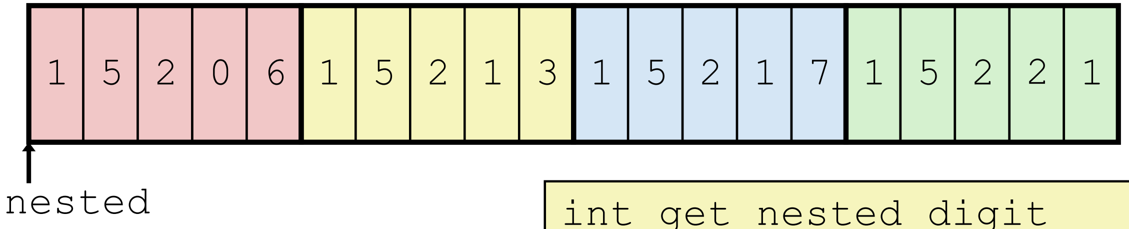
SWE2001: System Program
Lecture 0x0F: Final

Systems Security Lab @ SKKU

Nested Array Element Access Code

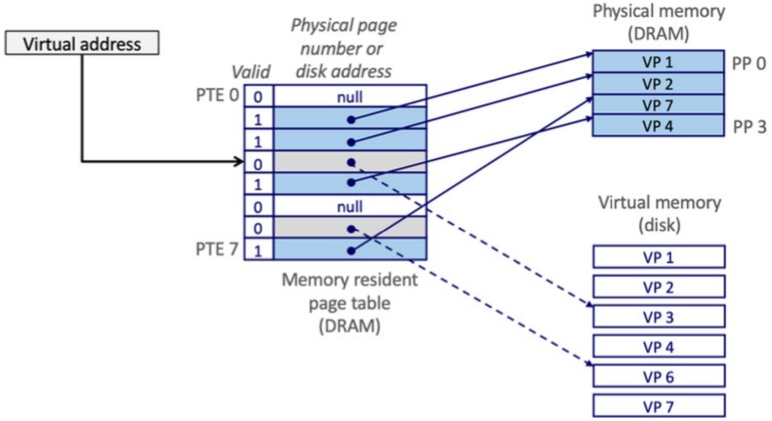


```
int get_nested_digit
  (int index, int dig)
{
  return nested[index][dig];
}
```

```
leaq   (%rdi,%rdi,4), %rax    # 5*index
addl   %rax, %rsi             # 5*index+dig
movl   nested(,%rsi,4), %eax # M[nested + 4*(5*index+dig)]
```

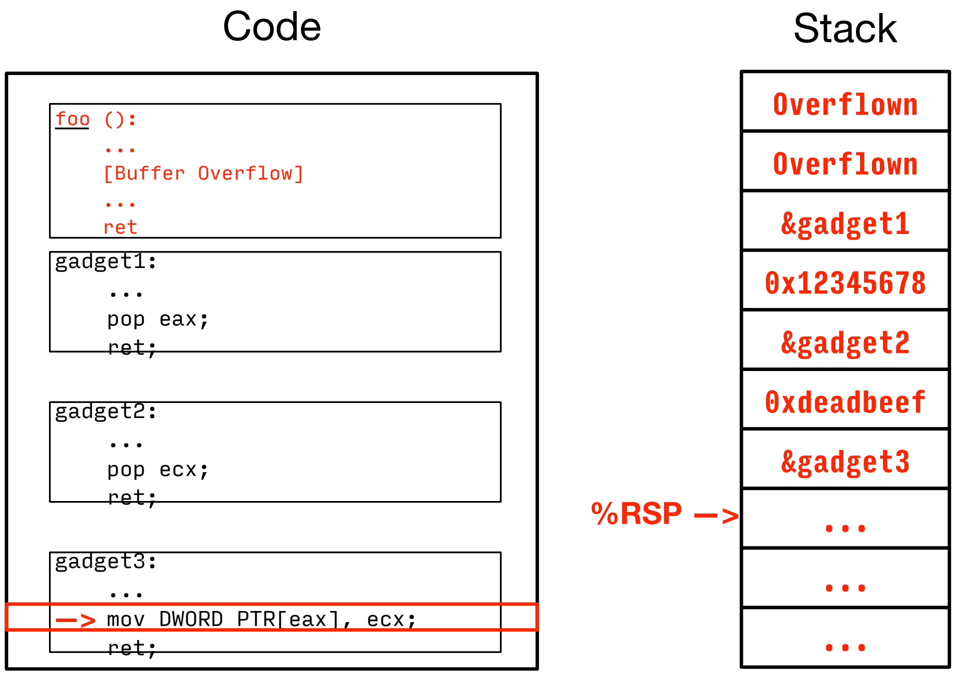
- **Array Elements**
  - **`nested[index][dig]`** is **`int`**
  - Address: **`nested + 20*index + 4*dig`**
    - = **`nested + 4*(5*index + dig)`**

