Bypassing Vendor Restrictions in HTC Devices

Pau Oliva Fora
pof@eslack.org

LaCon - Malaga

20 de septiembre de 2008

Parte I

Introducción

Introducción

Quien es HTC?

- Compañia Taiwanesa, fundada en 1997
- Fabricante de dispositivos móviles (PPC i Smartphone) con Windows Mobile
- Empezó haciendo outsorcing como ODM (Original Design Manufacturer)
- Desde 2007 HTC tiene dispositivos con su própia marca
- El que pone los nombres a los dispositivos fuma algo que no sabemos...

Introducción

Dispositivos con 1 procesador

- Procesadores TI Omap 850: Wizard, Prophet, Artemis, Elf...

Dispositivos con 2 procesadores

- Intel PXA + Qualcomm MSM6250 (Radio GSM): Wallaby, Himalaya, BlueAngel, Universal...
- Samsung SC2442 + Qualcomm MSM6275 (Radio GSM): Hermes, Trinity, Cheetah, Athena...
- Qualcomm MSM7200 (2 cores mARM+aARM): Kaiser, Polaris, Niki, Sedna...
- Qualcomm MSM7201A (2 cores): Diamond, Raphael, Victor, Venus, Blackstone...

Parte II

Formato de las ROMs

Partes de una ROM

Partes de una ROM

- IPL - Initial Program Loader
- SPL - Secondary Program Loader
- MainSplash - First Boot Splash Screen
- SubSplash - Second Boot Splash Screen
- GSM - Radio Stack (incluye OEMSBL o radio bootloader + otras partes)
- OS - WinCE IMGFS
- ExtROM - Extended ROM (storage pequeño para customizaciones del operador)
- GPS - GPS firmware

Formato NBF

Formato NBF

- Existen 5 versiones distintas
- Encriptado usando DES (xor)
- protegido con un checksum
- Cabecera de 64 bytes:
  - Dispositivo
  - Operador
  - Idioma de la rom...

Tools

- itsme perl scripts: AlpineNBFdecode.pl, TyphoonNBFdecode.pl, decodenbf.pl
- HTC64 Extended ROM tool (para universal, wizard, prophet...)

Formato NBH

Cabecera NBH:

- `magic[]={'R','0','0','0','F','F','\n'};`

```
struct HTCIMAGEHEADER {
char device[32];
unsigned long sectiontypes[32];
unsigned long sectionoffsets[32];
unsigned long sectionlengths[32];
char CID[32];
char version[16];
char language[16];
};
```

Formato NBH

NBH section types:

- 0x100 IPL, 0x101 G4IPL
- 0x200 SPL
- 0x300 Radio, 0x301 Signed Radio
- 0x400 OS, 0x401 (OS +64M)
- 0x600 MainSplash, 0x601 SubSplash

NBH bloques de datos:

- Cada bloque tiene una cabecera de bloque: 4bytes (block size), 4bytes(signature size), 1byte (flag)
- Después están los datos + la firma del bloque
- El último bloque tiene flag=2 + una firma para todo el fichero

Formato NBH

NBH toolz:

- yang - yet another nbh generator
- htcrt - htc rom tool
- nbhgen
- nbhextract

En estas tools los certificados son self-signed o se mete mierda donde deberían ir las firmas

Parte III

Mecanismos de protección

Protección en dispositivos viejos

El flasher (RUU) se autentica al bootloader mediante password dinámico:

- Podemos ver el password en la memória del flasher una vez lo ha calculado
- Podemos desensamblar el algoritmo y hacer un programa que lo calcule (testSPL.exe)
- Podemos parchear el bootloader para que acepte cualquier password

Protección en dispositivos viejos

La RUU consulta el CID del dispositivo y sólo te deja flashear ROMs con ese CID:

- Podemos parchear la RUU (MaUpgradeUt_noID.exe)
- Podemos modificar el NBF para que el CID coincida (itsme nbf tools)
- También podemos parchear el bootloader, pero no vale la pena.

Protección en dispositivos viejos

Radio stack encriptada (XOR)

- itsme - decoderadio.pl

SIM-Lock (Dispositivos PXA desde Wallaby hasta BlueAngel):

- Podemos parchear la radio para que acepte cualquier código
- Podemos hacer un *at%simlock* (específico de HTC) y leer el código de memoria con HaRET
- All-in-one SIMLock Tool (xda-developers)

Protección en dispositivos nuevos

HTC se dio cuenta de que esto era una casa de putas...

