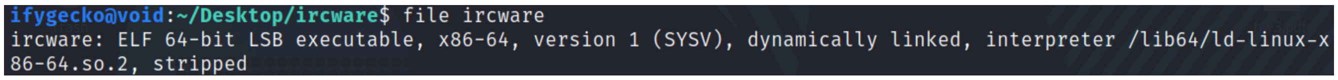
Challenge: ircware

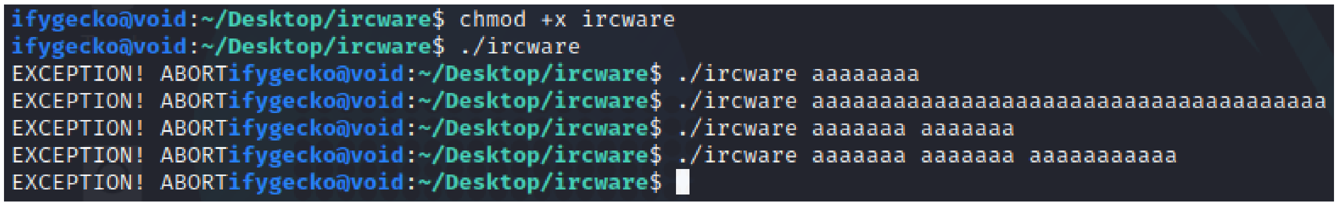
Solver(s): ifyGecko

Upon downloading the challenge file I ran the common linux tool ‘file’ to gather information about what kind of file I was working with.



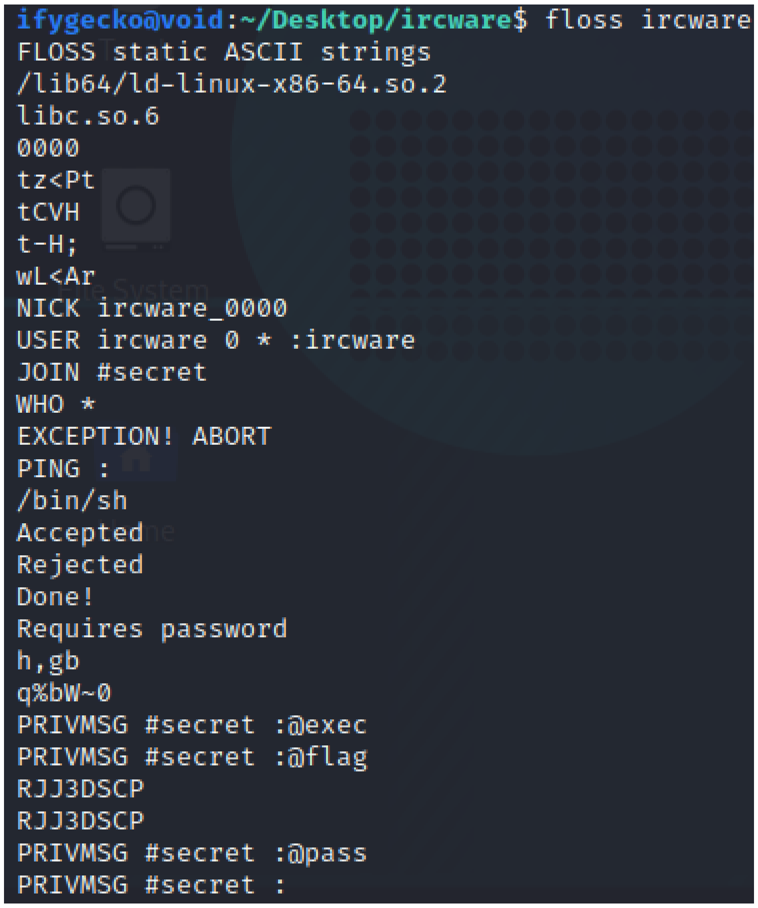
```
ifygecko@void:~/Desktop/ircware$ file ircware
ircware: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x
86-64.so.2, stripped
```

This is a stripped and dynamically linked x86_64 ELF binary, so I should have no problems running it on my machine. For curiosities sake I decided to run the program with various input to see how it would respond.