Problem 7 Paging Terminologies (5pts, no partial pts)



Consider the figure above. A page that correspond to a *virtual address* may or may not exist in the page tables. If it exists, we call it a (a) __________ when it doesn’ t it is a (b) __________ the page indeeed *does not* exist, we have to copy the page content from the disk, which is called (c)__________.

Software Security and ROP

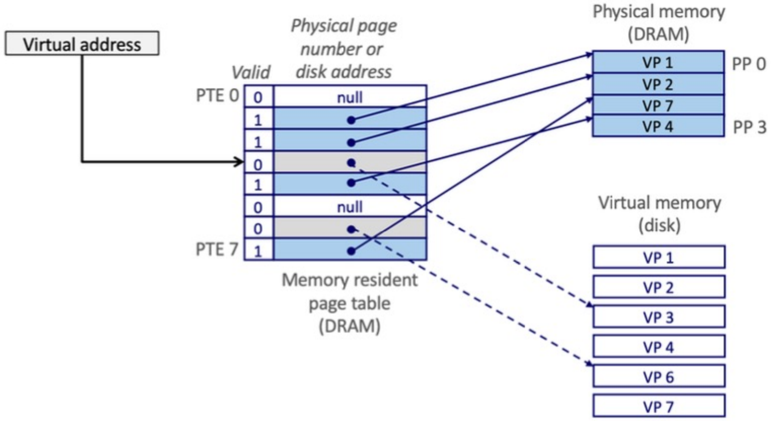


```
foo ():
    ...
    [Buffer Overflow]
    ...
    ret
gadget1:
    ...
    pop eax;
    ret;

gadget2:
    ...
    pop ecx;
    ret;

gadget3:
    ...
—> mov DWORD PTR[eax], ecx;
    ret;
```

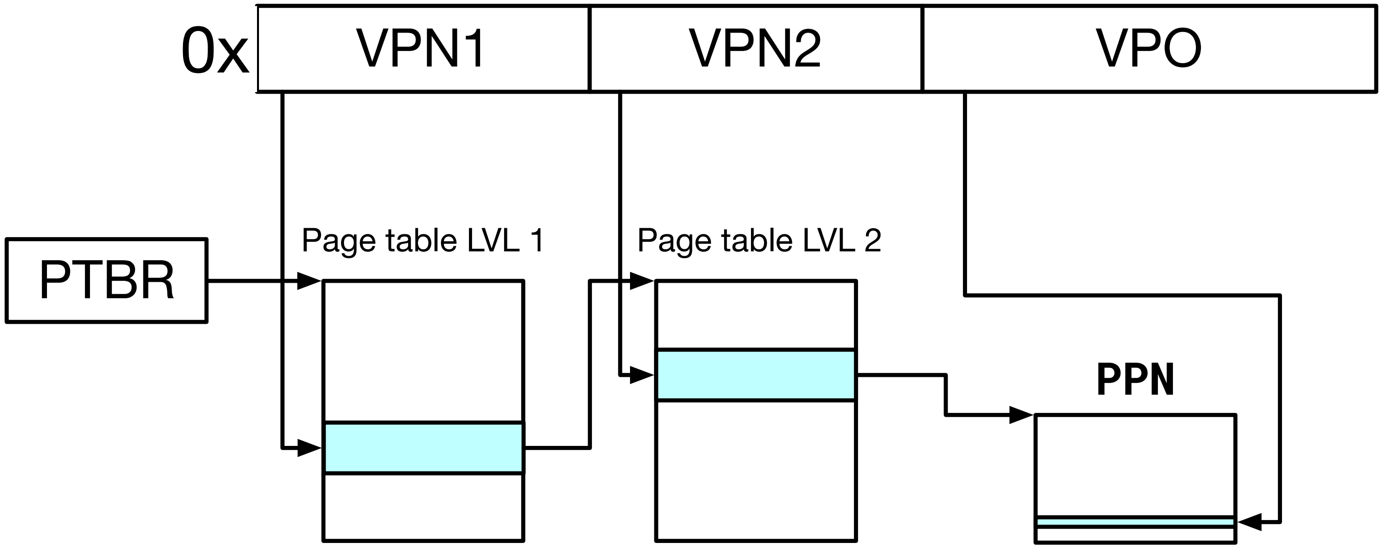
```
Overflown
Overflown
&gadget1
0x12345678
&gadget2
0xdeadbeef
&gadget3
...
...
...
```