- Password estático (BsaD5SeoA) se puede ver haciendo un strings a la RUU o a el SPL
- La RUU no checkea el CID, ahora el SPL verifica CID y ModelID
- Upgrades firmados con NBH, el bootloader verifica las firmas
- SimLock: en OMAP va cifrado en la ”security area” (contiene el CID, IMEI, MSL, etc..) en el DoC
- SimLock: en Qualcomm va cifrado en la ”security area” en una pequeña NAND que contiene la radio
- A partir de MSM7201A se introduce la MPU protection: No permite que OS acceda a memoria ”protegida”

Parte IV

0wning devices

¿Para que sirve flashear código homebrew?

ROM Cooking

- Cambiar el idioma de la ROM
- Debranding (quitar customizaciones del operador)
- Más espacio de storage libre
- Más RAM libre
- Quitamos aquellos ”paquetes” que no usamos, introducimos los programas que nos gustan

Unlocking

- SIM unlocking
- CID unlocking (para poder flashear cualquier ROM)
- Security Unlocking (para poder acceder al OEMSBL, etc.)

Historia MSM6250 / HTC Universal

- Un dispositivo ”SuperCID” (CID = 00000000) se puede liberar con cualquier MSL
- 3 xda-developers: Mamaich, buzz_ligthyear, arc
- Tardaron 8 meses en parchear una radio para que no checkeara la Security Area y devolviera siempre SuperCID
- No había problemas para flashear código sin firmar

Historia MSM6275 / HTC Hermes, Trinity, Cheetah, Athena

- Primer PPC que usa NBH (código firmado): No se puede hacer ROM cooking
- SPL 1.04: permite acceder al OEMSBL ('rtask a'), no verifica bien el CID
- SPL posteriores: No se dejan downgradear. No tienen comando 'rtask'.
- Des publica 'SSPL' (soft SPL) - permite saltar a un bootloader parcheado desde CE
- El OEMSBL (radio bootloader) no verifica firmas - radio parcheada para devolver SuperCID
- HardSPL: bootloader parcheado flasheado en dispositivo.

Historia Omap850 / HTC Artemis & HTC Elf (touch v1)

- USPL: Unsigned SPL - mezcla de SSPL + HardSPL (se deja sobreescribir)
- USPL: Usa psetmem + HaRET para saltar a bootloader
- SimLock en Elf: M-SYSTEMS S3 - BK1C, se puede leer desde OS
- SimLock en Artemis: Security Area en DoC - no hay unlocker ”gratis”, se puede liberar cambiando el IMEI (¡ilegal!) y usando un unlocker comercial

Historia MSM7200 / HTC Kaiser, Polaris, Niki

- Publicamos JumpSPL - CID unlocker ”generico”, solo hay que darle un bootloader parcheado
- JumpSPL se usa para SSPL, HardSPL y Security Unlocker
- La radio stack lleva su propia firma RSA; checkeada por el OEMSBL
- HTC la caga, y el OEMSBL se puede modificar (no se verifica la firma)
- JockyW publica radio parcheada - Se puede leer el código del SimLock de memória desde OEMSBL
- Security Unlocker - Se puede quitar el simlock con cualquier MSL

Historia MSM7201A / HTC Diamond, Raphael

- MPU Protection: memória SPL de sólo lectura. Hay que rebasear la SPL entera!!!
- cmonex & Olipro - publican SPL rebaseada (con su propio certificado!!)
- Radio firmada con RSA - OEMSBL no firmado - Introducen un ”security check”
- SIM/CID/Security unlockers disponibles

Parte V

Unbricking devices

Bricked devices

Es muy fácil brickear una PDA, no es tan facil resucitarla!

Bricks ”irreparables”

- IPL / SPL: Jtag o ”frankenkaiser” en MSM7200
- Bad-Blocks en la NAND: ’task 2a’ en MSM6275
- OEMSBL: reemplazar el MSM (hardware)

Bricks ”fáciles” de reparar

CID Corrupto

- Radio corrupta, OEMSBL intacto - Se corrompe el CID
- Si el OS arranca se pueden recuperar con SSPL (CID parcheado)
- Si el OS no arranca, se pueden recuperar si el dispositivo tiene HardSPL
- Si el OS no arranca: KITL + Platform Builder (reemplazar DLLs del RIL)

Pero siempre hay alguien que mete la pata hasta el fondo: No KITL, No HardSPL, OS no arranca!

