Mission Title

The Test

Historical Context

Following his capture by Claire and Tarain, Ethan opts to work with Recon to gather intel on SHAX. However, as a preliminary step, Recon subjects Ethan to an assessment to gauge his hacking prowess.

Overview of Technical Strategy

Ethan faces a challenge on a computer system. His objective is to acquire the highest level of access possible on the provided computer.

Brief Mission Overview

Greetings, Ethan, and welcome to the Recon Evaluation. We will grant you access to a computer system where your hacking abilities will be put to the test. To proceed with our team, you must successfully complete this challenge. Are you prepared to take control of the assigned system? Best of luck!

Detailed Mission Brief

After his arrest by Claire and Tarain, Ethan agrees to aid Recon in uncovering details about SHAX. As an initial step, Recon requires Ethan to undergo a test to verify his hacking skills.

Tools

- User: thetest
- Password: p4ssw0rdT3st

Questions

What is the correct input parameter?

- R3c0nT3sT!

What is the correct running input parameter?

- C0nc4tX0R!

What is the correct hostname string check?

- R3c0n

Items

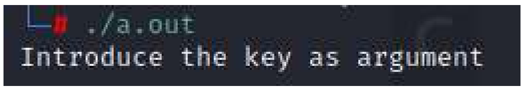
1. Search hardcoded information with strings.
2. The first check operation is a XOR hardcoded, search in strncmp
3. Copy binary to local machine and change the hostname to the necessary

Categories

- Reversing

Write Up

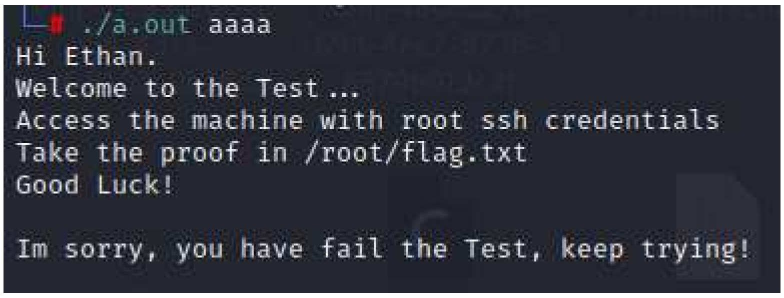
Connect to the computer using SSH credentials: Username: thetest Password: p4ssw0rdT3st
Within the /home/thetest directory, there's a binary file. Running this binary without any arguments yields:



```
└─# ./a.out
Introduce the key as argument
```

Figure 1

Execute with a argument will result in:



```
└─# ./a.out aaaa
Hi Ethan.
Welcome to the Test ...
Access the machine with root ssh credentials
Take the proof in /root/flag.txt
Good Luck!

Im sorry, you have fail the Test, keep trying!
```