- If you did Lab3, you will be fine
- It might take a little bit of creativity though

Problem 7 Paging Terminologies (5pts, no partial pts)

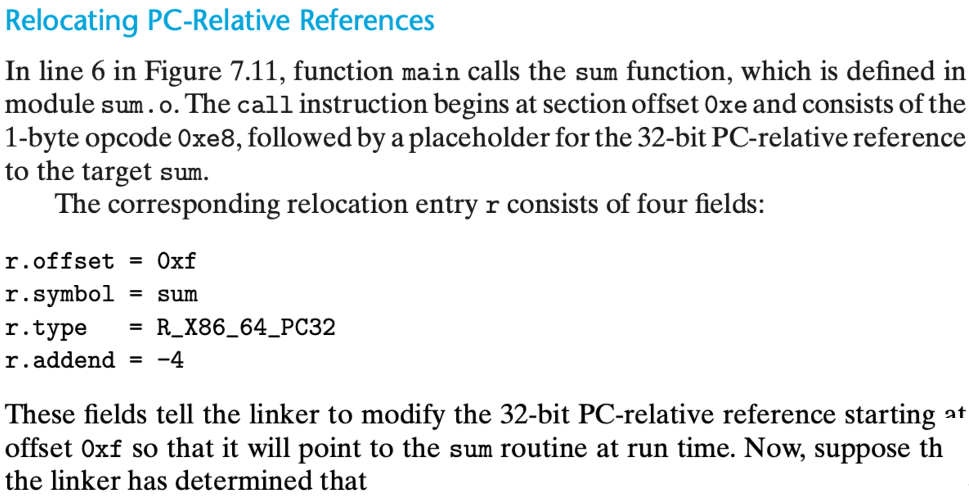


Consider the figure above. A page that correspond to a *virtual address* may or may not exist in the page tables. If it exists, we call it a (a) __________ when it doesn’ t it is a (b) __________ the page indeeed *does not* exist, we have to copy the page content from the disk, which is called (c)__________.


0x
VPN1
VPN2
VPO
PTBR
Page table LVL 1
Page table LVL 2
PPN

```
foreach section s {
    foreach relocation entry r {
        refptr = s + r.offset;  /* ptr to reference to be relocated */

        /* Relocate a PC-relative reference */
        if (r.type == R_X86_64_PC32) {
            refaddr = ADDR(s) + r.offset; /* ref's run-time address */
            *refptr = (unsigned) (ADDR(r.symbol) + r.addend - refaddr);
        }

        /* Relocate an absolute reference */
        if (r.type == R_X86_64_32)
            *refptr = (unsigned) (ADDR(r.symbol) + r.addend);
    }
}
```

