Software Reverse Engineering (Practice)

ICS312

Machine-Level and Systems Programming

Henri Casanova (henric@hawaii.edu)

Software Reverse Engineering

- RE (Reverse Engineering) is the process of understanding how an existing system accomplishes its tasks
  - i.e., how it was “engineered”
- SER (Software Reverse Engineering) is RE for software system
- It can be done with any kind of software, but typically it is applied to binary code
- It has many uses:
  - Malware analysis, legacy code modification, intellectual property disputes, …
- Like most things, it can be used unethically
  - e.g., reverse engineering licensed software and modify the code to remove the license check (i.e., software piracy)

SRE: Approaches

- First step: disassemble a binary into assembly code
  - For NASM, we have used the **`ndisasm`** disassembler
  - There are much more powerful (but expensive) disassembler tools used by reverse engineering, such as IDA
  - These tools present the user with the flowchart of the program where the jump instructions are arrows that connect blocks of assembly code
- **Static analysis:** look at the code and figure out what it does
  - Identify functions, control structures, libraries used, etc.
- **Dynamic analysis:** run the code and observe its execution (e.g., with a debugger)
  - Makes it possible to observe its behavior on different input (or to modify memory content on the fly)
- **Decompilation:** translate it back to high-level code
  - Sort of static analysis on steroids
  - If done correctly, very useful
  - Difficult due to code optimization (see a later module), or possibly code obfuscation techniques (see a later video, if time)

SRE Practice Problems

- A big SRE hurdle for most people is learning/ understanding assembly code
- But at this point in this course, you can now perform various SRE tasks
- What follows is a sequence of practice problems in which we decompile assembly code:
  - i.e., we translate from assembly code to high-level code
- Highly recommended: Take EE491F
  - A full SRE course!
- This completes all material for Midterm #3

(q1) SRE Example

```
f:
  push ebp
  mov ebp, esp
  mov eax, 12
  cmp [EBP+8], 0
  jne here
  inc eax
here:
  pop ebp
  ret
```

- Translate back to whatever high-level language…

(q1) Solutions

```
f:
  push ebp
  mov ebp, esp
  mov eax, 12
  cmp [EBP+8], 0
  jne here
  inc eax
here:
  pop ebp
  ret
```

```
int f(int x) {
  if (x != 0) {
    return 12;
  } else {
    return 13;
  }
}
```

```
int f(int x) {
  return 12 + (x == 0);
}
```

```
def f(x: int):
  if x != 0:
    return 12
  else:
    return 13
}
```

(q2) SRE Example

```
f:
  push ebp
  mov ebp, esp
  mov eax, [ebp+8]
  cmp eax, [ebp+12]
  jge here
  mov eax, [ebp+12]
  jmp there
here:
  mov eax, [ebp+8]
there:
  pop ebp
  ret
```

- Translate back to whatever high-level language…

(q2) Solutions

```
f:
  push ebp
  mov ebp, esp
  mov eax, [ebp+8]
  cmp eax, [ebp+12]
  jge here
  mov eax, [ebp+12]
  jmp there
here:
  mov eax, [ebp+8]
there:
  pop ebp
  ret
```

```
int f(int x, int y) {
  if (x >= y) {
    return x;
  } else {
    return y;
  }
}
```

```
def f(x: int, y: int):
  return max(x,y)
```

(q3) SRE Example

```
  mov eax, 0
  mov ecx, 0
here:
  mov edx, ecx
  shl edx, 2
  add eax, edx
  inc ecx
  cmp ecx, 20
  jne here
  call print_int
  call print_nl
```

- Translate back to whatever high-level language…

(q3) Solutions

```
  mov eax, 0
  mov ecx, 0
here:
  mov edx, ecx
  shl edx, 2
  add eax, edx
  inc ecx
  cmp ecx, 20
  jne here
  call print_int
  call print_nl
```

```
int eax = 0;
for (int ecx = 0; ecx != 20; ecx++) {
  eax += 4 * ecx;
}
printf("%d\n", eax);
```

```
int eax = 0;
int ecx = 0;
while (1) {
  eax += 4 * ecx;
  if (++ecx == 20) break;
}
printf("%d\n", eax);
```

```
x = sum([4*x for x in range(0,20)])
print(x)
```

(q4) SRE Example

```
  mov eax, 0AABBCC12h
here:
  cmp byte [eax], 0
  je  exit
  mov byte [eax], 0
  inc eax
  jmp here
exit:
```

- Translate back to C…

(q4) Solutions

```
  mov eax, 0AABBCC12h
here:
  cmp byte [eax], 0
  je  exit
  mov byte [eax], 0
  inc eax
  jmp here
exit:
```

```
char *ptr = 0xAABBCC12;
while (*ptr) {
  *ptr = 0;
  ptr++;
}
```

(q5) SRE Example

```
segment .data
  dd a -1
  db c 12
segment .text
  . . .
  mov eax, [a]
  movzx ebx, [c]
  add eax, ebx
  call print_int
  call print_nl
  . . .
```

(q5) Solutions

```
segment .data
  dd a -1
  db c 12
segment .text
  . . .
  mov eax, [a]
  movzx ebx, [c]
  add eax, ebx
  call print_int
  call print_nl
  . . .
```

```
int a = -1;
unsigned char c = 12;
. . .
printf("%d\n", (a + (unsigned int)c));
. . .
```

(q6) SRE Example

```
%include "asm_io.inc"
segment .data
        a       dd      3
segment .bss
        b       resd    1
segment .text
        global  asm_main
asm_main:
        enter   0,0
        pusha
        push    dword   4
        push    dword   6
        push    dword   [a]
        call    f
        add     esp, 12
        mov     [b], eax
        call    print_int
        call    print_nl
        popa
        mov     eax, 0
        leave
        ret
```

```
f:
        push    ebp
        mov     ebp, esp
        sub     esp, 4
        mov     dword [ebp-4], 1
        cmp     dword [ebp+8], 5
        jz      endf
        mov     ebx, [ebp+8]
        inc     ebx
        push    ebx
        push    dword [ebp+12]
        push    dword [ebp+16]
        call    f
        add     esp, 12
        mov     ebx, [ebp+12]
        add     ebx, eax
        mov     [ebp-4], ebx
endf:
        mov eax, [ebp-4];
        mov esp, ebp
        pop ebp
        ret
```

(q6) Solutions

```
int a = 3;
int b;

int main() {
  b = f(a, 6, 4);
  printf("%d\n",b);
  return 0;
}

int f(int x, int y, int z) {
  int w;
  w = 1;
  if (x == 5) {
    return w;
  } else {
   w = y + f(z, y, x+1);
   return w;
  }
}
```