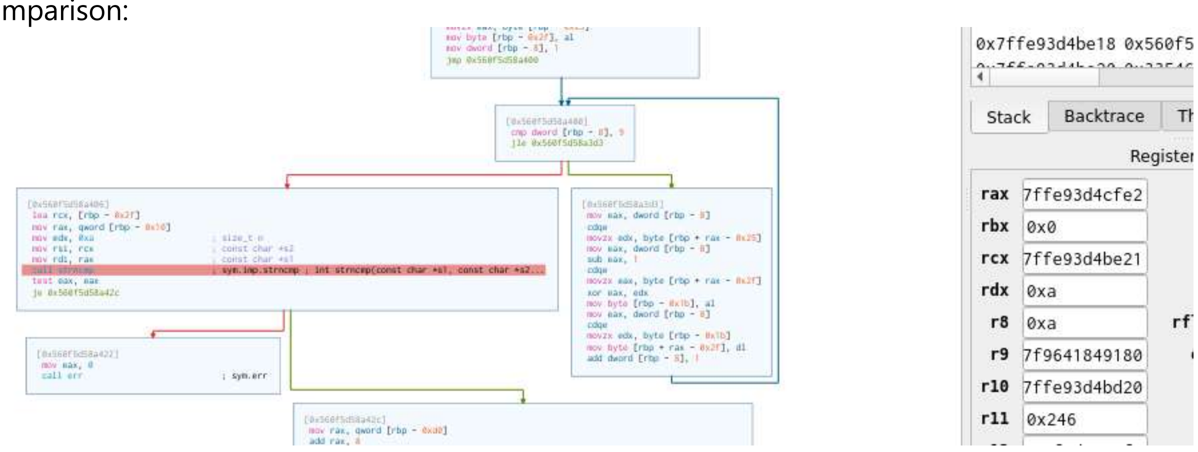
Figure 2

Running the binary with an argument provides:

This binary requires reverse engineering to understand its functionality.

Upon examining the code, a quick analysis reveals two major comparisons, the initial one involving the argument provided:

First comparison:



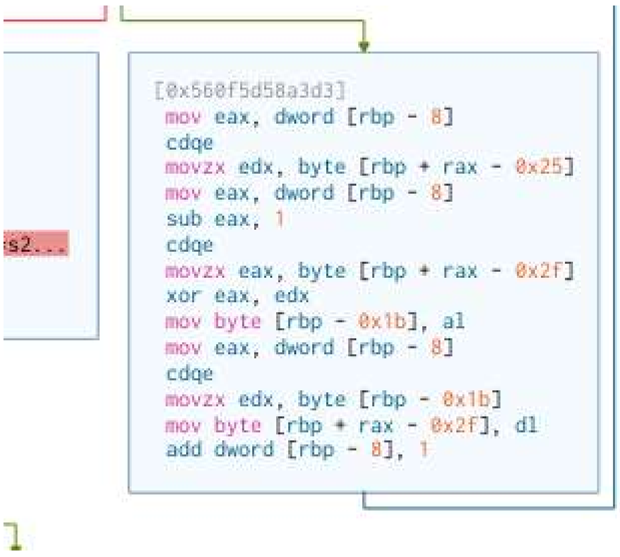
```
mov byte [rbp - 0x2f], al
mov dword [rbp - 8], 1
jmp 0x560f5d58a400

[0x560f5d58a400]
cmp dword [rbp - 8], 9
jle 0x560f5d58a3d3

[0x560f5d58a406]
lea rcx, [rbp - 0x2f]
mov rax, qword [rbp - 0x10]
mov edx, 0xa                 ; size_t n
mov rsi, rcx                 ; const char *s2
mov rdi, rax                 ; const char *s1
call strncmp                 ; sym.imp.strncmp ; int strncmp(const char *s1, const char *s2...
test eax, eax
je 0x560f5d58a42c

[0x560f5d58a3d3]
mov eax, dword [rbp - 8]
cdqe
movzx edx, byte [rbp + rax - 0x25]
mov eax, dword [rbp - 8]
sub eax, 1
cdqe
movzx eax, byte [rbp + rax - 0x2f]
xor eax, edx
mov byte [rbp - 0x1b], al
mov eax, dword [rbp - 8]
cdqe
movzx edx, byte [rbp - 0x1b]
mov byte [rbp + rax - 0x2f], dl
add dword [rbp - 8], 1

[0x560f5d58a422]
mov eax, 0
call err                     ; sym.err

[0x560f5d58a42c]
mov rax, qword [rbp - 0xd0]
add rax, 8
```

```
0x7ffe93d4be18 0x560f5
```

```
rax  7ffe93d4cfe2
rbx  0x0
rcx  7ffe93d4be21
rdx  0xa
 r8  0xa
 r9  7f9641849180
r10  7ffe93d4bd20
r11  0x246
```

Figure 3

The calculation of RCX is based on a graph derived from an earlier operation.



