Karlsruhe Institute of Technology

Institute for Theoretical Computer Science
Prof. Dr. J. Müller-Quade

Application Security Lab – Winter Term 2023/2024

Exercise Sheet 1

November 02, 2023 – Due: November 16, 2023, 8:00 am

Create an SSH-key (e.g. with `ssh-keygen -t ed25519`) and upload the contents of the public-key file to the submission system once you receive access. This SSH-key will be required to access the target applications from the second exercise sheet onwards. Please make sure to upload an SSH-key before the deadline of this sheet. We only deploy SSH-keys with the deployment of new exercise sheets, if you change your keys later, give us a note so we can update the keys of the currently deployed execises manually.

Exercise 1 (Assembler)**.** Which well-known algorithm is implemented by the assembler code in Figure 1? Explain in a few sentences, which part of the algorithm is implemented by which parts of the assembler code.

```
global f

f:
    push rbp
    mov rbp, rsp
    cmp rsi, 1
    jbe .l4
    cmp rsi, 2
    jbe .l3
    mov r8, [rdi]
    xor rcx, rcx
    mov rdx, rsi
.l0:
    inc rcx
    cmp rcx, rdx
    jae .l2
    mov rax, [rdi + 8 * rcx]
    cmp rax, r8
    jbe .l0
.l1:
    dec rdx
    cmp rdx, rcx
    jbe .l2
    mov rax, [rdi + 8 * rdx]
    cmp rax, r8
    ja .l1
    mov rax, [rdi + 8 * rcx]
    xchg rax, [rdi + 8 * rdx]
    mov [rdi + 8 * rcx], rax
    jmp .l0
.l2:
    dec rcx
    xchg r8, [rdi + 8 * rcx]
    mov [rdi], r8
    push rdi
    push rsi
    push rcx
    mov rsi, rcx
    call f
    pop rcx
    pop rsi
    pop rdi
    inc rcx
    sub rsi, rcx
    lea rdi, [rdi + rcx * 8]
    call f
    jmp .l4
.l3:
    mov rax, [rdi]
    mov rcx, [rdi + 8]
    cmp rax, rcx
    jbe .l4
    mov [rdi], rcx
    mov [rdi + 8], rax
.l4:
    leave
    ret
```

Figure 1: Assembler Code for Exercise 1

Exercise 2 (gdb)**.** You are given an executable binary. The main function calls two other functions, one of which returns a value that is passed as a parameter to the second function. The second function will (mostly) ignore this parameter. Determine the parameter value using `gdb`. Your flag is the lower-case hexadecimal representation of this parameter without leadings zeros, but including the preceding `0x`. (Example: `0x45ca0f36ceba4280`.) Upload a text file that explains in a few sentences how you got the flag and which `gdb` commands you used.

Exercise 3 (Calling Conventions)**.** You are given a 64-bit ELF binary, which contains a function named `foo_system_v`. This function takes eight unsigned 64-bit integers as parameters und uses the System V x64 calling convention. Write a function `foo_ms_abi` in assembler, having the same signature, that calls `foo_system_v` and returns the result of this call. `foo_ms_abi` shall use the Microsoft x64 calling convention.

Remark. Although the function `foo_system_v` is very simple, do not simply re-implement it in `foo_ms_abi`. Write code that "bridges" the differences between the calling conventions.

Hint: You can use the provided `Makefile` to compile your code and create a binary that tries to run the function. If the binary crashes, your solution is most likely not correct. Use `gdb` to debug your solution.

Hint: The given test code may not verify that your code matches *all* of the requirements for the target calling convention presented in the lab. Make sure your solution fulfills the requirements not tested, too.

Hint: Your solution should be generic enough to call *any* function with the same signature, by just replacing the name of the function called.