How System Call is Issued and Handled

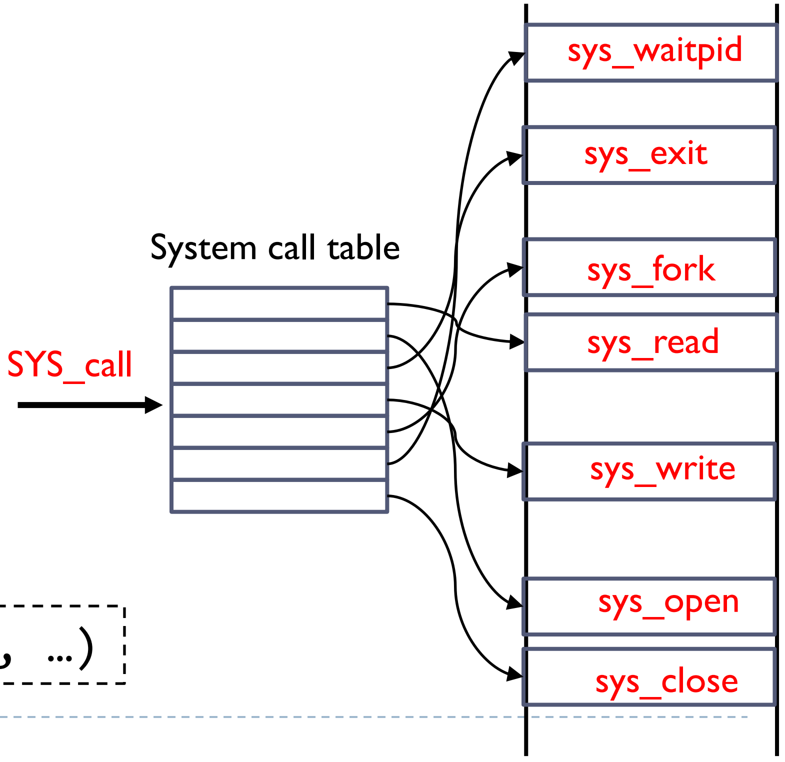
A system call is an interface that allows a user-level process to request functions or services from the kernel level.

- Process control
- File management
- Device management

How to issue a system call?

- System call table: a table of pointers in the kernel region, to different system call functions.
- A user process passes the index of the system call and parameters with the following API:

```
syscall(SYS_call, arg1, arg2, …)
```



Rootkit

Malware that obtains root privileges to compromise the computer

- Root user does not go though any security checks, and can perform any actions to the system
  - Insert and execute arbitrary malicious code in the system's code path
  - Hide its existence, e.g., malicious process, files, network sockets, from being detected.

How can the attacker gain the root privileges?

- Vulnerabilities in the software stack: buffer overflow, format string…

There are some common techniques for rootkits to compromise the systems.

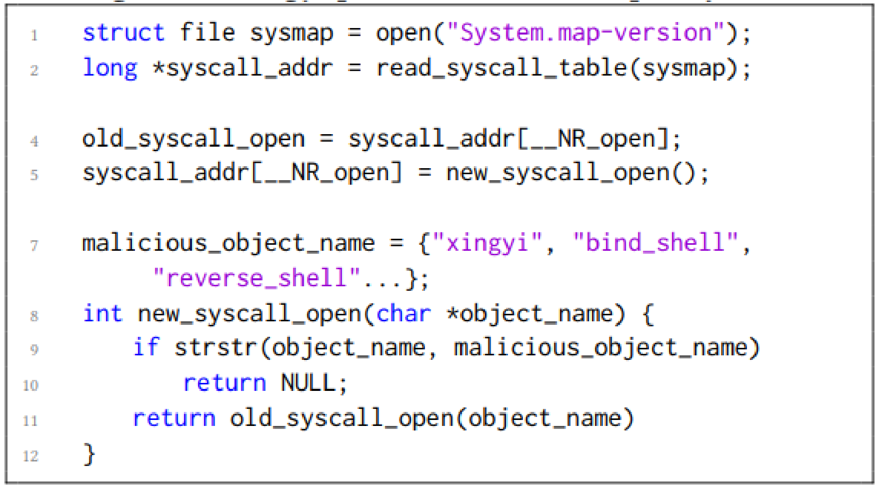
Highjack System-call Table

Rootkit changes pointers of certain entries in the system-call table.

