CS451 – Software Analysis

Lecture 9

Custom Disassembly (Linear)

Elias Athanasopoulos
athanasopoulos.elias@ucy.ac.cy

Linear disassembly

- Extracting the machine code in symbolic language
- Starts from a specific location in the binary or in the process and each byte is decoded as an opcode
  - Static: the binary is read from the disk
  - Dynamic: the binary is first mapped as a process and then its memory is read

Example

```
$ objdump -d disas-test.o
disas-test.o:     file format elf64-x86-64

Disassembly of section .text:


0000000000000000 <foo>:
   0: 55                    push   %rbp
   1: 48 89 e5              mov    %rsp,%rbp
   4: b8 01 00 00 00        mov    $0x1,%eax
   9: 5d                    pop    %rbp
   a: c3                    retq


000000000000000b <bar>:
   b: 55                    push   %rbp
   c: 48 89 e5              mov    %rsp,%rbp
   f: b8 02 00 00 00        mov    $0x2,%eax
  14: 5d                    pop    %rbp
  15: c3                    retq
```

Reading the output

- A disassembly most of the times would include
  - Address of the opcode
  - The value of the opcode, which is the actual machine code
  - Symbolic representation of the opcode

Some problems

- Where to start the disassembly?
  - In general, there is no clear starting point, since every byte can be a valid opcode
  - Binaries have sections that contain code and sections that contain data
  - When disassembling memory, code is mapped in different regions than data
- Where to stop the disassembly?
  - In some cases, the disassembler may keep up until a non-valid opcode is parsed or until a specific instruction is decoded (e.g., a `ret` may denote the end of a function)

Disassembling a section

```
$ objdump  -j.comment -d ./disas-test.o

./disas-test.o:     file format elf64-x86-64

Disassembly of section .comment:

0000000000000000 <.comment>:
   0: 00 47 43             add    %al,0x43(%rdi)
   3: 43 3a 20             rex.XB cmp (%r8),%spl
   6: 28 47 4e             sub    %al,0x4e(%rdi)
   9: 55                   push   %rbp
   a: 29 20                sub    %esp,(%rax)
   c: 38 2e                cmp    %ch,(%rsi)
   [...]
```

Selecting the end points

```
$ objdump  --start-address=4 --stop-address=0xa
-d ./disas-test.o


./disas-test.o:     file format elf64-x86-64

Disassembly of section .text:

0000000000000004 <foo+0x4>:
   4: b8 01 00 00 00        mov    $0x1,%eax
   9: 5d                    pop    %rbp
```

Custom disassembly

- Sometimes it is useful to write your own disassembler
  - For incorporating different strategies (linear is not the only option)
  - For applying other analysis techniques
- Many tools may need to support disassembly in part
  - A debugger may support the disassembling of code mapped in memory
- Obfuscated code
  - Code may be on purpose designed to evade simple disassembling

Obfuscated code

```
int overlapping(int i) {
  int j = 0;

  __asm__ __volatile__(
  "cmp   $0x0,%1          ; "
  ".byte 0x0f,0x85        ; " /* relative jne */
  ".long 2                ; " /* jne offset   */
  "xorl  $0x04,%0         ; "
  ".byte 0x04,0x90        ; " /* add al,0x90  */
  : "=r" (j)
  : "r" (i)
  :
  );
  return j;
}

int main(int argc, char *argv[])
{
  srand(time(NULL));
  printf("%d\n", overlapping(rand() % 2));

  return 0;
}
```

Using objdump with obfuscated code

```
$ objdump --start-address=0x400666 --stop-address=0x40068c -d overlapping_bb
0000000000400666 <overlapping>:
  400666: 55                    push   %rbp
  400667: 48 89 e5              mov    %rsp,%rbp
  40066a: 89 7d ec              mov    %edi,-0x14(%rbp)
  40066d: c7 45 fc 00 00 00 00  movl   $0x0,-0x4(%rbp)
  400674: 8b 45 ec              mov    -0x14(%rbp),%eax
  400677: 83 f8 00              cmp    $0x0,%eax
  40067a: 0f 85 02 00 00 00     jne    400682 <overlapping+0x1c>
  400680: 83 f0 04              xor    $0x4,%eax
  400683: 04 90                 add    $0x90,%al
  400685: 89 45 fc              mov    %eax,-0x4(%rbp)
  400688: 8b 45 fc              mov    -0x4(%rbp),%eax
  40068b: 5d                    pop    %rbp
$ objdump --start-address=0x400682 --stop-address=0x40068c -d overlapping_bb
0000000000400682 <overlapping+0x1c>:
  400682: 04 04                 add    $0x4,%al
  400684: 90                    nop
  400685: 89 45 fc              mov    %eax,-0x4(%rbp)
  400688: 8b 45 fc              mov    -0x4(%rbp),%eax
  40068b: 5d                    pop    %rbp
```

