Università Degli Studi Di Roma Tor Vergata

FACOLTÀ DI INGEGNERIA
Corso di Laurea Magistrale in Ingegneria Informatica

Patch-3.9.2.1-xserve

Antonio Papa

Anno Accademico 2012-2013

Indice

Capitolo 1

Patch Xserve

In questa breve trattazione verranno descritte le modifiche da effettuare per installare un kernel 3.9.2 vanilla sulle macchine Apple Xserve. Daremo per assunto che le macchine in questione abbiano la configurazione software illustrata nel tecnical report [1].
Passiamo quindi ad illustrare la patch.

1.1 Il problema delle system tables

Come illustrato nel [1] uno dei problemi da risolvere per avviare un kernel Linux a 64 bit è trovare le locazioni in Ram di alcune system table: le DMI e ACPI.
Il firmware del bios di solito colloca queste tabelle in predefinite regioni della low memory. Nel peggiore dei casi è lo stesso kernel che all'avvio cerca suddette aree di memoria.
I computer con un EFI firmware purtroppo si comportano in maniera differente, infatti gli indirizzi delle sistem table sono contenuti in una EFI system table, il cui indirizzo è passato al kernel come boot parameter. Quindi le varie system table possono essere collocate in qualsiasi locazione di memoria.
Nel caso dell'Xserve il firmware EFI colloca le system table in corrispondenza di high physical address quindi il kernel non riesce a trovarle durante la procedura di boot.
Per risolvere questo problema è necessario che all'avvio il kernel ispezioni l'EFI system table, al fine di determinare gli indirizzi delle altre system table.

1.2 Soluzione

Il problema descritto nel paragrafo precedente è già stata risolto nel [1] per i kernel 2.6.22-2.6.24, scopo del documento è estendere tali modifiche al fine di supportare kernel più recenti quali il 3.9.2.
Nel seguito verranno mostrati i cambiamenti da apportare al kernel 3.9.2.

1.2.1 Legacy Protected mode

Modificando elilo come descritto in [1] si riesce a caricare un kernel linux a 64-bit e a saltare al suo entry point. Purtroppo c'è un problema da risolvere: la CPU è ancora in

legacy protected mode, mentre il default entry point del kernel assume che la CPU è a 64-bit long mode. Per risolvere il problema si usa un altro entry point *startup_32()*, una funzione assembly locata in *arch/x86/boot/compressed/vmlinux.lds.S*. Questa funzione è compilata a 32 bit e forza l'esecuzione della CPU a 64 bit prima di invocare la funzione *startup_64()*:

```
--- linux-3.9.2/arch/x86/boot/compressed/vmlinux.lds.S  2013-05-11
    16:19:28.000000000 +0200
+++ linux-3.9.2.1/arch/x86/boot/compressed/vmlinux.lds.S  2013-09-09
    13:15:58.542658076 +0200
@@ -9,7 +9,7 @@

 #ifdef CONFIG_X86_64
 OUTPUT_ARCH(i386:x86-64)
-ENTRY(startup_64)
+ENTRY(startup_32)
 #else
 OUTPUT_ARCH(i386)
 ENTRY(startup_32)
```

1.2.2 Efi Xserve

Adesso è necessario leggere l'EFI system table per determinare gli indirizzi delle varie system configuration table (DMI, ACPI). Si inizia con il definire delle strutture per l'EFI configuration table e per l'EFI system table in *include/linux/efi.h*:

```
#ifdef CONFIG_EFI_XSERVE
typedef struct {
  efi_guid_t guid;
  u32 table;
} efi_config_table_t;
#else
typedef struct {
  efi_guid_t guid;
  u64 table;
} efi_config_table_64_t;

typedef struct {
  efi_guid_t guid;
  u32 table;
} efi_config_table_32_t;

typedef struct {
  efi_guid_t guid;
  unsigned long table;
} efi_config_table_t;
#endif
```

```
#ifdef CONFIG_EFI_XSERVE
typedef struct {
  efi_table_hdr_t hdr;
  u32 fw_vendor;  /* physical addr of CHAR16 vendor string */
```

```
    u32 fw_revision;
    u32 con_in_handle;
    u32 con_in;
    u32 con_out_handle;
    u32 con_out;
    u32 stderr_handle;
    u32 stderr;
    u32 runtime;
    u32 boottime;
    u32 nr_tables;
    u32 tables;
} efi_system_table_t;
#else
typedef struct {
/* Original definitions follow ... */
} efi_system_table_64_t;

typedef struct {
/* Original definitions follow ... */
} efi_system_table_32_t;

typedef struct {
/* Original definitions follow ... */
} efi_system_table_t;
#endif
```

