Reversing ARM Phone Firmwares

LaCon 08

Esteve Espuña
esteve@eslack.org

20 de septiembre de 2008

Reversing ARM Phone Firmwares

- Introducción a los procesadores ARM
- Objetivo
- Tools
- Qualcomm MSM
- SPL, un ejemplo de análisis de binario plano
- Inyección de código en bootloader

Parte II

Introducción a los procesadores ARM

Inicios

- Sophie Wilson y Steve Furber de **Acorn Computers Ltd** empezaron el diseño del ARM en 1983.
- Objetivo: Un procesador RISC compacto con baja latencia I/O.
- El primer procesador *production ready* fue el ARM2 lanzado en 1985:
  - Bus de 32bits
  - PC de 26bits
  - 30.000 transistores (más simple del mercado de 32b) $\rightarrow$ bajo consumo
  - No Caché

Inicios

- Finales de 80's Apple y VLSI junto con Acron empezaron un nuevo diseño.
- 1990 spin-off del equipo de diseño de Acron para formar Advanced RISC Machines Ltd.
  - En 1991 Apple-ARM (ARM6) para la Apple Newton PDA.
- 1994, ARM 610 Risc PC (RISC OS).
- ARM empieza a licenciar *cores* desede el principio. Por ejemplo: DEC StrongARM (ARM610 a 233Mhz), después Intel StrongARM, más tarde XScale, ahora comercializado por Marvell.
- Advanced RISC Machines Ltd. se convirtió en ARM en 1998 cuando salió al London Stock Exchange y al NASDAQ.

Actualidad

- El negocio de ARM es vender IP Cores. Cada fabricante añade sus periféricos.
- En 2006 :
  - Facturó 164.1 million USD en royalties, con 2.45 Billones de unidades vendidas.
  - Facturó 119.5 million USD en ventas de cores, con 65 procesadores.
  - 60 % en royalties y 40 % en licencias.
- ARM7TDMI es el mas popular, iPod, Nintendo DS, Nokia, Lego MindStorm, Audio Processing DreamCast, iriver ...

Diseño General

- Procesador cableado (que no cabreado).
- Arquitectura Load/Store.
- Registros de 32bits.
- 4 bytes / instrucción. Densidad de instrucciones baja (solución Thumb).
- **Ejecución condicional**, no hay predictor de saltos.
- 32bit barrel shifter para todas las instrucciones aritmeticas.

```
while (i != j)
{
    if (i > j)
        i -= j;
    else
        j -= i;
}
---------------
loop:  CMP  Ri, Rj   ; set condition
                     ; "NE" if (i != j)
                     ; "GT" if (i > j),
                     ; "LT" if (i < j)
       SUBGT  Ri, Ri, Rj ; if "GT", i = i-j;
       SUBLT  Rj, Rj, Ri ; if "LT", j = j-i;
       BNE    loop       ; if "NE", then loop
```

32-bit barrel shifter:

```
a += (j << 2);

---------------

ADD Ra, Ra, Rj, LSL #2
```

Condicionales

Bits 31-28 (part 1)

- ARM_COND_EQ 0x00000000 // Z set, equal
- ARM_COND_NE 0x10000000 // Z clear, no equal
- ARM_COND_HS 0x20000000 // C set, unsigned higher or same
- ARM_COND_LO 0x30000000 // C clear, unsgined lower
- ARM_COND_MI 0x40000000 // N set , negative
- ARM_COND_PL 0x50000000 // N clear , positive or zero
- ARM_COND_VS 0x60000000 // V set , overflow
- ARM_COND_VC 0x70000000 // V clear, no overflow
- ARM_COND_HI 0x80000000 // C set and Z clear, unsigned higher

Condicionales

Bits 31-28 (part 2)

- ARM_COND_LS 0x90000000 // C clear or Z, unsigned lower or same
- ARM_COND_GE 0xa0000000 // N set and V set , or N clear V clear, $>=$
- ARM_COND_LT 0xb0000000 // N clear and V clear , or N clear V set, $<$
- ARM_COND_GT 0xc0000000 // $>$
- ARM_COND_LE 0xd0000000 // $<=$
- ARM_COND_AL 0xe0000000 // Always
- ARM_COND_NV 0xf0000000 // reserved

Saltos (branch)

- **BX** Branch and exchange, para saltar de ARM a Thumb y viceversa.
- **BL** y **B**:
  - BL guarda en el registro LR el valor de PC+4
  - El salto es a un offset de 24 bits en CA2 : $Ad = PC + 8 + (Off << 2)$;
  - Direccionamos +/- 32Mb.