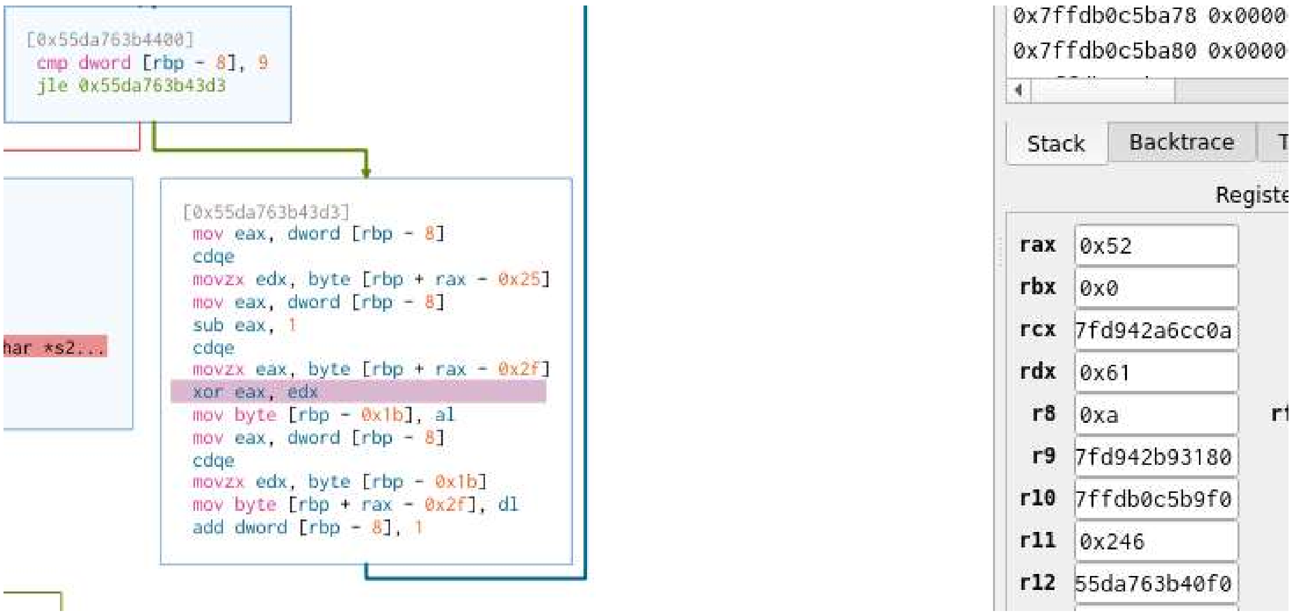
```
[0x560f5d58a3d3]
 mov eax, dword [rbp - 8]
 cdqe
 movzx edx, byte [rbp + rax - 0x25]
 mov eax, dword [rbp - 8]
 sub eax, 1
 cdqe
 movzx eax, byte [rbp + rax - 0x2f]
 xor eax, edx
 mov byte [rbp - 0x1b], al
 mov eax, dword [rbp - 8]
 cdqe
 movzx edx, byte [rbp - 0x1b]
 mov byte [rbp + rax - 0x2f], dl
 add dword [rbp - 8], 1
```

Figure 4

The operation involves an XOR process with hardcoded hexadecimal values that are shifted right by one position. This means the operation performed is as follows:

\[0x52 \, \text{XOR} \, 0x61 = 0x33\]

This XOR calculation results from applying the XOR bitwise operator to the given hexadecimal values, leading to the specified outcome.