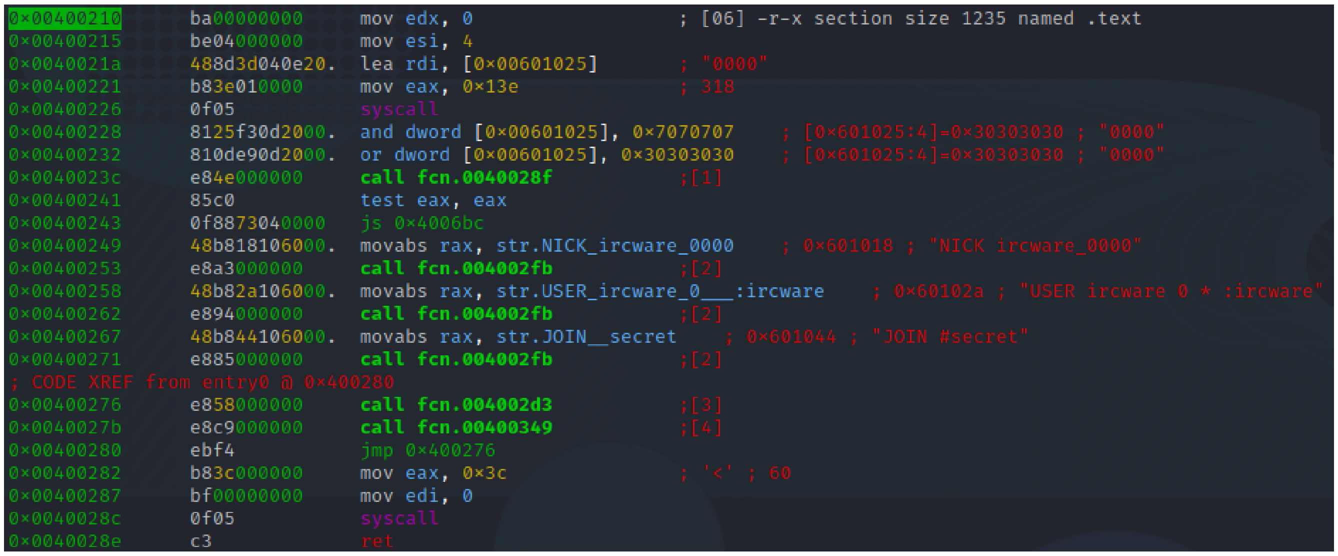
```
ifygecko@void:~/Desktop/ircware$ chmod +x ircware
ifygecko@void:~/Desktop/ircware$ ./ircware
EXCEPTION! ABORTifygecko@void:~/Desktop/ircware$ ./ircware aaaaaaaa
EXCEPTION! ABORTifygecko@void:~/Desktop/ircware$ ./ircware aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
EXCEPTION! ABORTifygecko@void:~/Desktop/ircware$ ./ircware aaaaaaa aaaaaaa
EXCEPTION! ABORTifygecko@void:~/Desktop/ircware$ ./ircware aaaaaaa aaaaaaa aaaaaaaaaaa
EXCEPTION! ABORTifygecko@void:~/Desktop/ircware$
```

This surprised me, I was not expecting the binary to continue to respond this way, but analysis must continue so my next step was to use FireEye’s ‘floss’ tool.



