permissions
- pull
  - personal data
  - contacts/texts

disabling keyguard via app

disabling keyguard via app

```
KeyguardManager keyguardManager = (KeyguardManager)
    getSystemService(Context.KEYGUARD_SERVICE);
KeyguardLock mkeyguardLock =
    keyguardManager.newKeyguardLock("unlock");
mkeyguardLock.disableKeyguard();
```

disabling keyguard via app

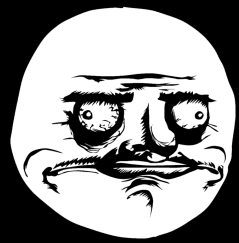
```
KeyguardManager keyguardManager = (KeyguardManager)
    getSystemService(Context.KEYGUARD_SERVICE);
KeyguardLock mkeyguardLock =
    keyguardManager.newKeyguardLock("unlock");
mkeyguardLock.disableKeyguard();
```

- hitting back/home button might enable keyguard again
  - depending on the device and the rom
  - might also get you to launcher activity (=win!)

disabling keyguard via app

```
KeyguardManager keyguardManager = (KeyguardManager)
    getSystemService(Context.KEYGUARD_SERVICE);
KeyguardLock mkeyguardLock =
    keyguardManager.newKeyguardLock("unlock");
mkeyguardLock.disableKeyguard();
```

- hitting back/home button might enable keyguard again
  - depending on the device and the rom
  - might also get you to launcher activity (=win!)
- solution: launch other activities/intents via our malicious app
  so no problem ;)



intercepting login credentials

1. install custom ca cert
2. set proxy in network settings
3. launch intercepting proxy
4. grab stuff
   - google auth token
   - facebook token, password
   - etc.

intercepting login credentials

1. install custom ca cert
2. set proxy in network settings
3. launch intercepting proxy
4. grab stuff
   - google auth token
   - facebook token, password
   - etc.

- no cert errors, since we installed a trusted CA cert
- unfortunately not everything uses system proxy
  - gapps, facebook work fine

grabbing google auth token

using the `mitmproxy` tool

```
2012-11-29 22:14:01 POST https://android.clients.google.com/auth

Request intercepted                                    Response
Content-Type:     application/x-www-form-urlencoded
Content-Length:   718
Host:             android.clients.google.com
Connection:       Keep-Alive
User-Agent:       GoogleLoginService/1.2 (A101 MR1)
URLEncoded form
accountType:       HOSTED_OR_GOOGLE
Email:             [redacted]@gmail.com
has_permission:    1
Token:             1/Ls[redacted]
service:           oauth2:https://www.googleapis.com/auth/plus.me
                   https://www.googleapis.com/auth/plus.stream.read
                   https://www.googleapis.com/auth/plus.stream.write
                   https://www.googleapis.com/auth/plus.circles.write
                   https://www.googleapis.com/auth/plus.circles.read
                   https://www.googleapis.com/auth/plus.photos.readwrite
source:            android
androidId:         3e[redacted]726
app:               com.google.android.apps.plus
client_sig:        38918[redacted]ed5788
device_country:    at
operatorCountry:   at
lang:              en
RefreshServices:   1
```