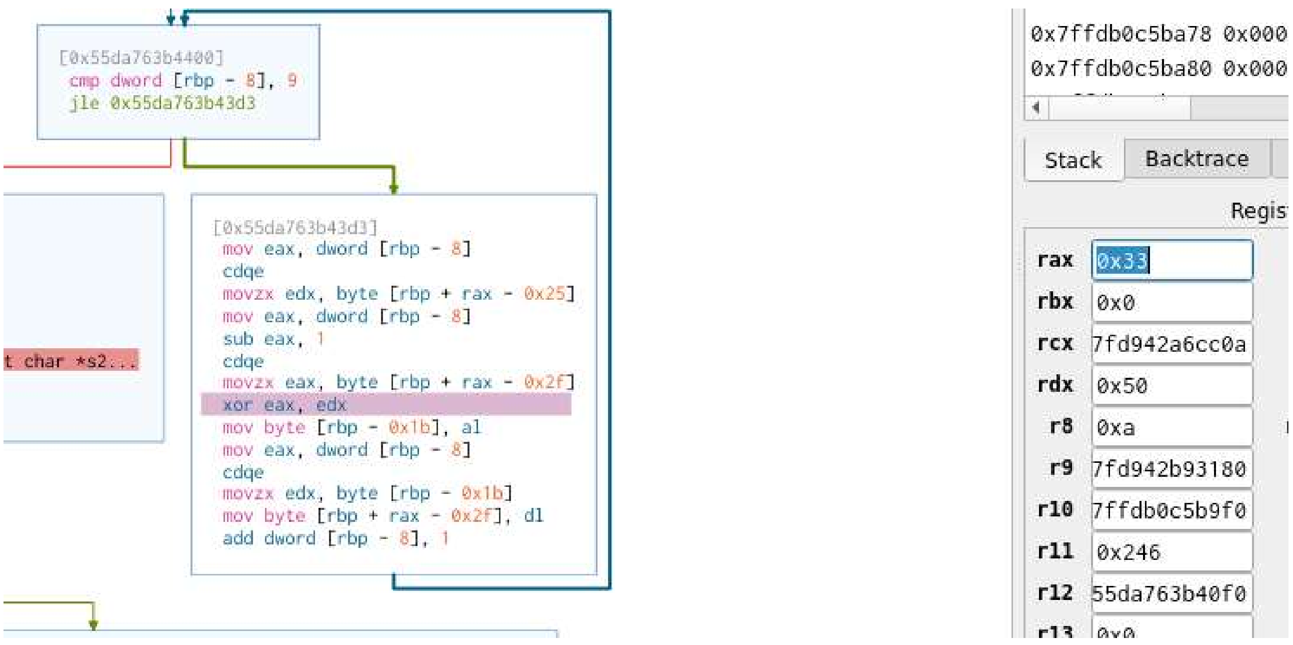
```
[0x55da763b4400]
 cmp dword [rbp - 8], 9
 jle 0x55da763b43d3
```

```
[0x55da763b43d3]
 mov eax, dword [rbp - 8]
 cdqe
 movzx edx, byte [rbp + rax - 0x25]
 mov eax, dword [rbp - 8]
 sub eax, 1
 cdqe
 movzx eax, byte [rbp + rax - 0x2f]
 xor eax, edx
 mov byte [rbp - 0x1b], al
 mov eax, dword [rbp - 8]
 cdqe
 movzx edx, byte [rbp - 0x1b]
 mov byte [rbp + rax - 0x2f], dl
 add dword [rbp - 8], 1
```

```
0x7ffdb0c5ba78 0x0000
0x7ffdb0c5ba80 0x0000
```

```
rax  0x52
rbx  0x0
rcx  7fd942a6cc0a
rdx  0x61
 r8  0xa
 r9  7fd942b93180
r10  7ffdb0c5b9f0
r11  0x246
r12  55da763b40f0
```