Dalla [1] si apprende che la ridefinizione delle strutture *efi_ config_ table_ t* ed *efi_ system_ table_ t* è dovuta alla mancanza di una variante delle suddette tabelle a 32 bit. Infatti nei kernel 2.6.2x i campi sono definiti come *unsigned long*, che in un sistema x86_64 sono a 64 bit. I kernel più recenti quali il 3.9.2 non soffrono di questo problema infatti sia per l'EFI configuration table che per l'EFI system table sono state previste tre versioni:

- a 64 bit
- a 32 bit
- con campi *unsigned long*

Per motivi di tempo si è scelto di non usare la versione a 32 bit, ma di preservare il più possibile la patch originale, quindi si è deciso di ridefinire le tre varianti in un'unica struttura a 32 bit. Naturalmente questa pecca stilistica andrebbe corretta, per rendere il lavoro più presentabile in ambito commerciale o accademico.
Adesso è necessario implementare un meccanismo che ricerca gli indirizzi delle config table nell'EFI system table. Quindi è necessario creare un nuovo file *efi_ xserve.c* in *arch/x86/kernel*:

```
#define PFX            "EFI: "

struct efi efi;
EXPORT_SYMBOL(efi);

void __init efi_init(void)
```

```
{
        efi_config_table_t *config_tables;
        efi_char16_t *c16;
        char vendor[100] = "unknown";
        unsigned long num_config_tables;
        int i = 0;

        memset(&efi, 0, sizeof(efi) );

        efi.systab = __boot_va(boot_params.efi_info.efi_systab);
        /*
         * Verify the EFI Table
         */
        if (efi.systab == NULL)
                printk(KERN_ERR PFX "Woah! Couldn't map the EFI system table
                    .\n");
        if (efi.systab->hdr.signature != EFI_SYSTEM_TABLE_SIGNATURE)
                printk(KERN_ERR PFX "Woah! EFI system table signature
                    incorrect\n");
        if ((efi.systab->hdr.revision >> 16) == 0)
                printk(KERN_ERR PFX "Warning: EFI system table version "
                       "%d.%02d, expected 1.00 or greater\n",
                       efi.systab->hdr.revision >> 16,
                       efi.systab->hdr.revision & 0xffff);

        /*
         * Grab some details from the system table
         */
        num_config_tables = efi.systab->nr_tables;
        config_tables = (efi_config_table_t *)__boot_va(efi.systab->tables);

        /*
         * Show what we know for posterity
         */
        c16 = (efi_char16_t *) __boot_va(efi.systab->fw_vendor);
        if (c16) {
                for (i = 0; i < (sizeof(vendor) - 1) && *c16; ++i)
                        vendor[i] = *c16++;
                vendor[i] = '\0';
        } else
                printk(KERN_ERR PFX "Could not map the firmware vendor!\n");

        printk(KERN_INFO PFX "EFI v%u.%.02u by %s \n",
               efi.systab->hdr.revision >> 16,
               efi.systab->hdr.revision & 0xffff, vendor);

        /*
         * Let's see what config tables the firmware passed to us.
         */
        if (config_tables == NULL)
                printk(KERN_ERR PFX "Could not map EFI Configuration Table!\n
                    ");
```

```
        efi.mps        = EFI_INVALID_TABLE_ADDR;
        efi.acpi       = EFI_INVALID_TABLE_ADDR;
        efi.acpi20     = EFI_INVALID_TABLE_ADDR;
        efi.smbios     = EFI_INVALID_TABLE_ADDR;
        efi.sal_systab = EFI_INVALID_TABLE_ADDR;
        efi.boot_info  = EFI_INVALID_TABLE_ADDR;
        efi.hcdp       = EFI_INVALID_TABLE_ADDR;
        efi.uga        = EFI_INVALID_TABLE_ADDR;

        printk(KERN_INFO PFX "Configuration tables:");
        for (i = 0; i < num_config_tables; i++) {
                if (efi_guidcmp(config_tables[i].guid, MPS_TABLE_GUID) == 0)
                    {
                        efi.mps = config_tables[i].table;
                        printk(" MPS=0x%x ", config_tables[i].table);
                } else
                    if (efi_guidcmp(config_tables[i].guid, ACPI_20_TABLE_GUID
                        ) == 0) {
                        efi.acpi20 = config_tables[i].table;
                        printk(" ACPI 2.0=0x%x ", config_tables[i].table);
                } else
                    if (efi_guidcmp(config_tables[i].guid, ACPI_TABLE_GUID)
                        == 0) {
                        efi.acpi = config_tables[i].table;
                        printk(" ACPI=0x%x ", config_tables[i].table);
                } else
                    if (efi_guidcmp(config_tables[i].guid, SMBIOS_TABLE_GUID)
                         == 0) {
                        efi.smbios = config_tables[i].table;
                        printk(" SMBIOS=0x%x ", config_tables[i].table);
                } else
                    if (efi_guidcmp(config_tables[i].guid, HCDP_TABLE_GUID)
                        == 0) {
                        efi.hcdp = config_tables[i].table;
                        printk(" HCDP=0x%x ", config_tables[i].table);
                } else
                    if (efi_guidcmp(config_tables[i].guid,
                        UGA_IO_PROTOCOL_GUID) == 0) {
                        efi.uga = config_tables[i].table;
                        printk(" UGA=0x%x ", config_tables[i].table);
                }
        }
        printk("\n");
}
```