Linker



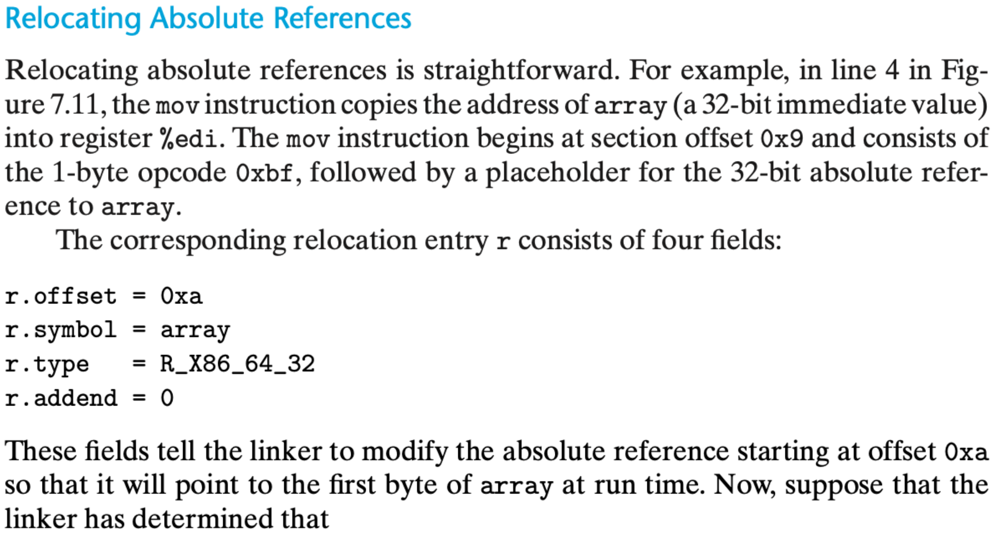
Relocating PC-Relative References

In line 6 in Figure 7.11, function `main` calls the `sum` function, which is defined in module `sum.o`. The `call` instruction begins at section offset `0xe` and consists of the 1-byte opcode `0xe8`, followed by a placeholder for the 32-bit PC-relative reference to the target `sum`.

The corresponding relocation entry `r` consists of four fields:

```
r.offset = 0xf
r.symbol = sum
r.type   = R_X86_64_PC32
r.addend = -4
```

These fields tell the linker to modify the 32-bit PC-relative reference starting at offset `0xf` so that it will point to the `sum` routine at run time. Now, suppose th the linker has determined that