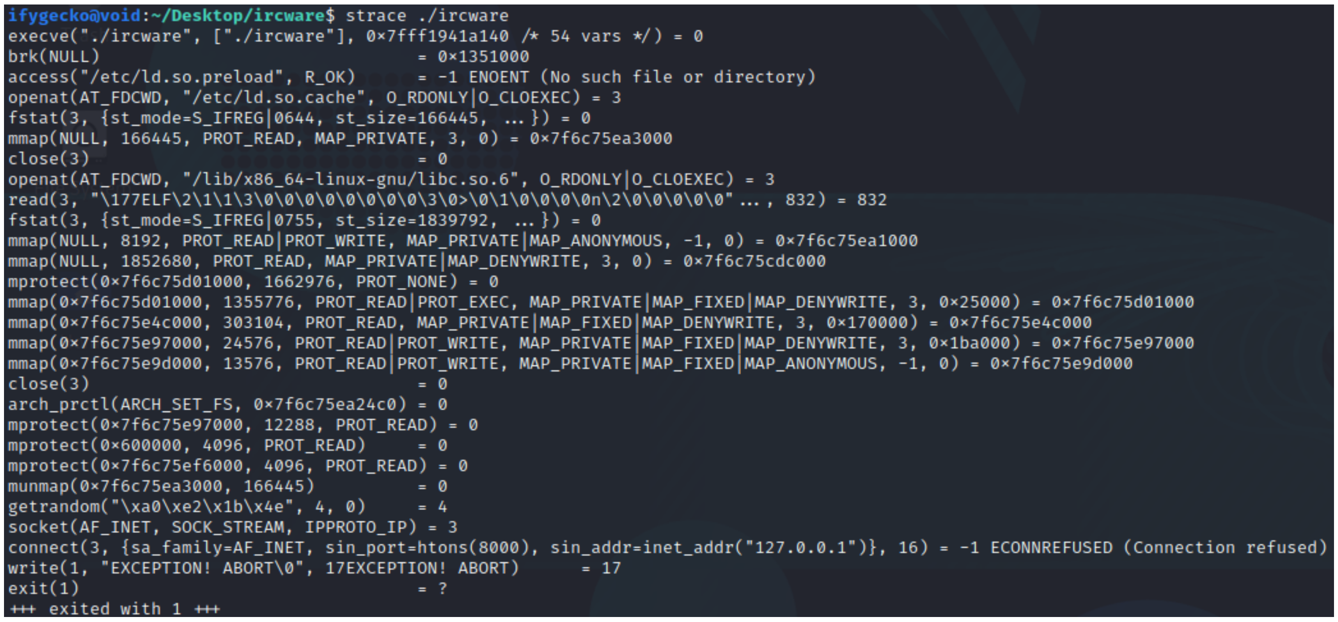
```
ifygecko@void:~/Desktop/ircware$ floss ircware
FLOSS static ASCII strings
/lib64/ld-linux-x86-64.so.2
libc.so.6
0000
tz<Pt
tCVH
t-H;
wL<Ar
NICK ircware_0000
USER ircware 0 * :ircware
JOIN #secret
WHO *
EXCEPTION! ABORT
PING :
/bin/sh
Accepted
Rejected
Done!
Requires password
h,gb
q%bW~0
PRIVMSG #secret :@exec
PRIVMSG #secret :@flag
RJJ3DSCP
RJJ3DSCP
PRIVMSG #secret :@pass
PRIVMSG #secret :
```

This gave some rather useful information such as strings that looked like commands with the notion of a ‘password’ along with an interesting string ‘RJJ3DSCP’. From here I proceeded to open the binary with radare2 and have a quick look at the program at the assembly level.