Come è possibile notare il meccanismo proposto è quello del [1], per ulteriori approfondimenti circa la sua implementazione si consiglia la lettura di suddetto report.
Al fine di abilitare le funzionalità fornite dalla pach si aggiunge una voce nel file *arch/x86/Kconfig*:

```
config EFI_XSERVE
       bool "Boot from EFI on Apple Xserve"
       depends on ACPI
       default n
```

```
        ---help---
        This enables the kernel to boot on EFI platforms using
        system configuration information passed to it from the firmware.
        EFI runtime services are not supported. Moreover, the kernel still
        relies on a correct BIOS e820 map being passed by the boot loader.

        This option is currently only useful for booting Apple Xserve
        (late 2006) systems.

        If you are not sure, just say N.
```

Per compilare il file *efi_xserve.c* è necessario modificare il file *arch/x86/kernel/Makefile*:

```
--- linux-3.9.2/arch/x86/kernel/Makefile  2013-05-11 16:19:28.000000000
    +0200
+++ linux-3.9.2.1/arch/x86/kernel/Makefile  2013-09-09 15:08:24.026472873
    +0200
@@ -45,6 +45,7 @@
 obj-$(CONFIG_STACKTRACE) += stacktrace.o
 obj-y        += cpu/
 obj-y        += acpi/
+obj-$(CONFIG_EFI_XSERVE) += efi_xserve.o
 obj-y        += reboot.o
 obj-$(CONFIG_X86_MSR)    += msr.o
 obj-$(CONFIG_X86_CPUID)    += cpuid.o
```

Infine invochiamo la funzione *efi_init()* in *arch/x86/kernel/efi_xserve.c* prima della scansione delle tabelle DMI e ACPI:

```
--- linux-3.9.2/arch/x86/kernel/setup.c 2013-05-11 16:19:28.000000000 +0200
+++ linux-3.9.2.1/arch/x86/kernel/setup.c 2013-09-28 14:15:37.825595000
    +0200
@@ -998,6 +998,10 @@
  if (efi_enabled(EFI_BOOT))
    efi_init();

+#ifdef CONFIG_EFI_XSERVE
+ efi_init();
+#endif
+
  dmi_scan_machine();
```

1.2.3 e820

Come descritto in [1] ci sono solo 5 tipi di memoria e820:

