Computer and network security

Final Project
Linux Rootkit

Introduction

Linux kernel is modular. It constitutes of several kernel modules, that can be easily loaded and unloaded without a restart.

Loading a kernel module is done by typing:

insmod {module_path}..

Unloading is performed by typing:

rmmod {module_name}.

To display the list of loaded kernel modules, we type:

lsmod.

In modular kernels, module are provided with much higher permissions/access rights than those provided to normal user applications. This allows us to simply implement a rootkit. A rootkit, simply put, is software that gives root access to a user or a process. Usually, kernel rootkits provide root access by anticipating a specific keyword sent from the user space to it. In this project we will create a kernel module that acts as a rootkit, and also performs the following:

- Hiding itself
- Hiding a process given its pid
- Logging Keystrokes

Overview

A kernel module requires two main functions. An entry and an exit function.

The entry function is called when the module is loaded, and the exit function is called when the module is unloaded.

To register the entry and exit functions we pass them as follows:

module_init(init);

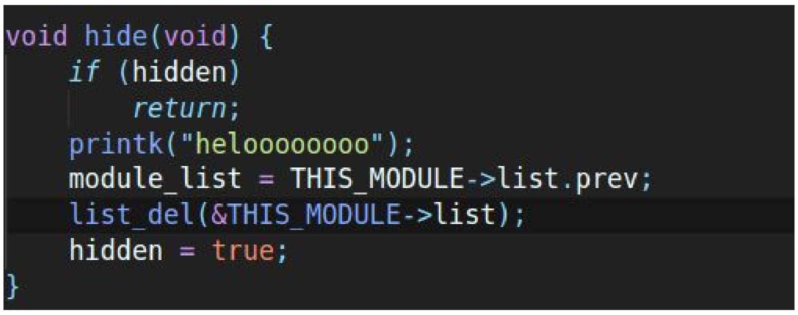
module_exit(clean_up);

In order to print commands we use printk function rather than printf, and the result is displayed by the command dmesg. Also, it should be noted that allocating memory and freeing it is done by the 'kmalloc' and the 'kfree' functions respectively, rather than 'malloc' and 'free'. To perform the functions of our rootkit when commanded by a user we have to send data to our rootkit signaling it to perform one of those functions. For a process from the user-space to communicate with another from the kernel-space we need a some medium to do so. In our project we used the 'proc' virtual filesystem for the most of such communication. The /proc directory in linux is a directory that is mainly used for providing important information about processes and files to user space applications. To be able to communicate with the kernel module, as explained, we create a directory under /proc in the init function called 'myddev'. We set it is access rights to maximum access rights (0777) this is to allow normal users to communicate with 'myddev' and utilize the keylogger. We also pass a file_operations struct pointer to our proc directory, the struct has a write attribute set with our own write function. In this write function we handle input given from the user and perform the correct rootkit function depending on the input provided. User provides input by typing the following in the terminal:
echo "command" > /proc/myddev. This is treated as writing and hence our write function will handle such command.

In the clean_up function this directory, 'myddev', is deleted.

Hiding Kernel Module

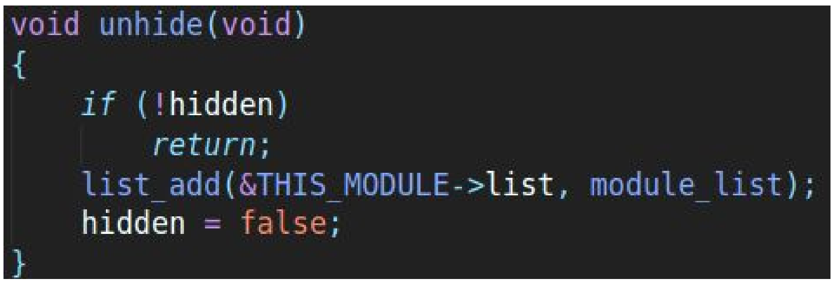
Hiding the kernel module is quite simple it is implemented as follows:



```
void hide(void) {
    if (hidden)
        return;
    printk("heloooooooo");
    module_list = THIS_MODULE->list.prev;
    list_del(&THIS_MODULE->list);
    hidden = true;
}
```

Here we save our the module’s list’s prev attribute, to be able to unhide the kernel module. Then we use the the ‘list_del’ function imported from ‘kernel/list.h’ to delete the module’s list attribute from the list of modules and hence the module will not be visible to the command ‘lsmod’. Here we set a boolean attribute ‘hidden’ to true so if the user tries to hide an already hidden process the rootkit will not try to hide it again.

Unhiding is implemented as follows:



```
void unhide(void)
{
    if (!hidden)
        return;
    list_add(&THIS_MODULE->list, module_list);
    hidden = false;
}
```

Simply here we add to the list of module the backed-up modules list, and we set the hidden variable to false for the same reason that was explained above.

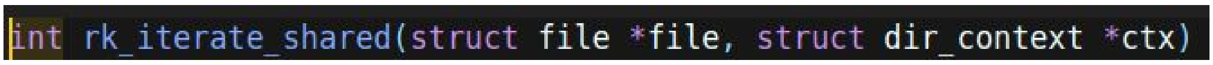
Hiding the module is done by typing the following: echo “hide” > /proc/myddev

Unhiding: echo “unhide” > /proc/myddev.