Figure 5



```
[0x55da763b4400]
cmp dword [rbp - 8], 9
jle 0x55da763b43d3
```

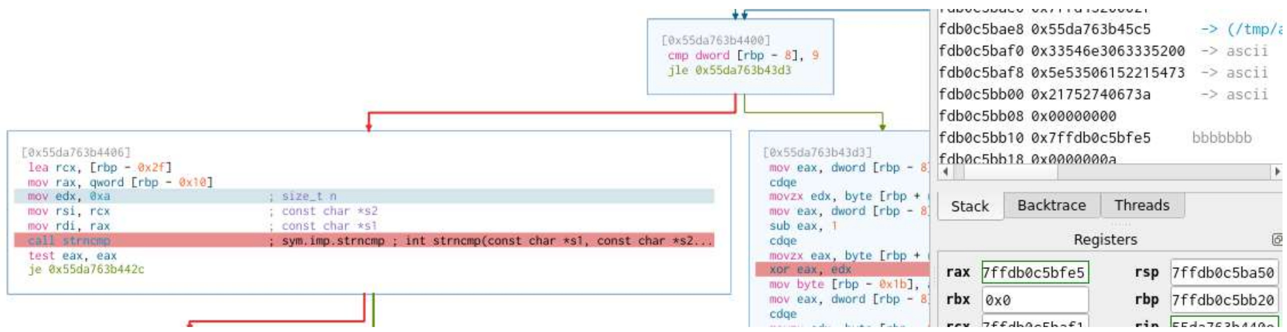
```
t char *s2...
```

```
[0x55da763b43d3]
mov eax, dword [rbp - 8]
cdqe
movzx edx, byte [rbp + rax - 0x25]
mov eax, dword [rbp - 8]
sub eax, 1
cdqe
movzx eax, byte [rbp + rax - 0x2f]
xor eax, edx
mov byte [rbp - 0x1b], al
mov eax, dword [rbp - 8]
cdqe
movzx edx, byte [rbp - 0x1b]
mov byte [rbp + rax - 0x2f], dl
add dword [rbp - 8], 1
```

```
0x7ffdb0c5ba78 0x000
0x7ffdb0c5ba80 0x000
```

```
rax  0x33
rbx  0x0
rcx  7fd942a6cc0a
rdx  0x50
 r8  0xa
 r9  7fd942b93180
r10  7ffdb0c5b9f0
r11  0x246
r12  55da763b40f0
r13  0x0
```

Figure 6



```
[0x55da763b4400]
cmp dword [rbp - 8], 9
jle 0x55da763b43d3
```

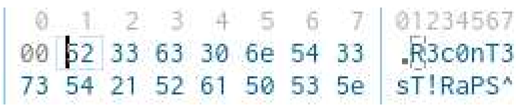
```
[0x55da763b4406]
lea rcx, [rbp - 0x2f]
mov rax, qword [rbp - 0x10]
mov edx, 0xa                 ; size_t n
mov rsi, rcx                 ; const char *s2
mov rdi, rax                 ; const char *s1
call strncmp                 ; sym.imp.strncmp ; int strncmp(const char *s1, const char *s2...
test eax, eax
je 0x55da763b442c
```

```
[0x55da763b43d3]
mov eax, dword [rbp - 8
cdqe
movzx edx, byte [rbp +
mov eax, dword [rbp - 8
sub eax, 1
cdqe
movzx eax, byte [rbp +
xor eax, edx
mov byte [rbp - 0x1b],
mov eax, dword [rbp - 8
cdqe
```

```
fdb0c5bae8 0x55da763b45c5       -> (/tmp/a
fdb0c5baf0 0x33546e3063335200   -> ascii
fdb0c5baf8 0x5e53506152215473   -> ascii
fdb0c5bb00 0x21752740673a       -> ascii
fdb0c5bb08 0x00000000
fdb0c5bb10 0x7ffdb0c5bfe5       bbbbbbb
fdb0c5bb18 0x0000000a
```

```
rax  7ffdb0c5bfe5     rsp  7ffdb0c5ba50
rbx  0x0              rbp  7ffdb0c5bb20
```

Figure 7



```
 0  1  2  3  4  5  6  7   01234567
00 52 33 63 30 6e 54 33   .R3c0nT3
73 54 21 52 61 50 53 5e   sT!RaPS^
```