Data Processing

- Todas aquellas instrucciones que operan sobre registros. Por ejemplo: AND, OR, EOR, ADD, CMP, SUB, MOV ...
- Formato de las instrucciones DP:
  - Op (Rn, Rd, Op2)
  - Op es la operación. 4 bits.
  - Rn es el primer operando. 4 bits.
  - Rd es el destino. 4 bits.
  - Op2 es un segundo operando. 12 bits.
- El primer operando (Rn) siempre es un registro.

Data Processing

- El segundo operando (Op2) 12 bits:
  - Registro más un shift: Shift(8b) , Rm(4b). A su vez shift puede ser otro registro.
  - Valor : Rotate(4b) , Imm(8b). $Valor = Imm >> (rotate * 2)$
- Modificación flags de condición. (1b)
- Sub-grupo de DP:
  - MUL/MLA, MULL/MLAL: Multiplicaciones.
  - MRS y MSR: Guardar y cargar los valores de los flags de condiciones.

Single data transfer

- **LDR, STR**: LOAD / STORE.
- Registro base (Rn 4b) , registro destino/orígen (Rd 4b) y offset (12b).
- La dirección destino/origen se calcula con el registro base y el offset, dónde offset es otro registro o un valor.
- Opciones de autoincremento y escritura byte/word.

Block data transfer

- **LDM, STM**: LOAD MULTIPLE / STORE MULTIPLE.
- Registro base (Rn 4b) y un listado de registros (16b).
- Listado de registros: 1 bit $\rightarrow$ 1 registro.
- Se carga/guarda el valor de cada registro desde la dirección base.
- Podemos especificar si se incrementa Rn y si el incremento es positivo o negativo.

Otras instrucciones

- **SWP**: Swap de valores.
- **SWI**: Software interrupt, tenemos 24bits de info. PC a 0x08.
- **CDP**: Coprocessor Data Operations
- **LDC, STC**: Load Coprocessor, Store Coprocessor
- **MRC, MCR**: Transferencia Coprocessor memoria.
- Instrucción indefinida : Cond_4b , 011b , X_20b , 1 , X_4b. Trap condicional.

Registros

- R0-R10 : genéricos
- R11 → fp, frame pointer
- R12 → ip, scratch register
- R13 → sp, stack pointer
- R14 → lr, link register
- R15 → pc, program counter

Parte III

Objetivo

Objetivo

Principal

- Analizar los distintos *bootloaders* que hay en los nuevos teléfonos de HTC.
  bootloader $\leftrightarrow$ firmware

Parte IV

Tools

Tools

- SPAM!!!:
  - radare. http://radare.nopcode.org/
- Análisis:
  - IDA Pro. http://www.hex-rays.com/idapro/
- Generación:
  - arm gcc cross-compiler

ARM function finder

IDA Plugin

- Plugin IDA para buscar funciones.
- Solo tiene sentido en binario.
- Funcionamiento:
  - Búsqueda posible prólogo.
  - Búsqueda posible epílogo.
  - Matching prólogo-epílogo.

ARM function finder

Prólogo

- **BL func**
- func: **STMFD SP!,** {**R4-R10,LR**}
- Store Multiple en SP.
- Hacemos un ”push”de los registros usados.

Epílogo

- tipo 1: **LDMFD SP!,** {**R4-R10,LR**} **; B LR**
- tipo 2: **LDMFD SP!,** {**R4-R10,PC**}
- Hacemos un ”pop”de los registros usados.
- Restaura el PC.

ARM function finder

Matching

- Los registros guardados y los cargados serán los mismos.
- Guardamos una pila de llamadas.
- Marca la primera instrucción para análisis.
- IDA hace el auto-análisis.