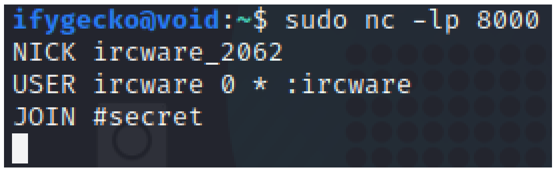
```
0x00400210      ba00000000      mov edx, 0                      ; [06] -r-x section size 1235 named .text
0x00400215      be04000000      mov esi, 4
0x0040021a      488d3d040e20.   lea rdi, [0x00601025]           ; "0000"
0x00400221      b83e010000      mov eax, 0x13e                  ; 318
0x00400226      0f05            syscall
0x00400228      8125f30d2000.   and dword [0x00601025], 0x7070707    ; [0x601025:4]=0x30303030 ; "0000"
0x00400232      810de90d2000.   or dword [0x00601025], 0x30303030    ; [0x601025:4]=0x30303030 ; "0000"
0x0040023c      e84e000000      call fcn.0040028f               ;[1]
0x00400241      85c0            test eax, eax
0x00400243      0f8873040000    js 0x4006bc
0x00400249      48b818106000.   movabs rax, str.NICK_ircware_0000    ; 0x601018 ; "NICK ircware_0000"
0x00400253      e8a3000000      call fcn.004002fb               ;[2]
0x00400258      48b82a106000.   movabs rax, str.USER_ircware_0___:ircware    ; 0x60102a ; "USER ircware 0 * :ircware"
0x00400262      e894000000      call fcn.004002fb               ;[2]
0x00400267      48b844106000.   movabs rax, str.JOIN__secret    ; 0x601044 ; "JOIN #secret"
0x00400271      e885000000      call fcn.004002fb               ;[2]
; CODE XREF from entry0 @ 0x400280
0x00400276      e858000000      call fcn.004002d3               ;[3]
0x0040027b      e8c9000000      call fcn.00400349               ;[4]
0x00400280      ebf4            jmp 0x400276
0x00400282      b83c000000      mov eax, 0x3c                   ; '<' ; 60
0x00400287      bf00000000      mov edi, 0
0x0040028c      0f05            syscall
0x0040028e      c3              ret
```

I immediately noted many system calls being used in this binary so instead of poking around in radare2 any more I decided my next best option would be to run the binary with ‘strace’ to get a high level view of what it’s doing with all of these system calls.



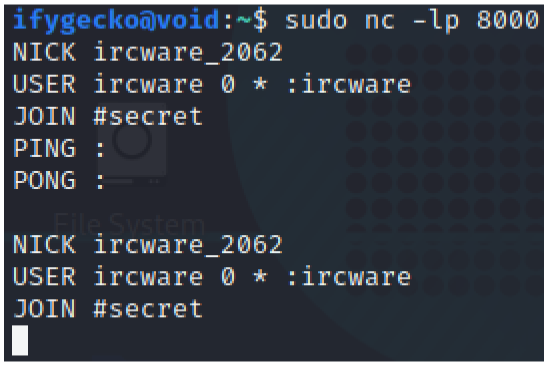
```
ifygecko@void:~/Desktop/ircware$ strace ./ircware
execve("./ircware", ["./ircware"], 0x7fff1941a140 /* 54 vars */) = 0
brk(NULL)                               = 0x1351000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=166445, ...}) = 0
mmap(NULL, 166445, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f6c75ea3000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0n\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1839792, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f6c75ea1000
mmap(NULL, 1852680, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f6c75cdc000
mprotect(0x7f6c75d01000, 1662976, PROT_NONE) = 0
mmap(0x7f6c75d01000, 1355776, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x25000) = 0x7f6c75d01000
mmap(0x7f6c75e4c000, 303104, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x170000) = 0x7f6c75e4c000
mmap(0x7f6c75e97000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1ba000) = 0x7f6c75e97000
mmap(0x7f6c75e9d000, 13576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f6c75e9d000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7f6c75ea24c0) = 0
mprotect(0x7f6c75e97000, 12288, PROT_READ) = 0
mprotect(0x600000, 4096, PROT_READ)     = 0
mprotect(0x7f6c75ef6000, 4096, PROT_READ) = 0
munmap(0x7f6c75ea3000, 166445)          = 0
getrandom("\xa0\xe2\x1b\x4e", 4, 0)     = 4
socket(AF_INET, SOCK_STREAM, IPPROTO_IP) = 3
connect(3, {sa_family=AF_INET, sin_port=htons(8000), sin_addr=inet_addr("127.0.0.1")}, 16) = -1 ECONNREFUSED (Connection refused)
write(1, "EXCEPTION! ABORT\0", 17EXCEPTION! ABORT)      = 17
exit(1)                                 = ?
+++ exited with 1 +++
```

Exactly what I was hoping for, something to make sense and move my progress forward. The program is trying to establish a connection with ‘localhost’ on port ‘8000’ and if it fails it just aborts execution. The only plausible action from here would be to start a netcat listener and run the binary again.