The module is hidden at first.

Hiding a process by its pid:

As explained earlier, the proc virtual file system is useful for many applications such as the ‘ps’ command which lists the current running processes. Utilizing this fact, we were able to change its file operations attribute ‘i_fop’. The file operations struct has an attribute iterate_shared, a function pointer, which is called by the ‘ps’ command. We change i_fop’s iterate_shared to another malicious one in the init function. Then we return it to normal in the clean_up function. Iterate_shared has the following signature:



```
int rk_iterate_shared(struct file *file, struct dir_context *ctx)
```

The dir_context struct is what is useful for us here. It has an actor attribute that is be default given a function called ‘filldir_t’ this function puts a process in the list of processes when called. We created our own actor function called ‘rk_filldir_t’. It checks if the current pid to be displayed in the list of processes is equivalent to a pid provided by the user to the rootkit to be hidden, if it is we return 0, which would not put it in the list of processes. The command is as follows:

‘echo hide-pid={pid} > /proc/myddev’

The root access:

The keyword defined for the root access is "rektt". This is the keyword that the attacker can type in the terminal to give himself root access.

In my_write function we check for the data typed by the user if the he types “rektt” we execute the following lines:

V(new_cred->uid) = V(new_cred->gid) = 0;

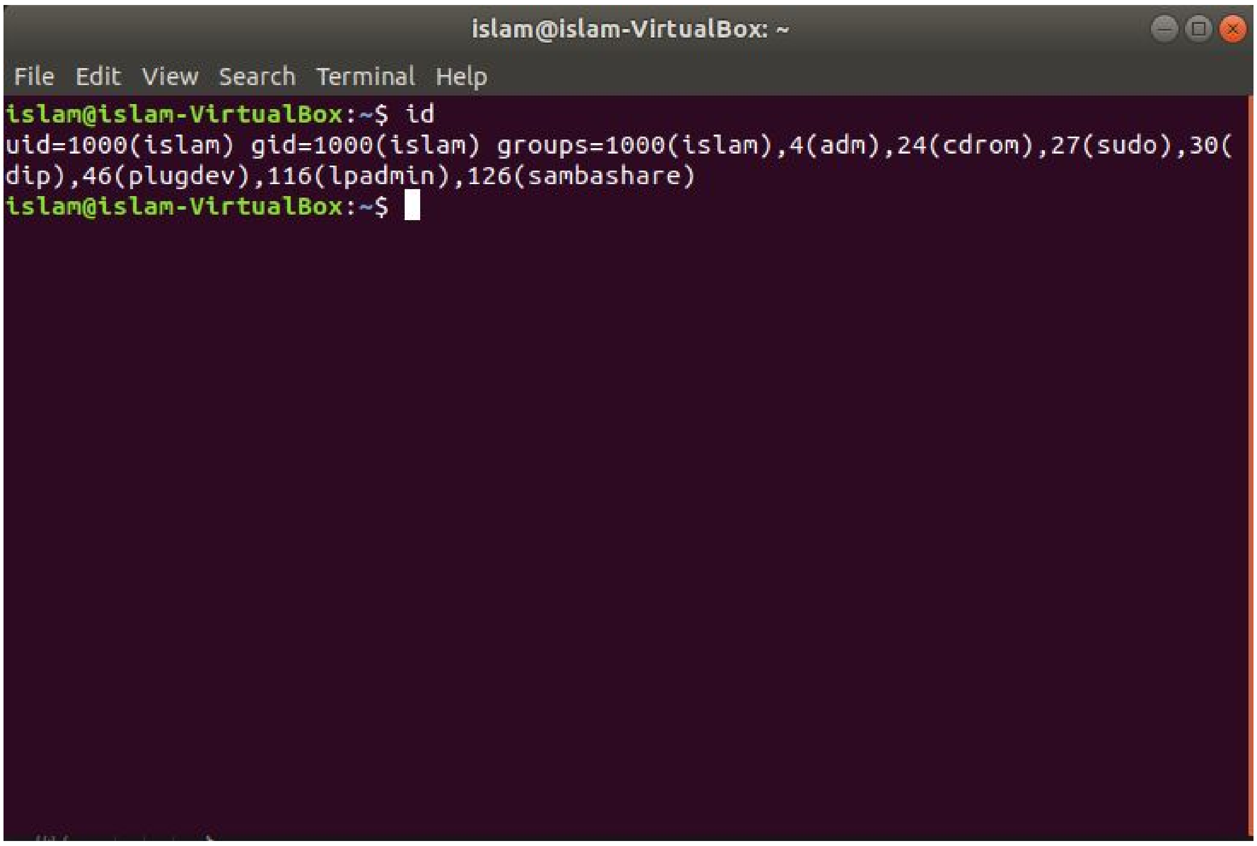
V(new_cred->euid) = V(new_cred->egid) = 0;

V(new_cred->suid) = V(new_cred->sgid) = 0;