Parte V

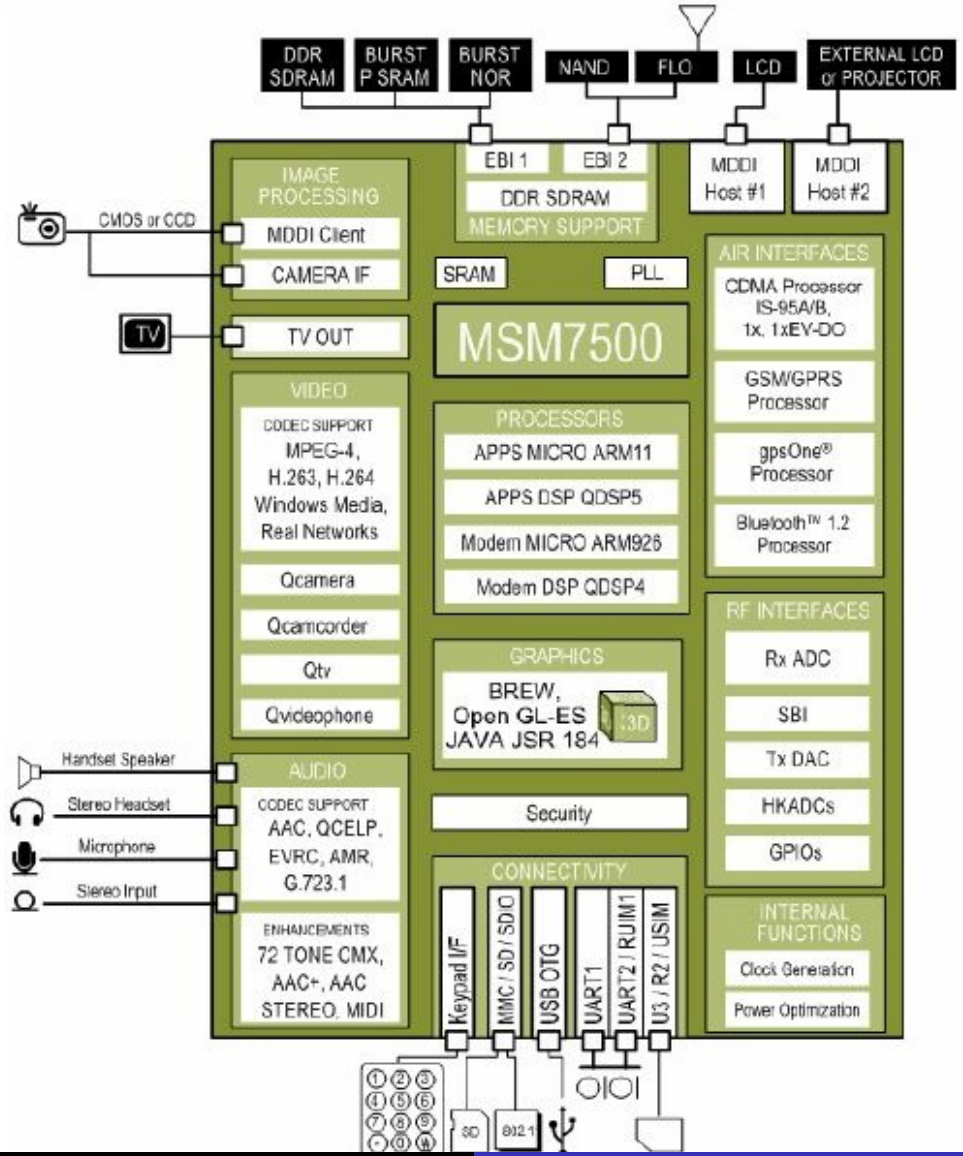
Qualcomm MSM

Qualcomm MSM: introducción

Multiprocessor Architecture

- Arquitectura con 2 procesadores.
- ARM1136EJS, procesador principal. aARM.
- ARM926, procesador enfocado al módem. mARM.
- DSP.
- DDR.

Qualcomm MSM: introducción



Qualcomm MSM: introducción

Seguridad : Introducción

- Trusted boot. Root of trust. ROM sign-checking loader.
- Separación de HW.
- Encriptación FS.
- e-FUSE.
- MPU.

Qualcomm MSM: Boot

Secure Boot

- Primary Bootloader (PBL)
- Secondary Bootloader (SBL)
  - QC Secondary Bootloader (QCSBL)
  - OEM Secondary Bootloader (OEMSBL)

Qualcomm MSM: Imágenes

Boot Images

- Tabla de particiones.
- Configuración de boot.
- QCSBL.
- OEMSBL image header.
- OEMSBL image.
- ARM9 modem image header.
- ARM9 modem image.
- ARM11 bootloader image header.
- ARM11 bootloader image.