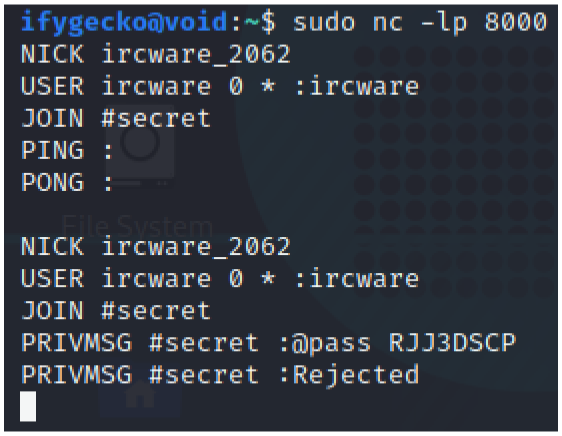
```
ifygecko@void:~$ sudo nc -lp 8000
NICK ircware_2062
USER ircware 0 * :ircware
JOIN #secret
```

Now that I saw this I immediately wondered if some of the strings I found with ‘floss’ are commands that the program is expecting to receive on the open connection. What better one to try than ‘PING :’.



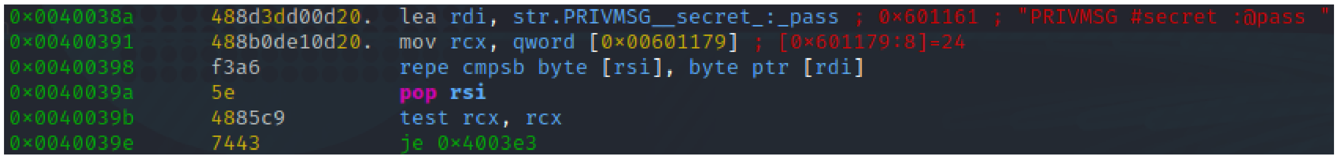
```
ifygecko@void:~$ sudo nc -lp 8000
NICK ircware_2062
USER ircware 0 * :ircware
JOIN #secret
PING :
PONG :

NICK ircware_2062
USER ircware 0 * :ircware
JOIN #secret
```

Things are becoming clearer now so the next logical one to try is ‘PRIVMSG #secret :@flag’ but sadly this did nothing. It probably does reveal the flag but only after the ‘PRIVMSG #secret :@pass’ command authenticates me via some password. Well, why not try ‘RJJ3DSCP’?