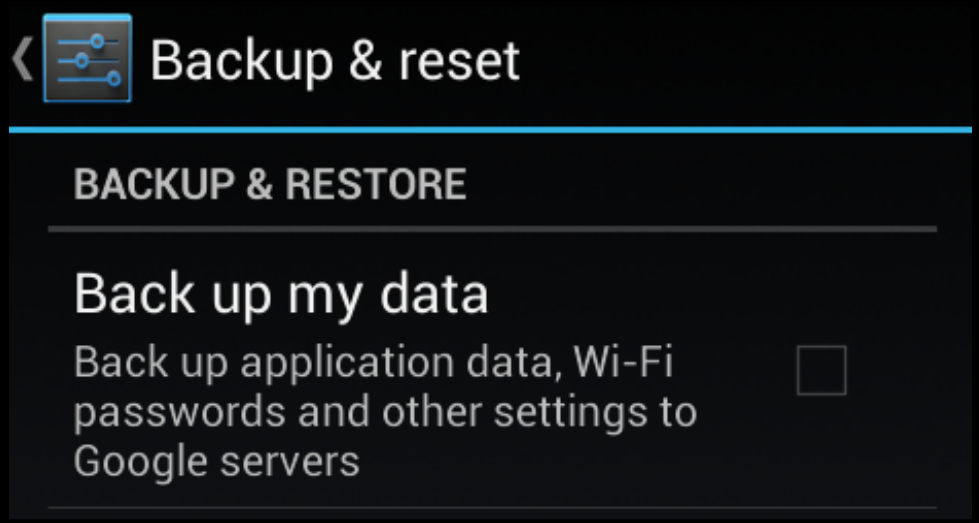

im in ur
sniffin ur paketz

google backups

- so we have the google auth token

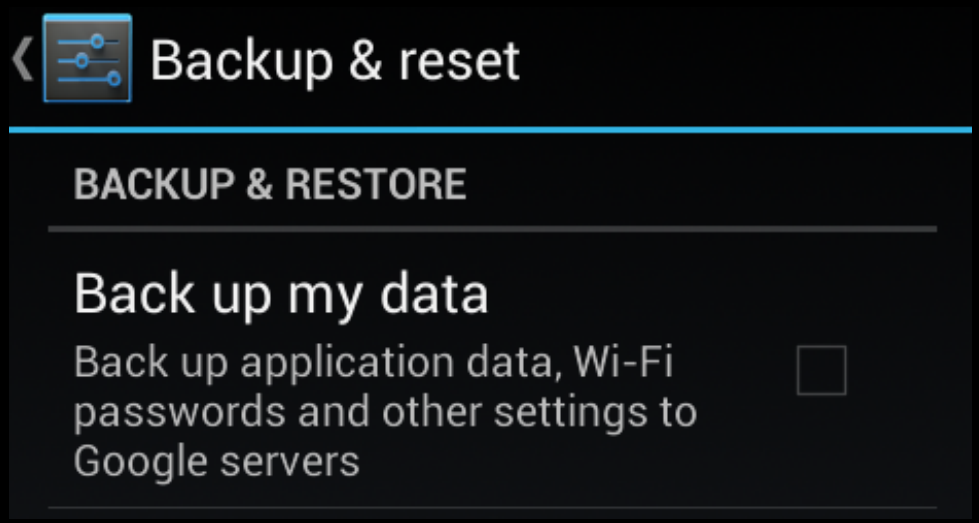
google backups

- so we have the google auth token



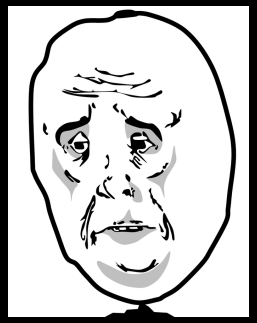
google backups

- so we have the google auth token

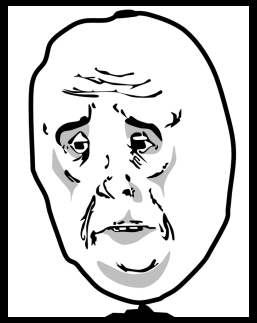


- adding auth token to rooted phone
  $\rightarrow$ provides access to everything backed up to google (in plaintext)

so still no root. . .

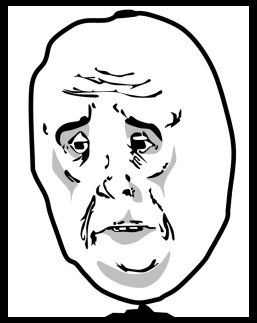


so still no root. . .

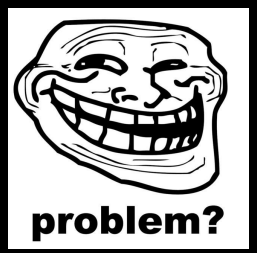


- well. . .

so still no root. . .



- well. . . get root!
  - root via adb restore by Bin4ry (for Android 4.0 and 4.1)
  - mempodroid
  - ZergRush
  - Gingerbreak
  - . . .