Parte VI

SPL, ejemplo

Parte VII

Inyección de código en bootloader

Introducción

- Cada dos por tres cambian el firmware de los distintos HTC. Nuevos teléfonos, versiones, ...
- No suele haber problema de espacio para la inyección.
- Existen compiladores.
- **Inyección de código compilado**

Introducción

ELF

- Formato estándar para ficheros ejecutables, objetos, librerías . . .
- Flexible, extensible y no ligado a ningún procesador.
- Usado ampliamente en Linux.
- Generaremos objetos para inyectarlos.

Pasos

Inyección

- Parser ELF
- Seleccionar secciones a copiar
- Inyectar en el offset del binario
- Hacer el relocating

Parser ELF

ELF Header

- Magic number
- Tipo de arquitectura
- Tipo de ejecutable (objeto)
- Entry Point (no usamos)
- Punteros a los distintos headers
- Tamaños y más info que tampoco usamos

Parser ELF

Section Headers

- Información sobre las secciones
- Lista , cada header nos marca el siguiente. sh_link
- Información sobre tipo y flags.
- Dirección virtual/fichero.
- Tamaño fichero/memoria.

Secciones a copiar

- Secciones necesarias para la ejecución.
- Empezamos por **.text**, código.
- Para cada símbolo en la tabla de relocations
  - Mirar a qué sección pertenece el símbolo.
  - Si no tengo la sección el la lista la añado.
- Al final, lista de secciones que se acaban llamando.

Inyección

- Inyectamos alineadas cada sección de la lista anterior.
- No importa el orden.
- Nos guardamos la base a fichero de cada sección.
- El código está insertado.

Relocation

- Obtenemos la tabla de relocation (SHT_REL).
- Obtenemos la tabla de simbolos (SHT_SYMTAB)
- Para cada entrada en la tabla de relocations:
  - Miramos el tipo de relocation
  - Parcheamos segun el tipo :
    - R_ARM_PLT32 o R_ARM_PC24
    - R_ARM_ABS32
- Solo queda guardar el fichero a disco (MENTIRA!)

Relocation

R_ARM_PLT32 o R_ARM_PC24

- Salto relativo a offset de 24b. ej: B 0x1231
- Parche:
  - Base de la sección del símbolo destino.
  - Offset del símbolo en la sección.
  - La tabla de relocation nos indica dónde tenemos que parchear.
  - Parcheamos los últimos 24bits, el resto es 4b para condicional, B/BL 4b.

Relocation

R_ARM_ABS32

- Posición de memória con un valor absoluto. 32 bits. ej: LD PC,=0x8c001A80
- Parche:
  - Base de la sección dónde se encuentra el símbolo destino, base
  - Valor del símbolo en la sección de relocation, rv .
  - El valor de la dirección dónde hacemos el parche, pv.
  - El parche es: $base + pv + rv$

Limitaciones

- No estan implementados todos los tipos de relocation.
- Un solo .o
- Difícil hacer imports.
- Punto dónde inyectar.
- Parchear manualmente la llamada a nuestras funciones

Ejemplo

```
int patch( int a) { return a+2; }

void entry ( )
{
  int a;
  for ( a = 0 ; a < 100 ; a ++ ) a+= patch (a);
}
```

```
0000001c <patch>:
1c:   e1a0c00d   mov     ip, sp
20:   e92dd800   push    {fp, ip, lr, pc}
...
3c:   e89da808   ldm     sp, {r3, fp, sp, pc}

00000040 <entry>:
40:   e1a0c00d   mov     ip, sp
44:   e92dd800   push    {fp, ip, lr, pc}
...
60:   ebfffffe   bl      1c <patch>
...
8c:   e89da808   ldm     sp, {r3, fp, sp, pc}
```

Relocations

- Queremos cargarlo en el offset 100 del fichero
- Base 8c001000
- En memoria: 8c001100
- Relocation Table (address, sección)
  - 00000060, section 1
- parche en 60 saltar a 1c
- Se traduce en parche 0x8c001160 saltar a 0x8c00111c