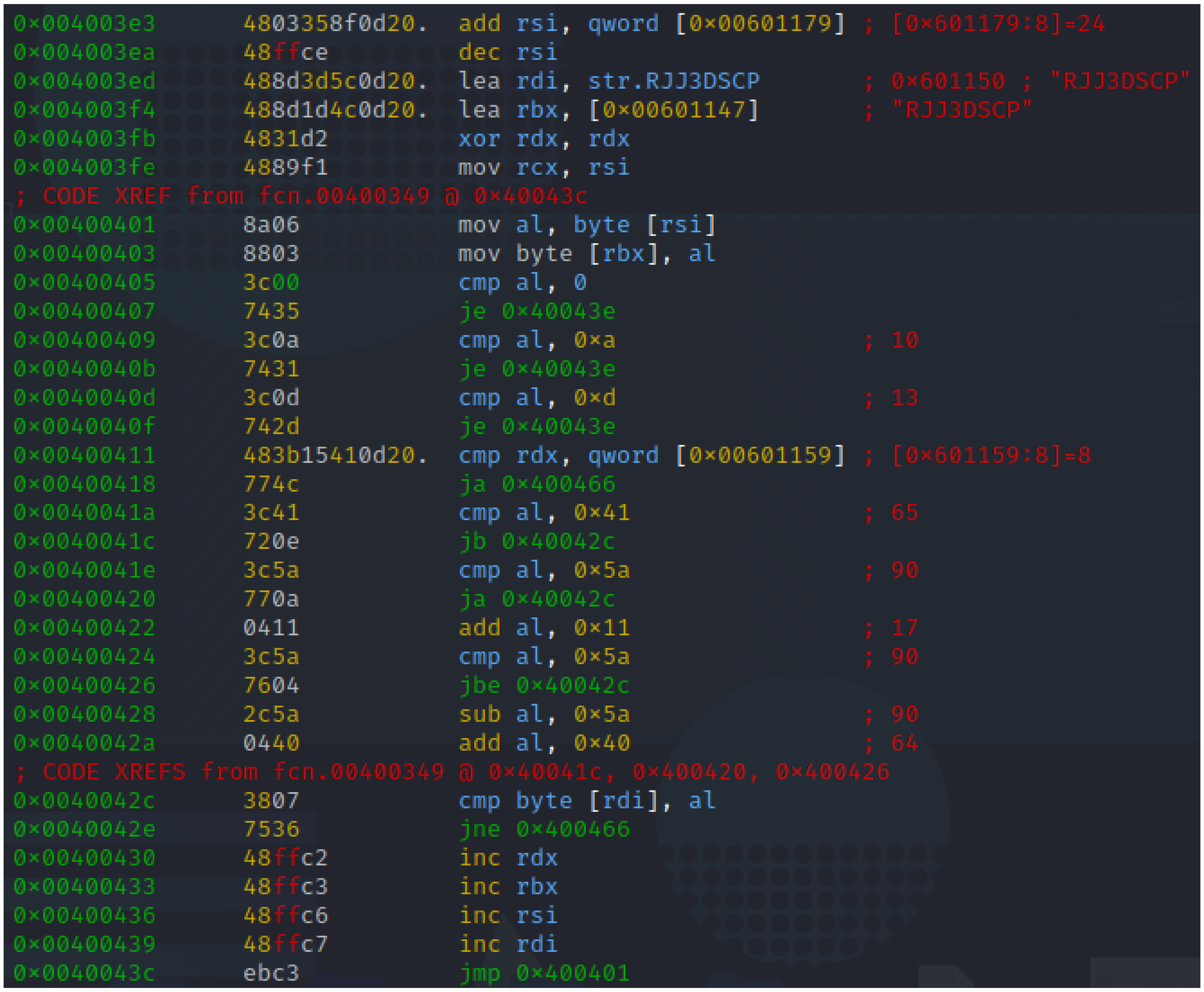
```
ifygecko@void:~$ sudo nc -lp 8000
NICK ircware_2062
USER ircware 0 * :ircware
JOIN #secret
PING :
PONG :

NICK ircware_2062
USER ircware 0 * :ircware
JOIN #secret
PRIVMSG #secret :@pass RJJ3DSCP
PRIVMSG #secret :Rejected
```

Well I guess it’s not that easy so it’s time to start looking at the authentication check and see if I can work out the correct password.



