20875 Software Engineering

Tutorial 2 – errata

- Question 3. The simplest way to solve the exercise is to look for strings that could be a password in the executable file itself. As a (much harder but) more systematic alternative, we can analyse the disassembly of its code. The output of `objdump -M intel -d secret` gives us the following for the `main` function:

```
0000000000401240 <main>:
  401240:       41 55                   push   r13
  401242:       41 b8 2c 30 40 00       mov    r8d,0x40302c
  401248:       b9 60 30 40 00          mov    ecx,0x403060
  40124d:       31 c0                   xor    eax,eax
  40124f:       41 54                   push   r12
  401251:       ba 90 30 40 00          mov    edx,0x403090
  401256:       55                      push   rbp
  401257:       89 fd                   mov    ebp,edi
  401259:       53                      push   rbx
  40125a:       48 89 f3                mov    rbx,rsi
  40125d:       be 10 30 40 00          mov    esi,0x403010
  401262:       48 81 ec 48 18 00 00    sub    rsp,0x1848
  401269:       48 8b 3d 90 47 00 00    mov    rdi,QWORD PTR [rip+0x4790]        # 405a00 <stderr@GLIBC_2.2.5>
  401270:       e8 6b fe ff ff          call   4010e0 <fprintf@plt>
  401275:       ba 10 00 00 00          mov    edx,0x10
  40127a:       be 01 00 00 00          mov    esi,0x1
  40127f:       48 8b 0d 7a 47 00 00    mov    rcx,QWORD PTR [rip+0x477a]        # 405a00 <stderr@GLIBC_2.2.5>
  401286:       bf 1c 30 40 00          mov    edi,0x40301c
  40128b:       e8 40 ff ff ff          call   4011d0 <fwrite@plt>
  401290:       31 c0                   xor    eax,eax
  401292:       48 89 e6                mov    rsi,rsp
  401295:       bf 2d 30 40 00          mov    edi,0x40302d
  40129a:       e8 11 ff ff ff          call   4011b0 <__isoc99_scanf@plt>
  40129f:       48 b8 4d 79 53 65 63    movabs rax,0x746572636553794d
  4012a6:       72 65 74
  4012a9:       48 33 04 24             xor    rax,QWORD PTR [rsp]
  4012ad:       48 ba 50 61 73 73 77    movabs rdx,0x64726f7773736150
  4012b4:       6f 72 64
  4012b7:       48 33 54 24 08          xor    rdx,QWORD PTR [rsp+0x8]
  4012bc:       48 09 d0                or     rax,rdx
  4012bf:       74 2e                   je     4012ef <main+0xaf>
  4012c1:       48 89 e6                mov    rsi,rsp
  4012c4:       bf 43 30 40 00          mov    edi,0x403043
  4012c9:       31 c0                   xor    eax,eax
  4012cb:       e8 c0 fd ff ff          call   401090 <printf@plt>
  4012d0:       83 fd 02                cmp    ebp,0x2
  4012d3:       74 23                   je     4012f8 <main+0xb8>
  4012d5:       bf c8 30 40 00          mov    edi,0x4030c8
  4012da:       e8 81 fd ff ff          call   401060 <puts@plt>
  4012df:       48 81 c4 48 18 00 00    add    rsp,0x1848
  4012e6:       31 c0                   xor    eax,eax
  4012e8:       5b                      pop    rbx
  4012e9:       5d                      pop    rbp
  4012ea:       41 5c                   pop    r12
  4012ec:       41 5d                   pop    r13
  4012ee:       c3                      ret
  4012ef:       80 7c 24 10 00          cmp    BYTE PTR [rsp+0x10],0x0
  4012f4:       74 da                   je     4012d0 <main+0x90>
  4012f6:       eb c9                   jmp    4012c1 <main+0x81>
```

As suggested by a student in class, the comparison of the user's input with the hidden password is performed just after the call to `scanf()` using the `xor` instructions:

```
40129a:       e8 11 ff ff ff          call   4011b0 <__isoc99_scanf@plt>
40129f:       48 b8 4d 79 53 65 63    movabs rax,0x746572636553794d
4012a6:       72 65 74
4012a9:       48 33 04 24             xor    rax,QWORD PTR [rsp]
4012ad:       48 ba 50 61 73 73 77    movabs rdx,0x64726f7773736150
4012b4:       6f 72 64
4012b7:       48 33 54 24 08          xor    rdx,QWORD PTR [rsp+0x8]
4012bc:       48 09 d0                or     rax,rdx
4012bf:       74 2e                   je     4012ef <main+0xaf>
...
4012ef:       80 7c 24 10 00          cmp    BYTE PTR [rsp+0x10],0x0  # <-- <main+0xaf>
4012f4:       74 da                   je     4012d0 <main+0x90>
```

First, `movabs rax,0x746572636553794d` moves the hexadecimal value `0x746572636553794d` into the register `rax` (`movabs` is a synonym for `mov`). Since we are on a little-endian ISA, that value corresponds to the bytes `0x4d, 0x79, 0x53, 0x65, 0x63, 0x72, 0x65, 0x74`. Referring an ASCII table, we can find that they correspond to the characters that form the first half of the password. Then, the instruction `xor rax,QWORD PTR [rsp]` does a bitwise `xor` of `rax` with the first 8 bytes at `[rsp]`, i.e. the first eight bytes of the buffer passed to `scanf`. If those bytes have a value exactly equal to that if `rax`, the result of the `xor` (stored in `rax` again) will be zero. It will be nonzero otherwise. The same thing happens for the next 8 bytes and `rdx`: they are compared to `0x64726f7773736150`, i.e. `0x50, 0x61, 0x73, 0x73, 0x77, 0x6f, 0x72, 0x64`. If both `rax` and `rdx` are zero, then we jump to `main + 0xaf` and check that `[rsp+16]` is zero. Thus, our password is 16 bytes long with values: `0x4d, 0x79, 0x53, 0x65, 0x63, 0x72, 0x65, 0x74, 0x50, 0x61, 0x73, 0x73, 0x77, 0x6f, 0x72, 0x64`.

- Exercise 8: I used the `-size` parameter of the `magick` command whereas I should have used the `-scale` parameter. The final `Makefile` is as follows:

```
IMAGES := $(shell ls source/*.jpeg)
THUMB := $(IMAGES:source/%.jpeg=thumbnail/%.jpeg)

.PHONY: default

default: $(THUMB)

$(THUMB) : thumbnail/%.jpeg : source/%.jpeg Makefile
        magick $(<) -scale "256x256" $(@)
```

A few links to the `magick` documentation:

  - General `magick` command documentation
  - `-scale` parameter
  - `-size` parameter
  - geometry specification