CTF name: Antisyphon Cyber Range

Challenge name: Mozzarella

Challenge description: Check this out this cheese.

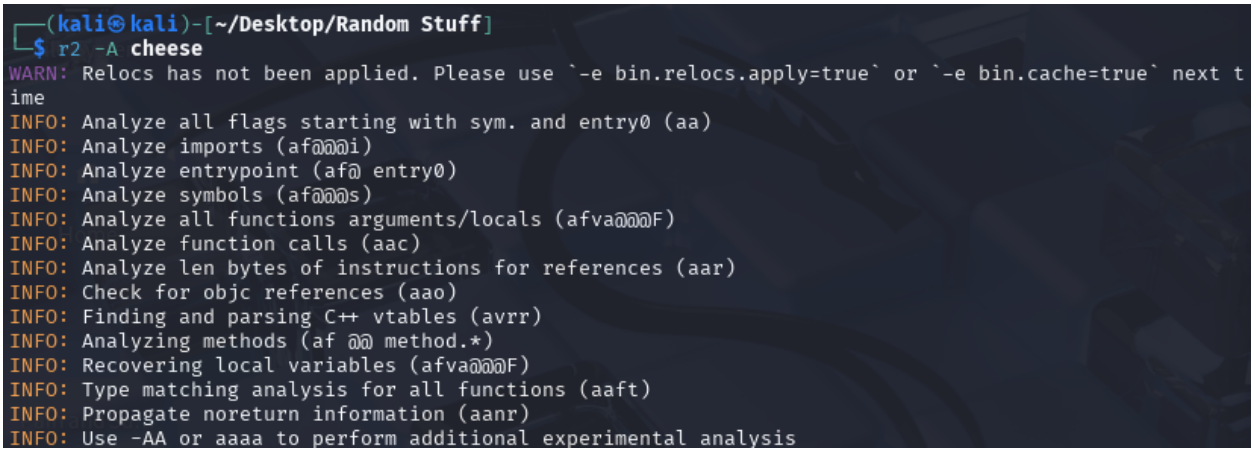
Challenge category: Reverse Engineering

Challenge points: 25

We are presented with a file called **cheese** and given simple instructions to check it out.

1. Since we don't know exactly what **cheese** is, we can drop it into Linux and run *file cheese* to view basic file information.

2. Since the filetype is Mach-O, we know it is an executible designed for MacOS. Unfortunately, since I am using Kali, I don't have any native tools to pull it apart. Fortunately a quick Google search gives me some options. I'm using radare2 since its CLI and looks like it will be able to pull apart a smaller file like this easily.
3. Radare2 can be installed with *sudo apt install radare2*.
4. *cd* into whatever directory ***cheese*** is in. Running *r2 -A cheese* opens the file up in Radare.



5. *afl* Is the Radare command to view all discovered functions in a bianary. No obvious flags here.

```
[0x100000f10]> afl
0x100000f6e    1      6 sym.imp.malloc
0x100000f10    1     93 main
0x100000c80    1    113 sym._createQueue
0x100000d50    4    131 sym._enqueue
0x100000d00    1     33 sym._isFull
0x100000d30    1     26 sym._isEmpty
0x100000de0    4    129 sym._dequeue
0x100000e70    4     72 sym._front
0x100000ec0    4     73 sym._rear
```

6. *pdf* is the command to **p**rint **d**isassembly of the **f**unction. No flags hidden here either.

```
[0x100000f10]> pdf
            ;-- entry0:
            ;-- _main:
            ;-- func.100000f10:
            ;-- rip:
┌ 93: int main (int argc, char **argv, char **envp);
│ afv: vars(2:sp[0xc..0x18])
│           0x100000f10      55              push rbp
│           0x100000f11      4889e5          mov rbp, rsp
│           0x100000f14      4883ec10        sub rsp, 0x10
│           0x100000f18      bfe8030000      mov edi, 0x3e8              ; 1000 ; int64_t arg1
│           0x100000f1d      c745fc0000..    mov dword [var_4h], 0
│           0x100000f24      e857fdffff      call sym._createQueue
│           0x100000f29      be0a000000      mov esi, 0xa                ; int64_t arg2
│           0x100000f2e      488945f0        mov qword [var_10h], rax
│           0x100000f32      488b7df0        mov rdi, qword [var_10h]    ; int64_t arg1
│           0x100000f36      e815feffff      call sym._enqueue
│           0x100000f3b      be14000000      mov esi, 0x14               ; 20 ; int64_t arg2
│           0x100000f40      488b7df0        mov rdi, qword [var_10h]    ; int64_t arg1
│           0x100000f44      e807feffff      call sym._enqueue
│           0x100000f49      be1e000000      mov esi, 0x1e               ; 30 ; int64_t arg2
│           0x100000f4e      488b7df0        mov rdi, qword [var_10h]    ; int64_t arg1
│           0x100000f52      e8f9fdffff      call sym._enqueue
│           0x100000f57      be28000000      mov esi, 0x28               ; '(' ; 40 ; int64_t arg2
│           0x100000f5c      488b7df0        mov rdi, qword [var_10h]    ; int64_t arg1
│           0x100000f60      e8ebfdffff      call sym._enqueue
│           0x100000f65      31c0            xor eax, eax
│           0x100000f67      4883c410        add rsp, 0x10
│           0x100000f6b      5d              pop rbp
└           0x100000f6c      c3              ret
```

7. *iz* lists all embedded strings in the binary.

```
[0x100000f10]> iz
[Strings]
nth paddr      vaddr       len size section            type  string
―――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――――
0   0x00000f8e 0x100000f8e 27  28   3.__TEXT.__cstring ascii the_more_i_c_the_less_i_see
```

8. ***the_more_i_c_the_less_i_see*** is our flag.