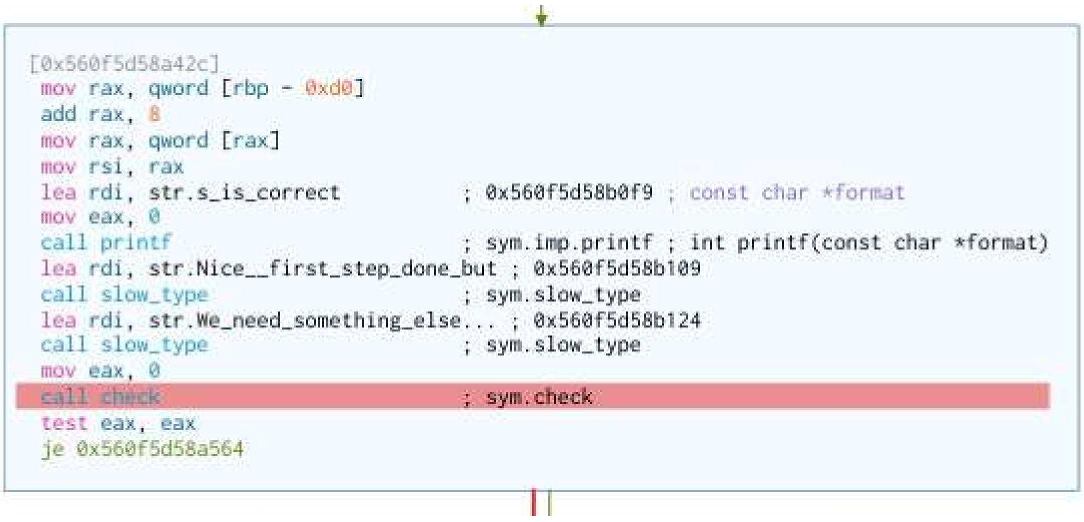
Figure 8

The complete string "3c0nT3sT!" is stored in the stack and iterated over 9 times.

Proceeding further into the `strncmp` function call and inspecting the hexdump, the complete value becomes evident.

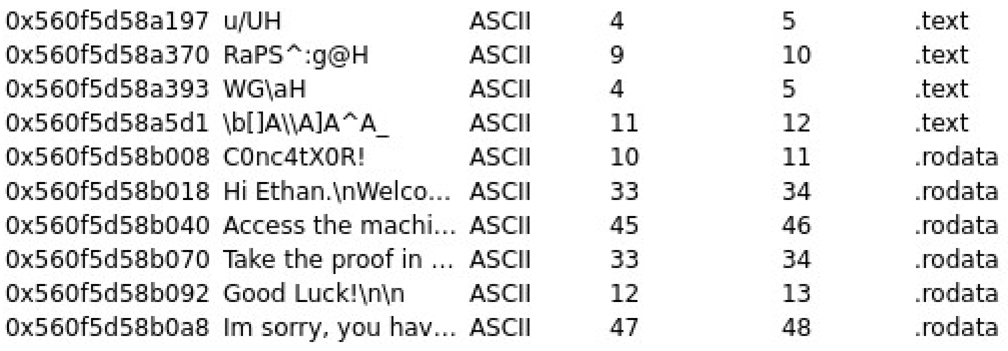
Within this call, RAX represents the passed argument, while RCX signifies the calculated value. Therefore, the argument must be "R3c0nT3sT!"

Following this initial verification with the argument, there's an additional validation step during runtime involving a hardcoded password and the machine hostname. The hardcoded password, "C0nc4tX0R!", is checked using the `check()` function.



```
[0x560f5d58a42c]
 mov rax, qword [rbp - 0xd0]
 add rax, 8
 mov rax, qword [rax]
 mov rsi, rax
 lea rdi, str.s_is_correct          ; 0x560f5d58b0f9 ; const char *format
 mov eax, 0
 call printf                        ; sym.imp.printf ; int printf(const char *format)
 lea rdi, str.Nice__first_step_done_but ; 0x560f5d58b109
 call slow_type                     ; sym.slow_type
 lea rdi, str.We_need_something_else... ; 0x560f5d58b124
 call slow_type                     ; sym.slow_type
 mov eax, 0
 call check                         ; sym.check
 test eax, eax
 je 0x560f5d58a564
```