CID corrupto - Stock SPL

CID Corrupto - Stock SPL

- Sólo podemos hablar con el SPL original, el OS no arranca
- No podemos flashear nada porque el CID no coincide
- No podemos modificar el NBH (firmado y SPL original)
- SOLUCIÓN: Explotar un fallo en el bootloader

Explotar un fallo en el bootloader

Abusing the checksum command

```
checksum [start addr] [length]
- Returns CRC checksum of memory
- Shows CRC checksum value on terminal
```

Checksums de 1 byte

- Podemos pedir checksums de cada byte individual de la memória
- Haciendo checksums de 1 byte podemos dumpear toda la RAM desde bootloader
- Quien coño necesita Jtag? :D

Explotar un fallo en el bootloader

Abusing the checksum command

```
checksum [start addr] [length]
- Returns CRC checksum of memory
- Shows CRC checksum value on terminal
```

Checksums de direcciones NAND

- Cuando pedimos checksum de una direccion de la flash el bootloader copia el contenido en un offset de la RAM
- Luego devuelve un error pq no permite hacer checksums de direcciones de la FLASH
- Usando el 1-byte checksum anterior, podemos dumpearla de RAM
- Es más lento, pero podemos dumpear toda la NAND :D

SPL Stack Overflow

HTC Trinity Memory Layout

```
0x80b00000 | xxxxxxxxxxx | \
  ....     | xxxxxxxxxxx |  > wdata buffer
0x80b10000 | xxxxxxxxxxx | /
           +-------------+
           |      .      |
           |      .      |
           +-------------+
0x8c000000 | SPL-begins  | \
  ....     | SPL SPL SPL |  ME
  ....     | SPL SPL SPL |  MO <--- how_far
  ....     | SPL SPL SPL |  RY
  ....     | SPL SPL SPL | /
0x8c040000 | SPL-ends    |
           +-------------+
           |      .      |
           |      .      |
           +-------------+
           |      .      | \
0x8c08cb90 |      .      |  s
  ....     |             |  t     /\
  ....     |             |  a     ||
  ....     |             |  c     ||
  ....     |             |  k     ||
0x8c08db90 |             | /
           |             |
```

SPL Stack Overflow

Trinity SPL stack overflow

- Haciendo llamadas recursivas a 'ruustart' podemos hacer un overflow en la stack
- Controlamos un buffer de 100 bytes donde podemos meter lo que queramos

Primer paso

- Poner un patrón de bytes conocido en la stack
- Abusar del comando 'checksum' ciclicamente para encontrar el offset del top de la stack y el tamaño de stack frame

SPL Stack Overflow

Segundo Paso

- Cargar nuestro código no firmado usando 'wdata'
- Obtenemos un 'invalid cert error' del SPL
- Pero nuestros datos estan guardados en el buffer del comando wdata: 0x80b00000
- Meteremos ahi una IPL modificada para que no cargue SPL de la NAND y nuestra SPL parcheada

SPL Stack Overflow

Tercer Paso

- Necesitamos calcular cuantas iteraciones hacen falta para llegar al final de la SPL
- En las primeras recursiones de 'ruustart' ponemos padding con 0's (sólo las necesitamos para hacer el overflow de la stack)
- Luego metemos nuestro shellcode:
  - Es un handler que ejecuta el loader que reside en la RAM
  - Copia las IPL+SPL parcheadas en la RAM
  - Deshabilita la caché de instrucciones de ARM y el direccionamiento virtual
  - Salta al offset 0 para iniciar la IPL

SPL Stack Overflow

Cuarto Paso

- Despues de meter el shellcode, enviamos las siguientes iteraciones de ruustart con padding que contiene instrucciones 'branch'
- Estos branches, son saltos relativos al handler (shellcode)
- Calculamos cuantas llamadas necesitamos hacer a ruustart (basandonos en el offset del objetivo, el offset inicial de la stack y el tamaño de stack frame)

SPL Stack Overflow

Último Paso

- Finalemnte, necesitamos una forma de saltar a nuestro código parcheado;
- llamremos a una funcion del bootloader que tenga el entry point alineado con el stack frame 'overfloweado'
- Sólo controlamos 0x64 bytes de todo el tamaño de stack frame (típicamente 0x0e)
- Esto va a variar en cada versión distinta de SPL - bruteforcing

SPL Stack Overflow

HTC Hermes Memory Layout

```
           |     .       | \
           |     .       |  s     /\
           |             |  t     ||
  ....     |             |  a     ||
           |             |  c     ||
  ....     |             |  k
0x8c033b90 |             | /
           +-------------+
           |     .       |
           |     .       |
           +-------------+
0x8c080000 | SPL-begins  | \
  ....     | SPL SPL SPL |  ME
  ....     | SPL SPL SPL |  MO <--- how_far; we never reach here :(
  ....     | SPL SPL SPL |  RY
  ....     | SPL SPL SPL | /
0x8c0c0000 | SPL-ends    |
           +-------------+
           |     .       |
```

SPL Stack Overflow

HTC Hermes Stack overflow

- La stack crece en la misma dirección que en Trinity
- Pero el código de la SPL está debajo de la stack, así que nunca conseguimos sobreescribirla
- No podemos llamar a una función que salte a nuestro handler

The end