Relocating Absolute References

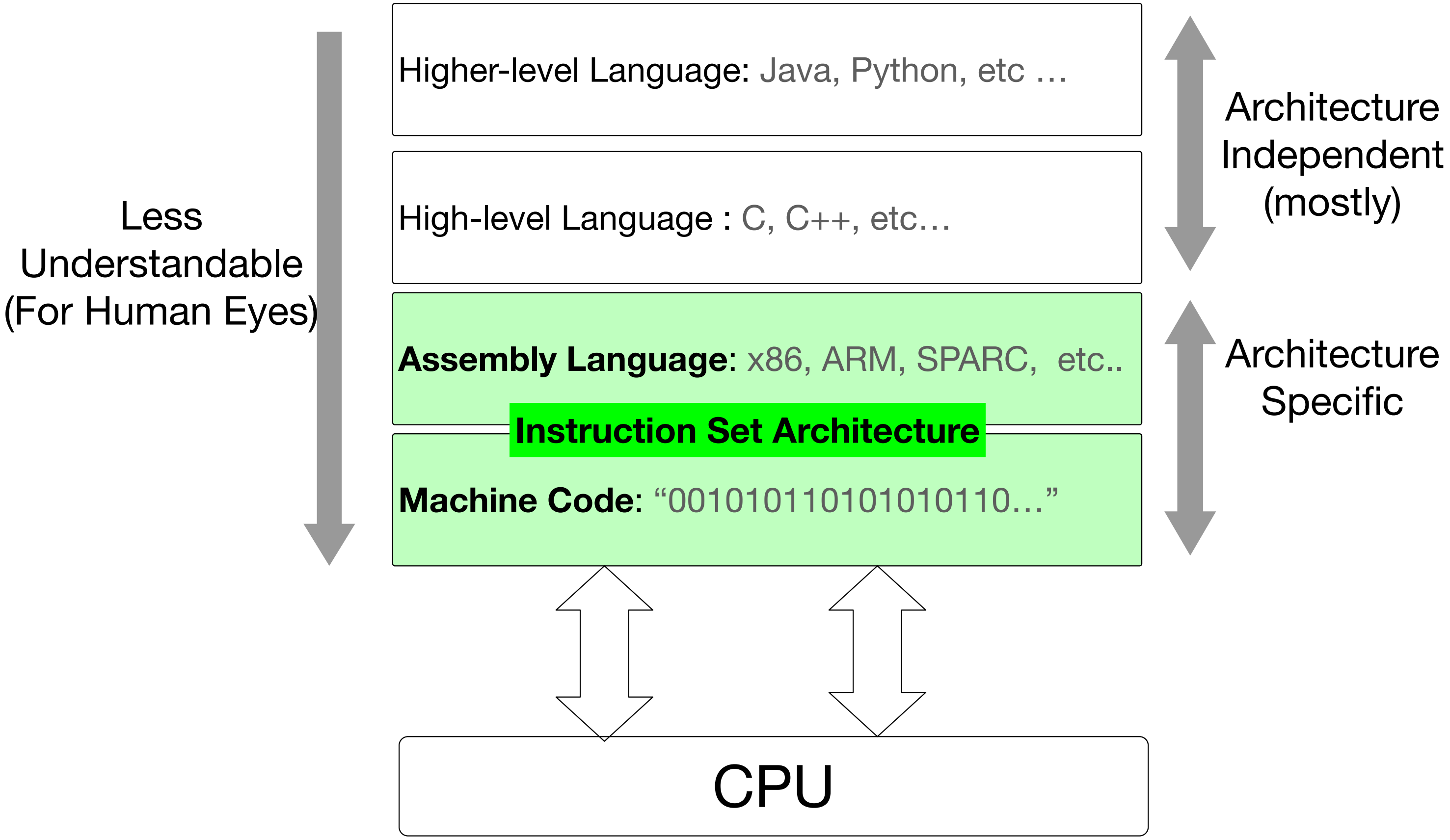
Relocating absolute references is straightforward. For example, in line 4 in Figure 7.11, the `mov` instruction copies the address of `array` (a 32-bit immediate value) into register `%edi`. The `mov` instruction begins at section offset `0x9` and consists of the 1-byte opcode `0xbf`, followed by a placeholder for the 32-bit absolute reference to `array`.

The corresponding relocation entry `r` consists of four fields:

```
r.offset = 0xa
r.symbol = array
r.type   = R_X86_64_32
r.addend = 0
```

These fields tell the linker to modify the absolute reference starting at offset `0xa` so that it will point to the first byte of `array` at run time. Now, suppose that the linker has determined that

Getting Low



Less Understandable (For Human Eyes)

Higher-level Language: Java, Python, etc …

High-level Language : C, C++, etc…

Assembly Language: x86, ARM, SPARC, etc..

Instruction Set Architecture

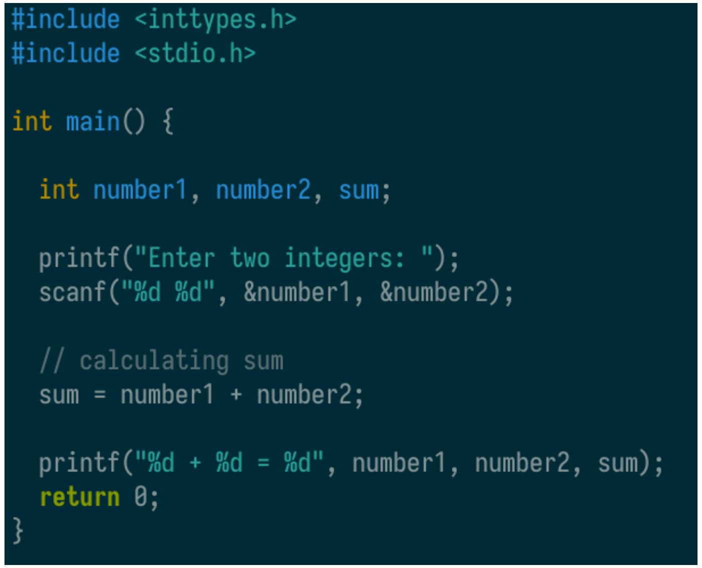
Machine Code: “00101011010101010110…”