1. E820_RAM
2. E820_RESERVED
3. E820_ACPI
4. E820_NVS

5. E820_EXE_CODE

Ma nel kernel 3.9.2 vengono definite solo le prime 4, mentre la quinta è definita come *E820_UNUSABLE*. Per ovviare a questo problema modifichiamo il file *arch/x86/include/uapi/asm/e820.h* nel seguente modo:

```
--- linux-3.9.2/arch/x86/include/uapi/asm/e820.h  2013-05-11
    16:19:28.000000000 +0200
+++ linux-3.9.2.1/arch/x86/include/uapi/asm/e820.h  2013-09-09
    16:48:54.262307000 +0200
@@ -36,9 +36,13 @@
 #define E820_RESERVED  2
 #define E820_ACPI  3
 #define E820_NVS 4
-#define E820_UNUSABLE  5
-

+#ifdef CONFIG_EFI_XSERVE
+ #define E820_EFI_EXEC_CODE 5
+ #define E820_UNUSABLE 6
+#else
+ #define E820_UNUSABLE 5
+#endif
 /*
  * reserved RAM used by kernel itself
  * if CONFIG_INTEL_TXT is enabled, memory of this type will be
```

E' bene segnalare un'altra pecca stilistica, infatti come è possibile notare la definizione *E820_UNUSABLE* non viene cancellata, ma le viene semplicemente assegnato un tipo di regione inesistente (ricordiamo che ci sono solo 5 tipi di regione). Questo è stato volutamente fatto per motivi di tempo, al fine di non dover controllare le parti di codice che avrebbero potuto usare *E820_UNUSABLE*.
Adesso si devono modificare le funzioni *e820_print_type()* e *e820_type_to_string()* nel file *arch/x86/kernel/e820.c* per gestire la casistica relativa al tipo di regione di memoria *E820_EXE_CODE*:

```
--- linux-3.9.2/arch/x86/kernel/e820.c  2013-05-11 16:19:28.000000000 +0200
+++ linux-3.9.2.1/arch/x86/kernel/e820.c  2013-09-09 17:24:50.742248000
    +0200
@@ -149,6 +149,11 @@
  case E820_UNUSABLE:
    printk(KERN_CONT "unusable");
    break;
+#ifdef CONFIG_EFI_XSERVE
+   case E820_EFI_EXEC_CODE:
+   printk(KERN_CONT "EFI runtime services code");
+   break;
+#endif
  default:
    printk(KERN_CONT "type %u", type);
    break;
@@ -915,6 +920,9 @@
  case E820_ACPI: return "ACPI Tables";
```

```
  case E820_NVS:  return "ACPI Non-volatile Storage";
  case E820_UNUSABLE: return "Unusable memory";
+#ifdef CONFIG_EFI_XSERVE
+ case E820_EFI_EXEC_CODE: return "EFI Runtime services code";
+#endif
  default:  return "reserved";
  }
 }
```

1.2.4 ACPI e DMI

Infine bisogna modificare le funzioni di scan dell'ACPI e del DMI in modo da evitare una ricerca iterativa delle loro config table. Infatti tramite l'EFI config table disponiamo già degli indirizzi fisici delle due tabelle.
Quindi si modifica la funzione di scan del DMI: *dmi_scan_machine()* locata in *drivers/firmware/dmi_scan.c*:

```
--- linux-3.9.2/drivers/firmware/dmi_scan.c 2013-05-11 16:19:28.000000000
    +0200
+++ linux-3.9.2.1/drivers/firmware/dmi_scan.c 2013-09-09 19:07:35.856844000
    +0200
@@ -470,6 +470,21 @@
  char __iomem *p, *q;
  int rc;

+ #if CONFIG_EFI_XSERVE
+   if (efi.smbios != EFI_INVALID_TABLE_ADDR) {
+   p = dmi_ioremap(efi.smbios, 32);
+   if (p == NULL)
+     goto error;
+
+   rc = dmi_present(p + 0x10); /* offset of _DMI_ string */
+   dmi_iounmap(p, 32);
+   if (!rc) {
+     dmi_available = 1;
+     goto out;
+   }
+   }
+ #endif
+
  if (efi_enabled(EFI_CONFIG_TABLES)) {
    if (efi.smbios == EFI_INVALID_TABLE_ADDR)
      goto error;
```