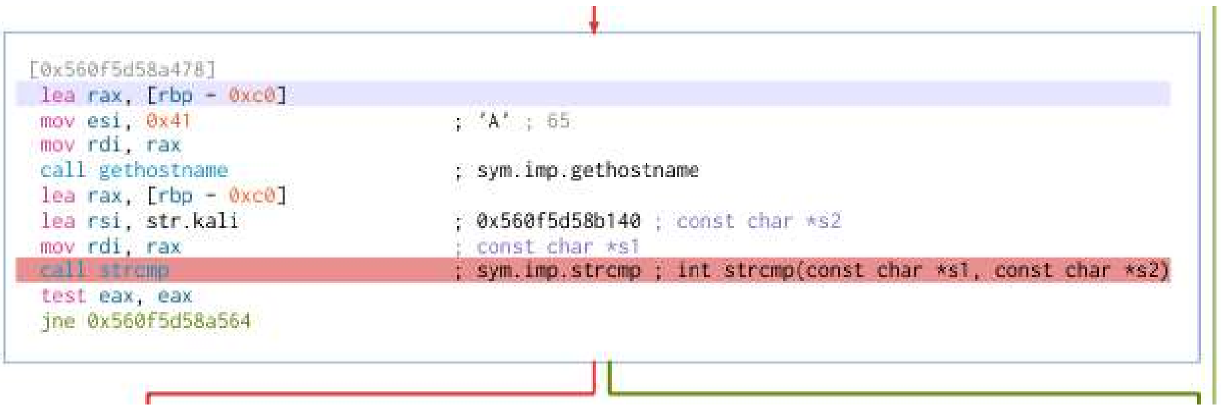
Figure 9



```
0x560f5d58a197 u/UH              ASCII  4   5   .text
0x560f5d58a370 RaPS^:g@H         ASCII  9   10  .text
0x560f5d58a393 WG\aH             ASCII  4   5   .text
0x560f5d58a5d1 \b[]A\\A]A^A_     ASCII  11  12  .text
0x560f5d58b008 C0nc4tX0R!        ASCII  10  11  .rodata
0x560f5d58b018 Hi Ethan.\nWelco... ASCII 33  34  .rodata
0x560f5d58b040 Access the machi... ASCII 45  46  .rodata
0x560f5d58b070 Take the proof in ... ASCII 33 34 .rodata
0x560f5d58b092 Good Luck!\n\n    ASCII  12  13  .rodata
0x560f5d58b0a8 Im sorry, you hav... ASCII 47 48  .rodata
```

Figure 10

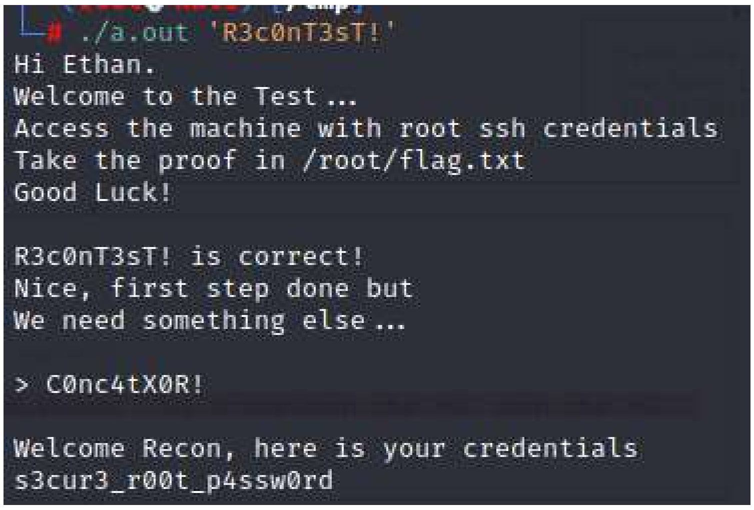
The machine hostname should be "R3c0n".



```
[0x560f5d58a478]
lea rax, [rbp - 0xc0]
mov esi, 0x41                     ; 'A' ; 65
mov rdi, rax
call gethostname                  ; sym.imp.gethostname
lea rax, [rbp - 0xc0]
lea rsi, str.kali                 ; 0x560f5d58b140 ; const char *s2
mov rdi, rax                      ; const char *s1
call strcmp                       ; sym.imp.strcmp ; int strcmp(const char *s1, const char *s2)
test eax, eax
jne 0x560f5d58a564
```