CPU

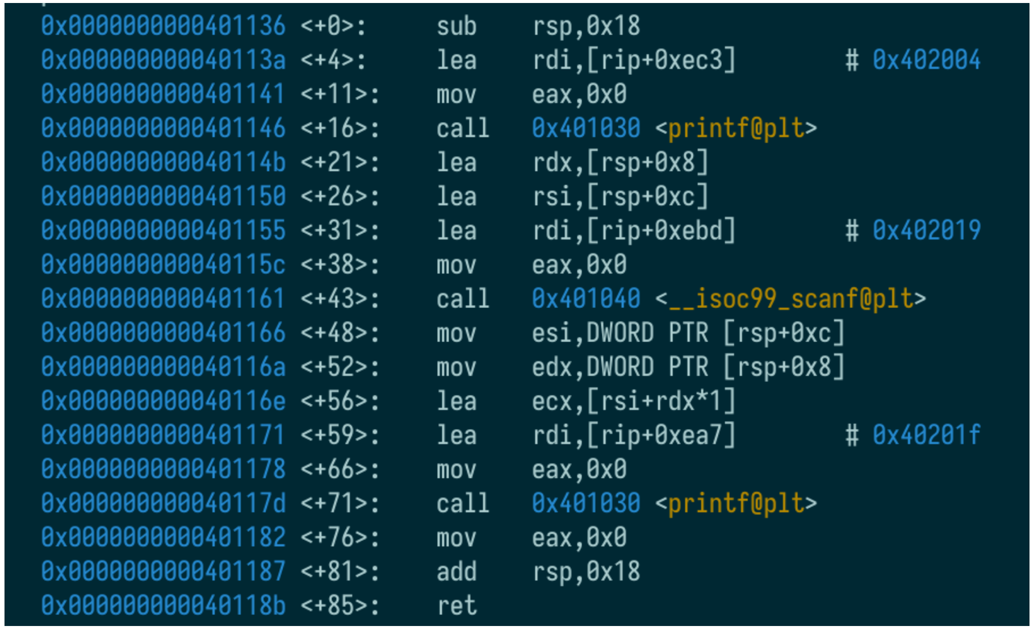
Architecture Independent (mostly)

Architecture Specific

Understanding Programs at Binary-level



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

int main() {

  int number1, number2, sum;

  printf("Enter two integers: ");
  scanf("%d %d", &number1, &number2);

  // calculating sum
  sum = number1 + number2;