Agenda

evil maids

detour: the android boot process

attack scenarios

- unrooted phones
- adb access
- rooted phones

protecting yourself

prerequisites

- rooted phone
- custom ROM, recovery
- adb access

well. . .

. . . you are **totally** screwed!

well. . .

. . . you are **totally** screwed!

the attack

```
adb pull /data/data/
adb pull /system/data/
```

the attack

```
adb pull /data/data/
adb pull /system/data/
```

- credentials
- wifi passwords
- all data
- install malware/rootkits for future use

the attack

```
adb pull /data/data/
adb pull /system/data/
```

- credentials
- wifi passwords
- all data
- install malware/rootkits for future use

prerequisites

- rooted phone
- custom ROM, custom recovery
- **no** adb access

no adb access

- ok so no adb access

no adb access

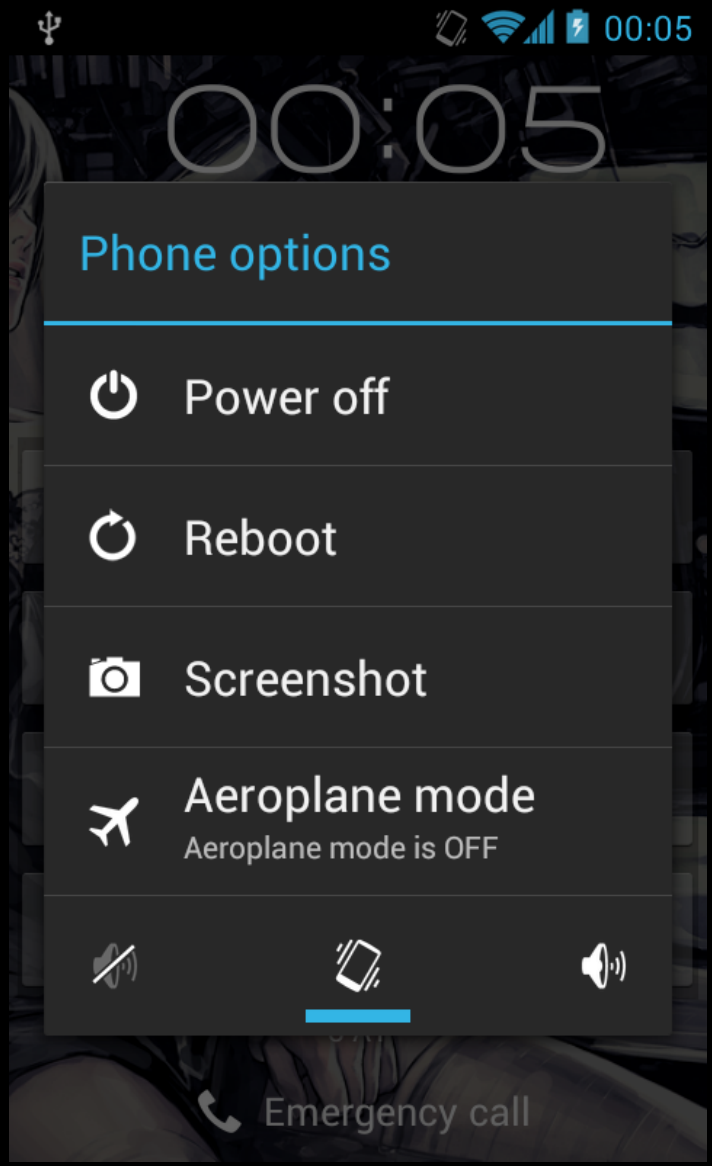
- ok so no adb access
- but custom recovery (e.g. clockworkmod)