V(new_cred->fsuid) = V(new_cred->fsgid) = 0;

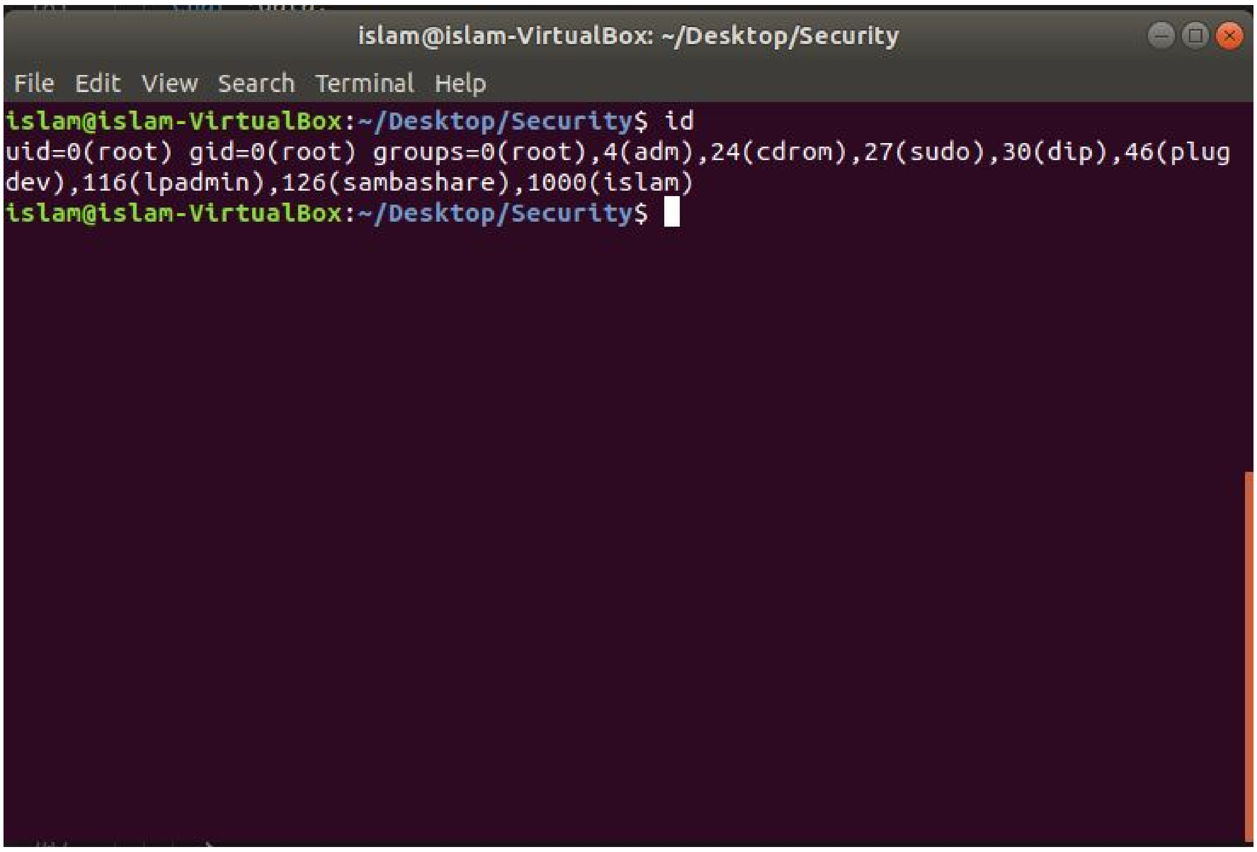
Setting these values to 0 enables root access.

Normally these values are set to 1000 which indicates normal user mode:


islam@islam-VirtualBox: ~
File Edit View Search Terminal Help

```
islam@islam-VirtualBox:~$ id
uid=1000(islam) gid=1000(islam) groups=1000(islam),4(adm),24(cdrom),27(sudo),30(
dip),46(plugdev),116(lpadmin),126(sambashare)
islam@islam-VirtualBox:~$ 
```

In case of root mode:


islam@islam-VirtualBox: ~/Desktop/Security
File Edit View Search Terminal Help

```
islam@islam-VirtualBox:~/Desktop/Security$ id
uid=0(root) gid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plug
dev),116(lpadmin),126(sambashare),1000(islam)
islam@islam-VirtualBox:~/Desktop/Security$ 
```

You can see here that the values are set to 0.

So enabling root access is done by setting those values to 0.

The keylogger:

I have 3 main functions for the keylogger

```
void keycode_to_string(int keycode, int shift_mask, char *buf,
int type)
```

This function translates the integer keycode returned when a key is pressed to a readable name in English

```
static ssize_t keys_read(struct file *filp,char *buffer,size_t len,

    loff_t *offset);
```

This function reads the contents of the keys file that contains the keys pressed

```
static int keysniffer_cb(struct notifier_block *nblock,unsigned long
code,void *_param);
```

This is a callback function called by the keynotifier when a key is pressed. This function then calls the keycode_to_string function and copies the the English name of the pressed key to the buffer.