Capstone

- A disassembly framework with a simple API that supports many popular instruction sets
  - x86/x86-64, ARM, MIPS, etc.
- Capstone is written in C
  - Several bindings have been developed for using the framework through other programming languages, such as Python
- Capstone is open source
  - The code can be compiled to run in most popular operating systems (Linux, macOS, Windows)

Requirements

```
$ yum list installed | grep capstone
capstone.x86_64          4.0.2-5.el8   @epel
capstone-devel.x86_64    4.0.2-5.el8   @epel
```

Disassembling a buffer

```
     /* Buffer with code.  */
    const unsigned char * code = (unsigned char *)
"\x55\x48\x89\xe5\x48\x83\xec\x10\x89\x7d\xfc\x48\x89\x75\xf0\xbf\x00\x00
\x00\x00\xe8\x00\x00\x00\x00\xb8\x00\x00\x00\x00\xc9\xc3";

    /* Initialize the engine.  */
    csh handle;
    if (cs_open(CS_ARCH_X86, CS_MODE_64, &handle) != CS_ERR_OK)
        return -1;

    /* AT&T */
    cs_option(handle, CS_OPT_SYNTAX, CS_OPT_SYNTAX_ATT);

    disas(handle, code);

    cs_close(&handle);
```

The disassembly routine

```
void disas(csh handle, const unsigned char *buffer) {
    cs_insn *insn;
    size_t count;

    count = cs_disasm(handle, buffer, 32, 0x0, 0, &insn);

    if (count > 0) {
        size_t j;
        for (j = 0; j < count; j++) {
            fprintf(stderr, "0x%"PRIx64":\t%s\t\t%s\n",
                    insn[j].address, insn[j].mnemonic, insn[j].op_str);
        }
        cs_free(insn, count);
    } else
        fprintf(stderr, "ERROR: Failed to disassemble given code.\n");
}
```

Output

```
$ ./disas-mem
0x0:    pushq %rbp
0x1:    movq %rsp, %rbp
0x4:    subq $0x10, %rsp
0x8:    movl %edi, -4(%rbp)
0xb:    movq %rsi, -0x10(%rbp)
0xf:    movl $0, %edi
0x14:   callq 0x19
0x19:   movl $0, %eax
0x1e:   leave
0x1f:   retq
```

Disassembling a file

- Use libelf to load the ELF
- Find the .text section
- Use Capstone to disassemble the code

Load the file

```
void disas_file(char *filename, csh handle) {
    Elf *elf;

    /* Initialization.  */
    if (elf_version(EV_CURRENT) == EV_NONE)
        DIE("(version) %s", elf_errmsg(-1));

    int fd = open(filename, O_RDONLY);

    elf = elf_begin(fd, ELF_C_READ, NULL);
    if (!elf)
        DIE("(begin) %s", elf_errmsg(-1));
[...]
```

Find .text

```
    /* Loop over sections.  */
    Elf_Scn *scn = NULL;
    GElf_Shdr shdr;
    size_t shstrndx;
    if (elf_getshdrstrndx(elf, &shstrndx) != 0)
        DIE("(getshdrstrndx) %s", elf_errmsg(-1));

    while ((scn = elf_nextscn(elf, scn)) != NULL) {
        if (gelf_getshdr(scn, &shdr) != &shdr)
            DIE("(getshdr) %s", elf_errmsg(-1));

        /* Locate .text  */
        if (!strcmp(elf_strptr(elf, shstrndx, shdr.sh_name), ".text")) {
            Elf_Data *data = NULL;
            size_t n = 0;

            data = elf_getdata(scn, data);
            disas(handle, data->d_buf, data->d_size);
        }
    }
}
```

Disassemble .text

```
void disas(csh handle, const unsigned char *buffer, unsigned int size) {
    cs_insn *insn;
    size_t count;

    count = cs_disasm(handle, buffer, size, 0x0, 0, &insn);

    if (count > 0) {
        size_t j;
        for (j = 0; j < count; j++) {
            fprintf(stderr, "0x%"PRIx64":\t%s\t\t%s\n",
                        insn[j].address, insn[j].mnemonic,
                        insn[j].op_str);
        }
        cs_free(insn, count);
    } else
        fprintf(stderr, "ERROR: Failed to disassemble given code!\n");
}
```

Homework

- Write a C program that can load and disassemble, using Capstone, the PLT section of an ELF executable