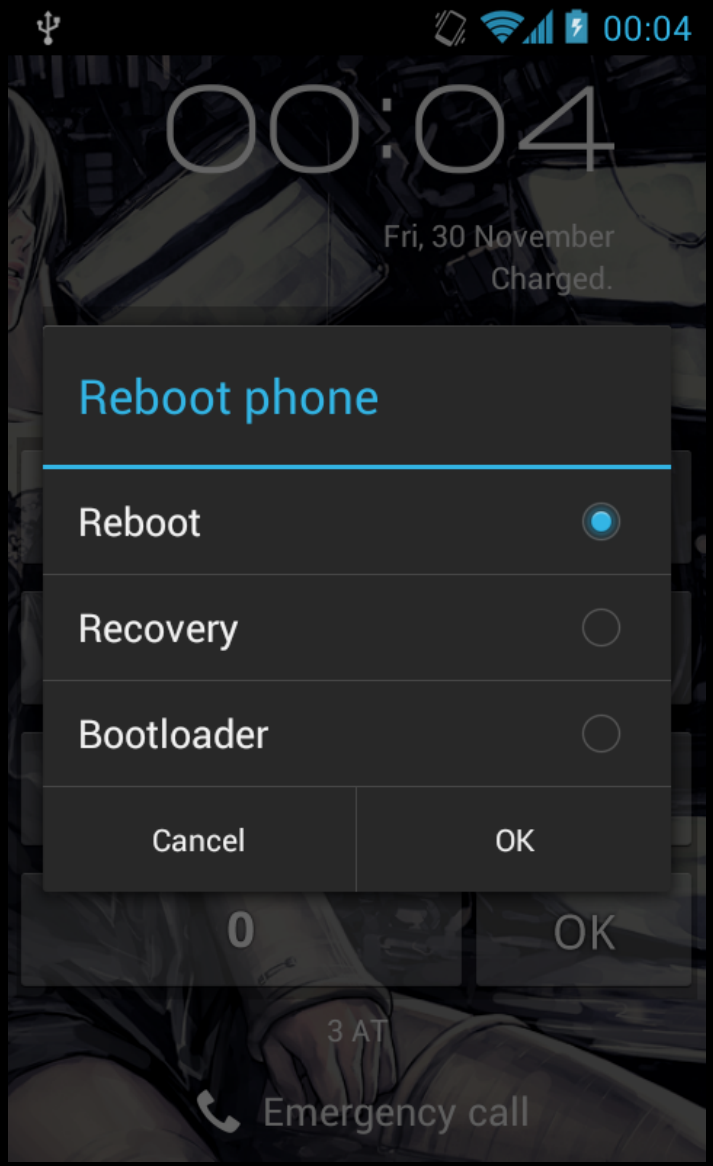
no adb access

- ok so no adb access
- but custom recovery (e.g. clockworkmod)

- remember the bootloader stuff?
- bootloader is usually unlocked
- we can boot/execute arbitrary code :)

reboot menu


00:05
00:05
Phone options
Power off
Reboot
Screenshot
Aeroplane mode
Aeroplane mode is OFF
Emergency call


00:04
00:04
Fri, 30 November
Charged.
Reboot phone
Reboot
Recovery
Bootloader
Cancel
OK
0
OK
3 AT
Emergency call