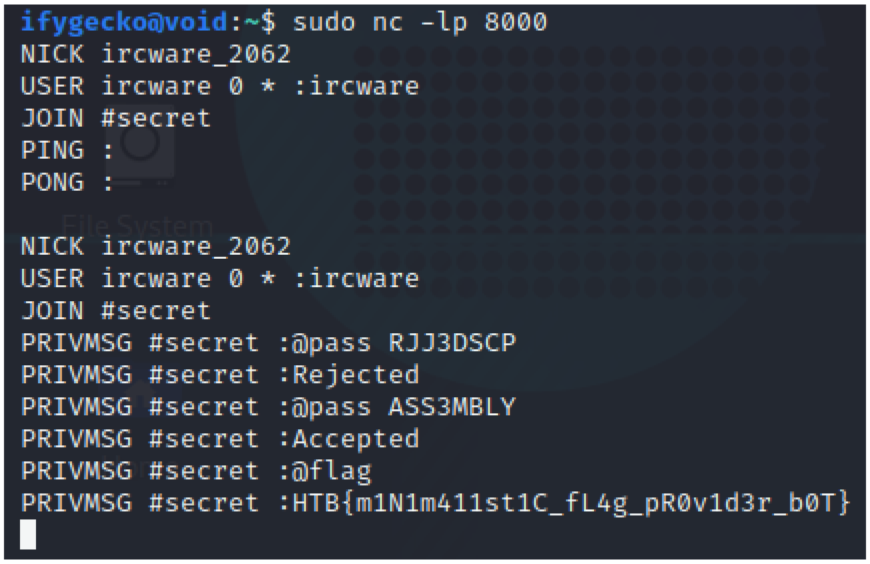
```
0x0040038a      488d3dd00d20.  lea rdi, str.PRIVMSG__secret_:_pass ; 0x601161 ; "PRIVMSG #secret :@pass "
0x00400391      488b0de10d20.  mov rcx, qword [0x00601179] ; [0x601179:8]=24
0x00400398      f3a6           repe cmpsb byte [rsi], byte ptr [rdi]
0x0040039a      5e             pop rsi
0x0040039b      4885c9         test rcx, rcx
0x0040039e      7443           je 0x4003e3
```

Somewhat quickly I found the location where the program is checking if it received the ‘PRIVMSG #secret :@pass’ string. From here I should easily be able to find where and how it is checking the provided password.



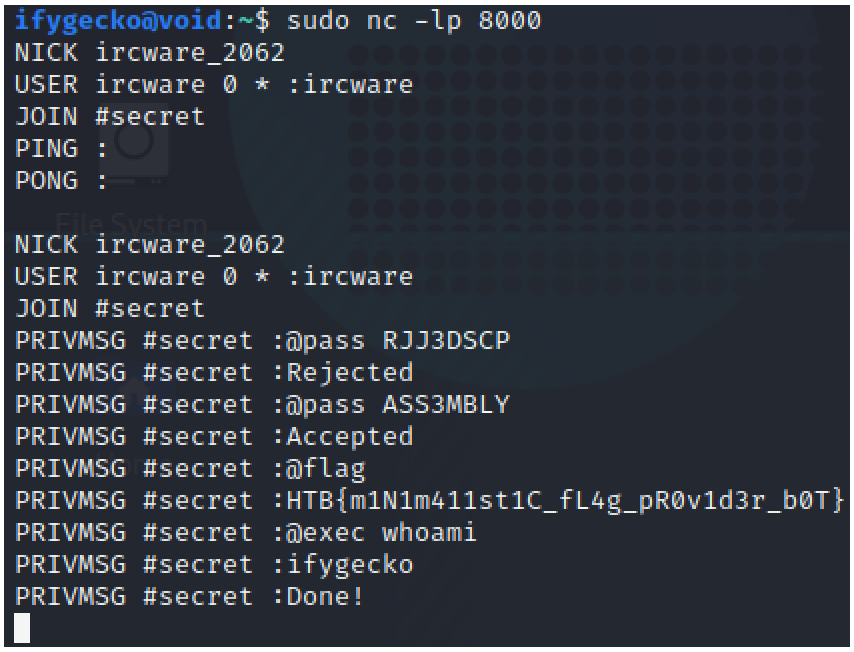
```
0×004003e3      4803358f0d20.   add rsi, qword [0×00601179] ; [0×601179:8]=24
0×004003ea      48ffce          dec rsi
0×004003ed      488d3d5c0d20.   lea rdi, str.RJJ3DSCP       ; 0×601150 ; "RJJ3DSCP"
0×004003f4      488d1d4c0d20.   lea rbx, [0×00601147]       ; "RJJ3DSCP"
0×004003fb      4831d2          xor rdx, rdx
0×004003fe      4889f1          mov rcx, rsi
; CODE XREF from fcn.00400349 @ 0×40043c
0×00400401      8a06            mov al, byte [rsi]
0×00400403      8803            mov byte [rbx], al
0×00400405      3c00            cmp al, 0
0×00400407      7435            je 0×40043e
0×00400409      3c0a            cmp al, 0×a                 ; 10
0×0040040b      7431            je 0×40043e
0×0040040d      3c0d            cmp al, 0×d                 ; 13
0×0040040f      742d            je 0×40043e
0×00400411      483b15410d20.   cmp rdx, qword [0×00601159] ; [0×601159:8]=8
0×00400418      774c            ja 0×400466
0×0040041a      3c41            cmp al, 0×41                ; 65
0×0040041c      720e            jb 0×40042c
0×0040041e      3c5a            cmp al, 0×5a                ; 90
0×00400420      770a            ja 0×40042c
0×00400422      0411            add al, 0×11                ; 17
0×00400424      3c5a            cmp al, 0×5a                ; 90
0×00400426      7604            jbe 0×40042c
0×00400428      2c5a            sub al, 0×5a                ; 90
0×0040042a      0440            add al, 0×40                ; 64
; CODE XREFS from fcn.00400349 @ 0×40041c, 0×400420, 0×400426
0×0040042c      3807            cmp byte [rdi], al
0×0040042e      7536            jne 0×400466
0×00400430      48ffc2          inc rdx
0×00400433      48ffc3          inc rbx
0×00400436      48ffc6          inc rsi
0×00400439      48ffc7          inc rdi
0×0040043c      ebc3            jmp 0×400401
```