Figure 11

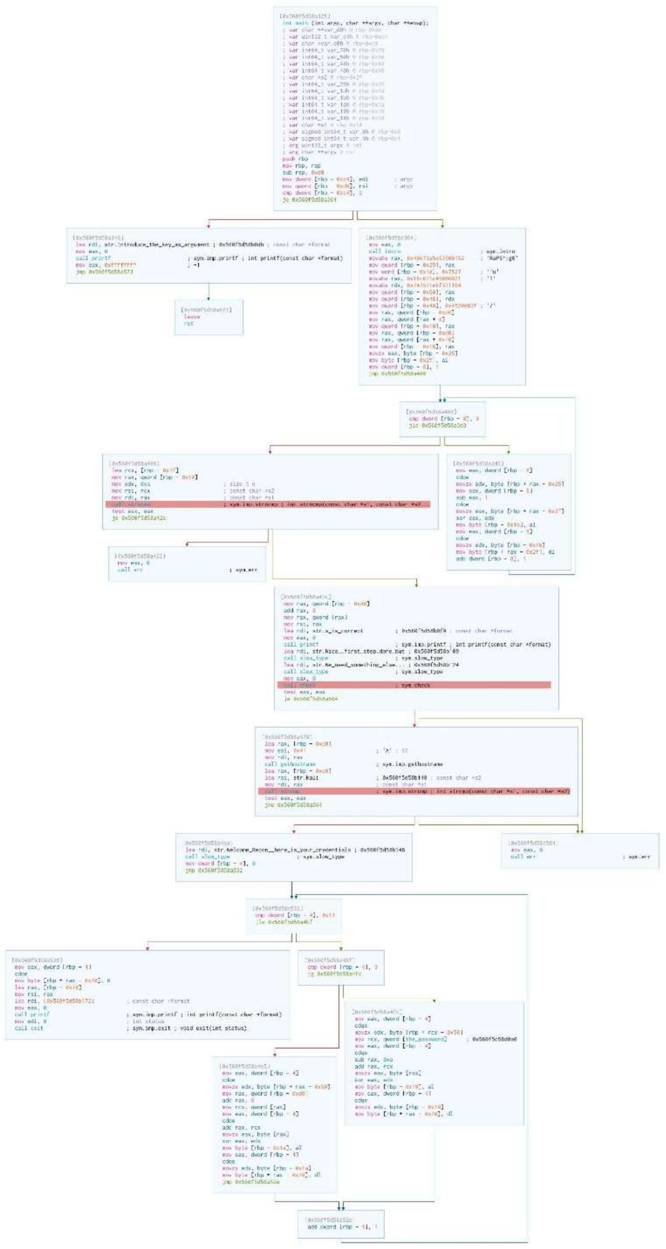
Success!



```
└─# ./a.out 'R3c0nT3sT!'
Hi Ethan.
Welcome to the Test...
Access the machine with root ssh credentials
Take the proof in /root/flag.txt
Good Luck!

R3c0nT3sT! is correct!
Nice, first step done but
We need something else...

> C0nc4tX0R!

Welcome Recon, here is your credentials
s3cur3_r00t_p4ssw0rd
```

Figure 12

Full Graph



Flag Information

flag{3nd_0f_R3v3rs1ng_T3st}