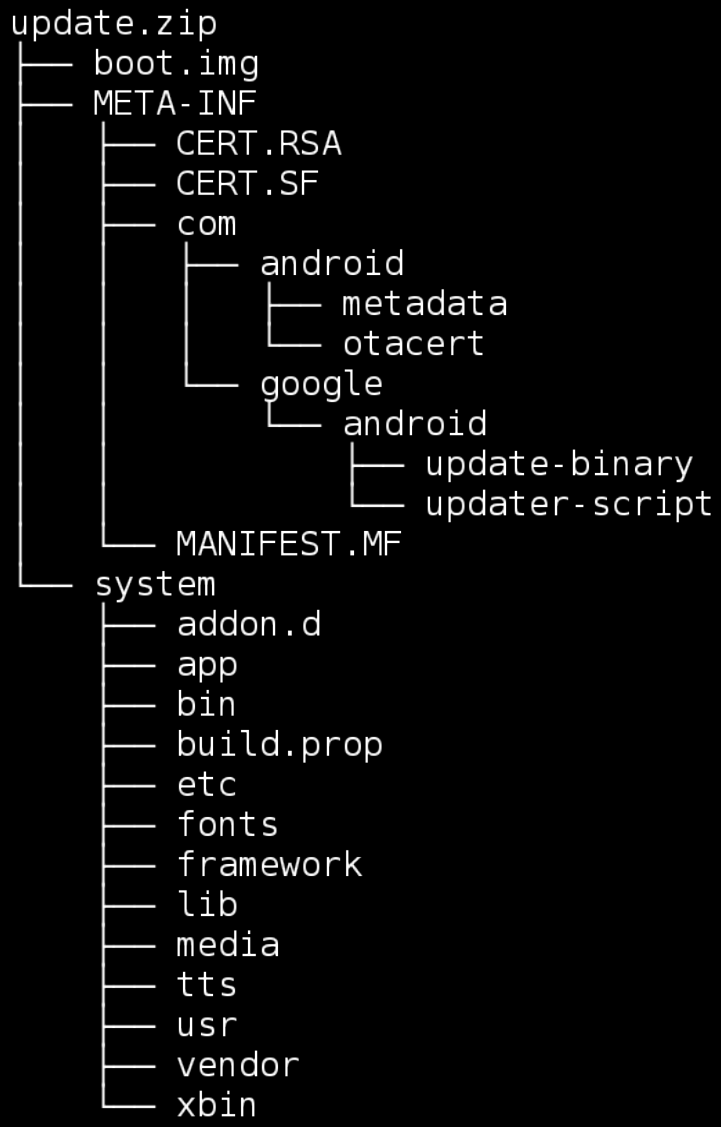
no reboot menu

- drain power
- load again
- boot into recovery via shortcuts
  - e.g. volume down + power button (HTC Desire S)

installing rootkits via recovery

- recoveries allow flashing update.zip
- usually used to flash new ROMs
- most have usb mass storage mode for sdcard enabled

typical update.zip structure



the attack

1. write rootkit running as system service
2. reboot phone to recovery
3. install rootkit via update.zip
4. reboot phone to normal OS
5. exfiltrate all data over network

the attack

1. write rootkit running as system service
2. reboot phone to recovery
3. install rootkit via update.zip
4. reboot phone to normal OS
5. exfiltrate all data over network

prerequisites

- rooted phone
- (custom ROM)
- **no** custom recovery
- **no** adb access
- unlocked bootloader

modify boot/recovery partition

- boot image contains kernel and init scripts
  - kernel-based rootkit (complicated)
  - malicious init scripts (easier)
- use fastboot to flash boot.img or directly boot into kernel

```
fastboot flash boot boot.img
```

- or: flash custom recovery and use previous vector via `update.zip`

Agenda

evil maids

detour: the android boot process

attack scenarios

- unrooted phones
- adb access
- rooted phones

protecting yourself

how to protect yourself?

- don't root your phone/flash custom roms

how to protect yourself?

- don't root your phone/flash custom roms
  - just kidding ;)

how to protect yourself?

- don't root your phone/flash custom roms
  - just kidding ;)
- just don't loose your phone...
- use encryption if possible
- lock bootloader again, if possible
- use stock recovery without options to flash zip
- unfortunately no really good solution
- `AdbdSecure` app
  - screen locked: adb off
  - screen unlocked: adb on

well. . .

. . . you are still **totally** screwed!

well. . .

. . . you are still **totally** screwed!

thx for the attention!

thx for the attention!

scared? ;)

thx for the attention!

scared? ;)

questions?

references

- “Physical Drive-By Downloads” by `@thekos`
- “Android Modding for the Security Practitioner” by Dan Rosenberg
- “Smudge Attack on Smartphone Touch Screens” by Aviv et. al.

- Phone2Phone adb

  `https://github.com/kosborn/p2p-adb/`
- `http://tjworld.net/wiki/Android/HTC/Vision/BootProcess`
- `http://wiki.opticaldelusion.org/wiki/Motoactv`
- Root with `adb restore` by Bin4ry (works on 4.X)

  `http://forum.xda-developers.com/showthread.php?t=1886460`
- `http://www.uni-ulm.de/en/in/mi/staff/koenings/catching-authtokens.html`

credits also go to: @theKos, @djrbliss, #droidsec, the modding community and everyone else I ripped of ;)