  printf("%d + %d = %d", number1, number2, sum);
  return 0;
}
```

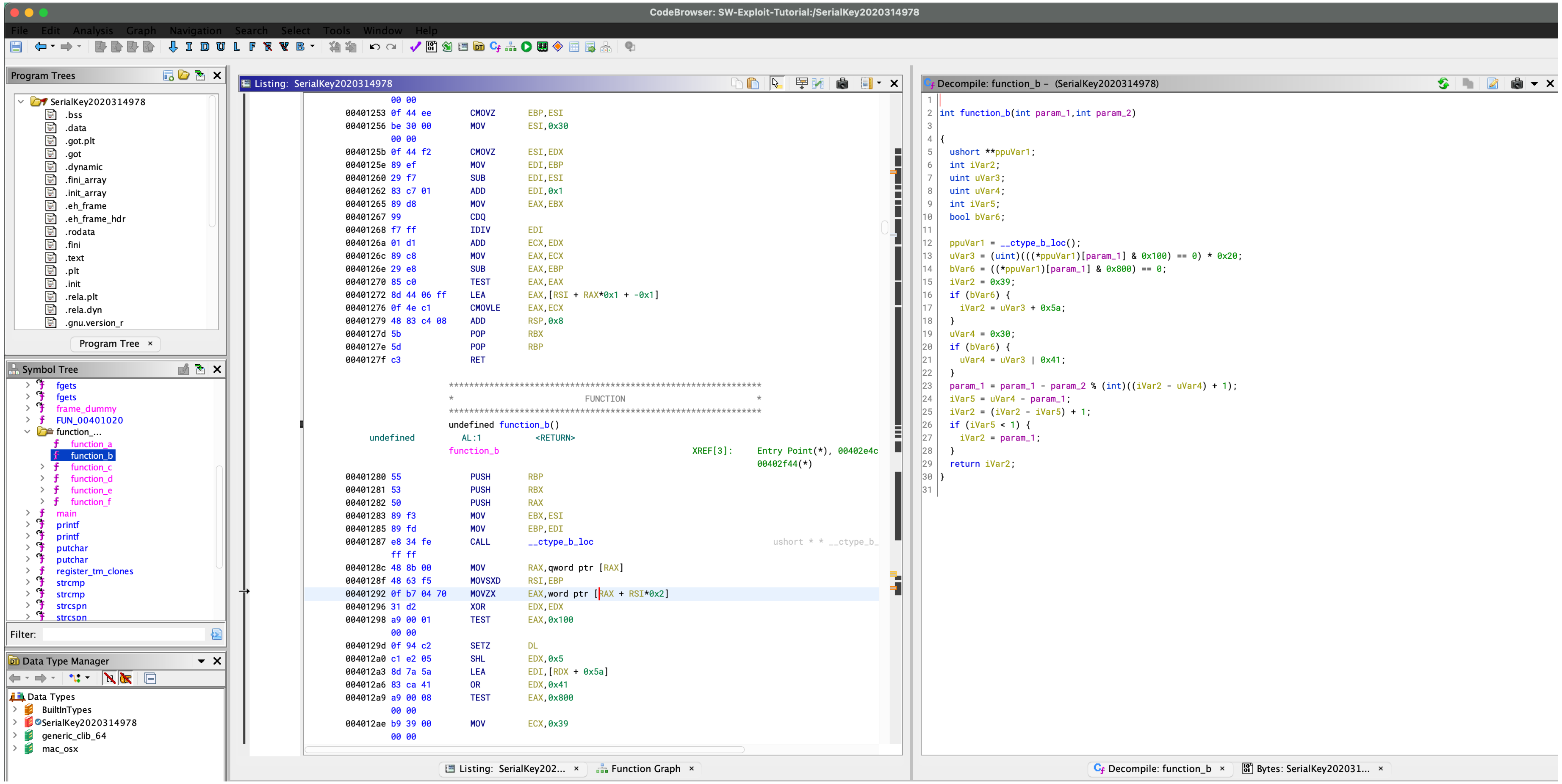


```
0x0000000000401136 <+0>:    sub    rsp,0x18
0x000000000040113a <+4>:    lea    rdi,[rip+0xec3]        # 0x402004
0x0000000000401141 <+11>:   mov    eax,0x0
0x0000000000401146 <+16>:   call   0x401030 <printf@plt>
0x000000000040114b <+21>:   lea    rdx,[rsp+0x8]
0x0000000000401150 <+26>:   lea    rsi,[rsp+0xc]
0x0000000000401155 <+31>:   lea    rdi,[rip+0xebd]        # 0x402019
0x000000000040115c <+38>:   mov    eax,0x0
0x0000000000401161 <+43>:   call   0x401040 <__isoc99_scanf@plt>
0x0000000000401166 <+48>:   mov    esi,DWORD PTR [rsp+0xc]
0x000000000040116a <+52>:   mov    edx,DWORD PTR [rsp+0x8]
0x000000000040116e <+56>:   lea    ecx,[rsi+rdx*1]
0x0000000000401171 <+59>:   lea    rdi,[rip+0xea7]        # 0x40201f
0x0000000000401178 <+66>:   mov    eax,0x0
0x000000000040117d <+71>:   call   0x401030 <printf@plt>
0x0000000000401182 <+76>:   mov    eax,0x0
0x0000000000401187 <+81>:   add    rsp,0x18
0x000000000040118b <+85>:   ret
```

Low-level Knowledge Helps You Write Better Code

- Be a *Programmer*, not a coder
- Have the intuition into how your program will interact with hardware
- You will natrually see why Code A will run faster than Code B

Advanced Reverse Engineering