- Other processes calling these system calls will execute the attacker's code

Example

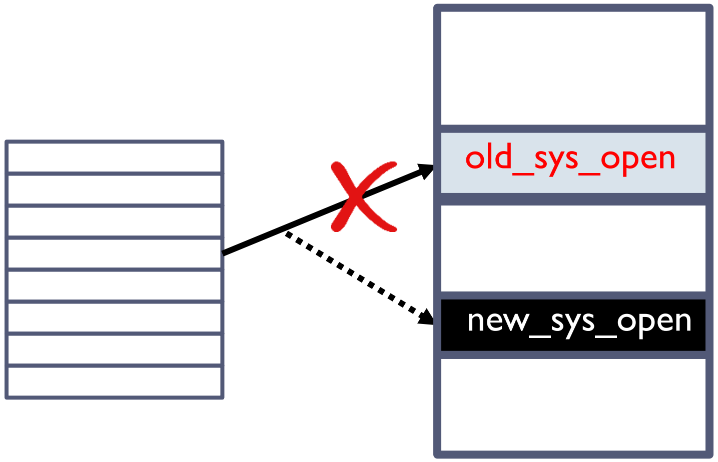
- `syscall_open` is used to display the running process (`ps` command)
- Rootkit redirects this system call to `new_syscall_open`
  - When the object to be opened matches the malicious name, return NULL to hide it
  - Otherwise, call normal `old_syscall_open`



```
struct file sysmap = open("System.map-version");
long *syscall_addr = read_syscall_table(sysmap);

old_syscall_open = syscall_addr[__NR_open];
syscall_addr[__NR_open] = new_syscall_open();

malicious_object_name = {"xingyi", "bind_shell",
      "reverse_shell"...};
int new_syscall_open(char *object_name) {
    if strstr(object_name, malicious_object_name)
        return NULL;
    return old_syscall_open(object_name)
}
```



https://sw0rdm4n.wordpress.com/2014/11/03/xingyiquan-simple-linux-kernel-rootkit-for-kernel-3-x-and-kernel-2-6-x/

Compromise System Call Functions

Rootkit can also directly change the system call function.

Example

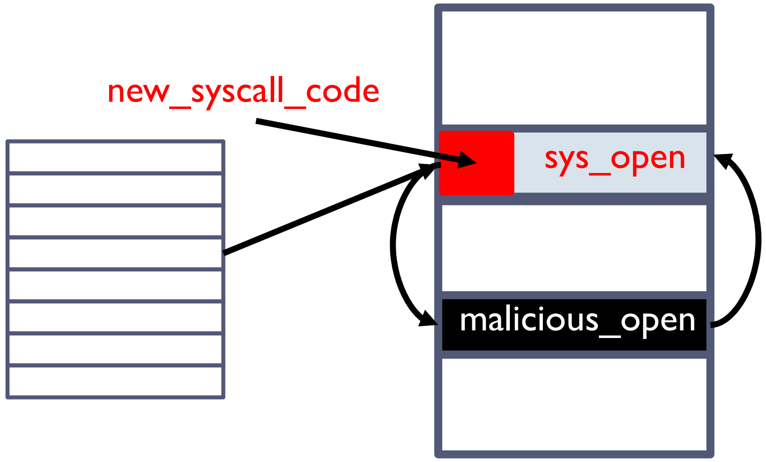
- Replace the first 7 bytes of `syscall_open` as jump to `malicious_open`.
  - This faked system call will issue malicious function, restore the original system call and then call the correct one.

```c
struct file sysmap = open("System.map-version");
long *syscall_addr = read_syscall_table(sysmap);
syscall_open = syscall_addr[__NR_open];

char old_syscall_code[7];
memncpy(old_syscall_code, syscall_open, 7);

char pt[4];
memncpy(pt, (long)malicious_open, 4)
char new_syscall_code[7] =
{"\xbd",pt[0],pt[1],pt[2],pt[3], // movl %pt, %ebp
"\xff","\xe5"};                    // jmp %ebp
memncpy(syscall_open, new_syscall_code, 7);

int malicious_open(char *object_name) {
    malicious_function();
    memncpy(syscall_open, old_syscall_code, 7);
    return syscall_open(object_name);
}
```



http://www.ouah.org/stealth-syscall.txt