e la funzione di scan dell'ACPI: la *acpi_os_get_root_pointer()* locata in *drivers/acpi/osl.c*:

```
--- linux-3.9.2/drivers/acpi/osl.c  2013-05-11 16:19:28.000000000 +0200
+++ linux-3.9.2.1/drivers/acpi/osl.c  2013-09-28 16:11:09.661857457 +0200
@@ -248,7 +248,12 @@
  if (acpi_rsdp)
    return acpi_rsdp;
```

```
 #endif
-
+#ifdef CONFIG_EFI_XSERVE
+ if (efi.acpi20 != EFI_INVALID_TABLE_ADDR)
+   return efi.acpi20;
+ else if (efi.acpi != EFI_INVALID_TABLE_ADDR)
+   return efi.acpi;
+#endif
  if (efi_enabled(EFI_CONFIG_TABLES)) {
    if (efi.acpi20 != EFI_INVALID_TABLE_ADDR)
      return efi.acpi20;
@@ -262,7 +267,20 @@
  } else {
    acpi_physical_address pa = 0;

+#ifdef CONFIG_EFI_XSERVE
+ if (efi.acpi20 != EFI_INVALID_TABLE_ADDR)
+ {
+   pa = efi.acpi20;
+   return_ACPI_STATUS(AE_OK);
+ }
+ else if (efi.acpi != EFI_INVALID_TABLE_ADDR)
+ {
+   pa =  efi.acpi;
+   return_ACPI_STATUS(AE_OK);
+ }
+#else
    acpi_find_root_pointer(&pa);
+#endif
    return pa;
  }
 }
```

Capitolo 2

Configurazione

In questo capitolo vengono descritti gli step necessari per installare un kernel vanilla 3.9.2 su una macchina Apple Xserve. E' bene ricordare che la configurazione software di partenza deve essere quella illustrata nel [1] e che bisogna disporre dei seguenti file:

1. patch-3.9.2.1-xserve (a)
2. config-3.9.2.1-xserve (b)
3. initrd.img (c)

Detto ciò si passa ad illustrare i vari step.

- Per prima cosa è necessario procurarsi il sorgente di un kernel vanilla 3.9.2, scaricabile al seguente indirizzo: *https://www.kernel.org/pub/linux/kernel/v3.x/*
- Applichiamo la patch *patch-3.9.2.1-xserve* (a) al kernel:

```
$ cd /usr/src/linux-3.9.2
$ patch -p1 < ../patch-3.9.2.1-xserve
$ mv linux-3.9.2 linux-3.9.2.1-xserve
```

- Utilizziamo il file *config-3.9.2.1-xserve* (b) che contiene la configurazione per l'Xserve

```
$ cd /usr/src/linux-3.9.2.1-xserve
$ cp ../config-3.9.2.1-xserve ./.config
```

- Compiliamo il kernel ed installiamo i moduli

```
$ cd /usr/src/linux-3.9.2.1-xserve
$ make -j4
$ cat arch/x86/boot/bzImage > vmlinuz-3.9.2.1
# make modules_install
```

- Montiamo la partizione in cui è locato elilo

```
# mount -tvfat /dev/sda1 /mnt/hd
```

- Spostiamo l'immagine del kernel e l'*initrd.img* (c) nella partizione di elilo:

```
# cp initrd.img /mnt/hd/efi/linux
# mv /usr/src/linux-3.9.2.1-xserve/vmlinuz-3.9.2.1 /mnt/hd/efi/
    linux
```

- Modifichiamo il file *elilo.conf* aggiungendo la seguente entry:

```
image=vmlinuz-3.9.2.1
label=vmlinuz-3.9.2.1
append="earlyprintk=ttyS0,115200 ignore_loglevel console=ttyS0,115200n8
     load_ramdisk=1 prompt_ramdisk=0 ramdisk_size=12800 rw init=/
    linuxrc ROOTDEV=/dev/sda3"
root=/dev/ram
initrd=initrd.img
```

Una volta effettuate le seguenti modifiche, non resta che riavviare la macchina e allo start selezionare il kernel 3.9.2.1.

Bibliografia

[1] M. Cesati and A.Morari. Installing a 64-bit Linux distribution on the Intel-based Apple Xserv.