```
                    00 00
     00401253 0f 44 ee        CMOVZ      EBP,ESI
     00401256 be 30 00        MOV        ESI,0x30
              00 00
     0040125b 0f 44 f2        CMOVZ      ESI,EDX
     0040125e 89 ef           MOV        EDI,EBP
     00401260 29 f7           SUB        EDI,ESI
     00401262 83 c7 01        ADD        EDI,0x1
     00401265 89 d8           MOV        EAX,EBX
     00401267 99              CDQ
     00401268 f7 ff           IDIV       EDI
     0040126a 01 d1           ADD        ECX,EDX
     0040126c 89 c8           MOV        EAX,ECX
     0040126e 29 e8           SUB        EAX,EBP
     00401270 85 c0           TEST       EAX,EAX
     00401272 8d 44 06 ff     LEA        EAX,[RSI + RAX*0x1 + -0x1]
     00401276 0f 4e c1        CMOVLE     EAX,ECX
     00401279 48 83 c4 08     ADD        RSP,0x8
     0040127d 5b              POP        RBX
     0040127e 5d              POP        RBP
     0040127f c3              RET

                        ****************************************************************
                        *                          FUNCTION                            *
                        ****************************************************************
                        undefined function_b()
          undefined       AL:1           <RETURN>
                        function_b                             XREF[3]:     Entry Point(*), 00402e4c
                                                                            00402f44(*)
     00401280 55              PUSH       RBP
     00401281 53              PUSH       RBX
     00401282 50              PUSH       RAX
     00401283 89 f3           MOV        EBX,ESI
     00401285 89 fd           MOV        EBP,EDI
     00401287 e8 34 fe        CALL       __ctype_b_loc                                 ushort * * __ctype_b_
              ff ff
     0040128c 48 8b 00        MOV        RAX,qword ptr [RAX]
     0040128f 48 63 f5        MOVSXD     RSI,EBP
     00401292 0f b7 04 70     MOVZX      EAX,word ptr [RAX + RSI*0x2]
     00401296 31 d2           XOR        EDX,EDX
     00401298 a9 00 01        TEST       EAX,0x100
              00 00
     0040129d 0f 94 c2        SETZ       DL
     004012a0 c1 e2 05        SHL        EDX,0x5
     004012a3 8d 7a 5a        LEA        EDI,[RDX + 0x5a]
     004012a6 83 ca 41        OR         EDX,0x41
     004012a9 a9 00 08        TEST       EAX,0x800
              00 00
     004012ae b9 39 00        MOV        ECX,0x39
              00 00
```

```c

int function_b(int param_1,int param_2)

{
  ushort **ppuVar1;
  int iVar2;
  uint uVar3;
  uint uVar4;
  int iVar5;
  bool bVar6;

  ppuVar1 = __ctype_b_loc();
  uVar3 = (uint)(((*ppuVar1)[param_1] & 0x100) == 0) * 0x20;
  bVar6 = ((*ppuVar1)[param_1] & 0x800) == 0;
  iVar2 = 0x39;
  if (bVar6) {
    iVar2 = uVar3 + 0x5a;
  }
  uVar4 = 0x30;
  if (bVar6) {
    uVar4 = uVar3 | 0x41;
  }
  param_1 = param_1 - param_2 % (int)((iVar2 - uVar4) + 1);
  iVar5 = uVar4 - param_1;
  iVar2 = (iVar2 - iVar5) + 1;
  if (iVar5 < 1) {
    iVar2 = param_1;
  }
  return iVar2;
}

```

Systems Security Lab

SUNGKYUNKWAN UNIVERSITY 1398 성균관대학교

If you enjoyed reversing and exploiting

- See you in security courses I teach {Introduction to Information Security, Internet Security, Special Topics in Systems Security (Graduate)}
- I own ctf.skku.edu, where we have Capture-The-Flag competitions for classes and also just for fun

Hackers Wanted

시스템
보안
연구실

- We are always open to undergraduate internships
- If you enjoy hacking stuff please come and join us

Our Interesting Projects

시스템
보안
연구실

- **Defenseive:** Just-in-Time program transformation for security
- **Offensive:** ML model stealing
- **Offensive:** Bypassing modern hardware security features in 2021
- #1 Rule in research topic @ SSLab
  - The student must enjoy it

Thank you for your hard work,
good luck on your exam,
and see you latger