Of course the check followed immediately after, and it is using ‘RJJ3DSCP’ for the check. After a couple of minutes of reviewing the assembly code I knew that the password provided by the user would be transformed by the sequence of instruction and each character would be compared against each in the string ‘RJJ3DSCP’. With this being the case all I would need to do is work from ‘RJJ3DSCP’ to the correct password with the inverse operation used to transform the correct password into ‘RJJ3DSCP’. To explain I will demonstrate with the first letter ‘R’ which is hex ‘0x52’. Starting at the top and checking each cmp we reach “add al, 0x11’ but in our case we are actually working backwards to we perform 0x52 – 0x11 = 0x41 which passes the “cmp al, 0x5a” jump below so our first character is 0x41 or ‘A’. Following this process, I got the password ‘ASS3MBLY’.



```
ifygecko@void:~$ sudo nc -lp 8000
NICK ircware_2062
USER ircware 0 * :ircware
JOIN #secret
PING :
PONG :

NICK ircware_2062
USER ircware 0 * :ircware
JOIN #secret
PRIVMSG #secret :@pass RJJ3DSCP
PRIVMSG #secret :Rejected
PRIVMSG #secret :@pass ASS3MBLY
PRIVMSG #secret :Accepted
PRIVMSG #secret :@flag
PRIVMSG #secret :HTB{m1N1m411st1C_fL4g_pR0v1d3r_b0T}
```

Score! Even though the flag was found I did not stop here. I remember seeing the strings "PRIVMSG #secret :@exec" and "/bin/sh" so I wanted to know if it could actually run shell commands.



```
ifygecko@void:~$ sudo nc -lp 8000
NICK ircware_2062
USER ircware 0 * :ircware
JOIN #secret
PING :
PONG :

NICK ircware_2062
USER ircware 0 * :ircware
JOIN #secret
PRIVMSG #secret :@pass RJJ3DSCP
PRIVMSG #secret :Rejected
PRIVMSG #secret :@pass ASS3MBLY
PRIVMSG #secret :Accepted
PRIVMSG #secret :@flag
PRIVMSG #secret :HTB{m1N1m411st1C_fL4g_pR0v1d3r_b0T}
PRIVMSG #secret :@exec whoami
PRIVMSG #secret :ifygecko
PRIVMSG #secret :Done!
```

As expected it does actually run shell commands!