Rootkit – by

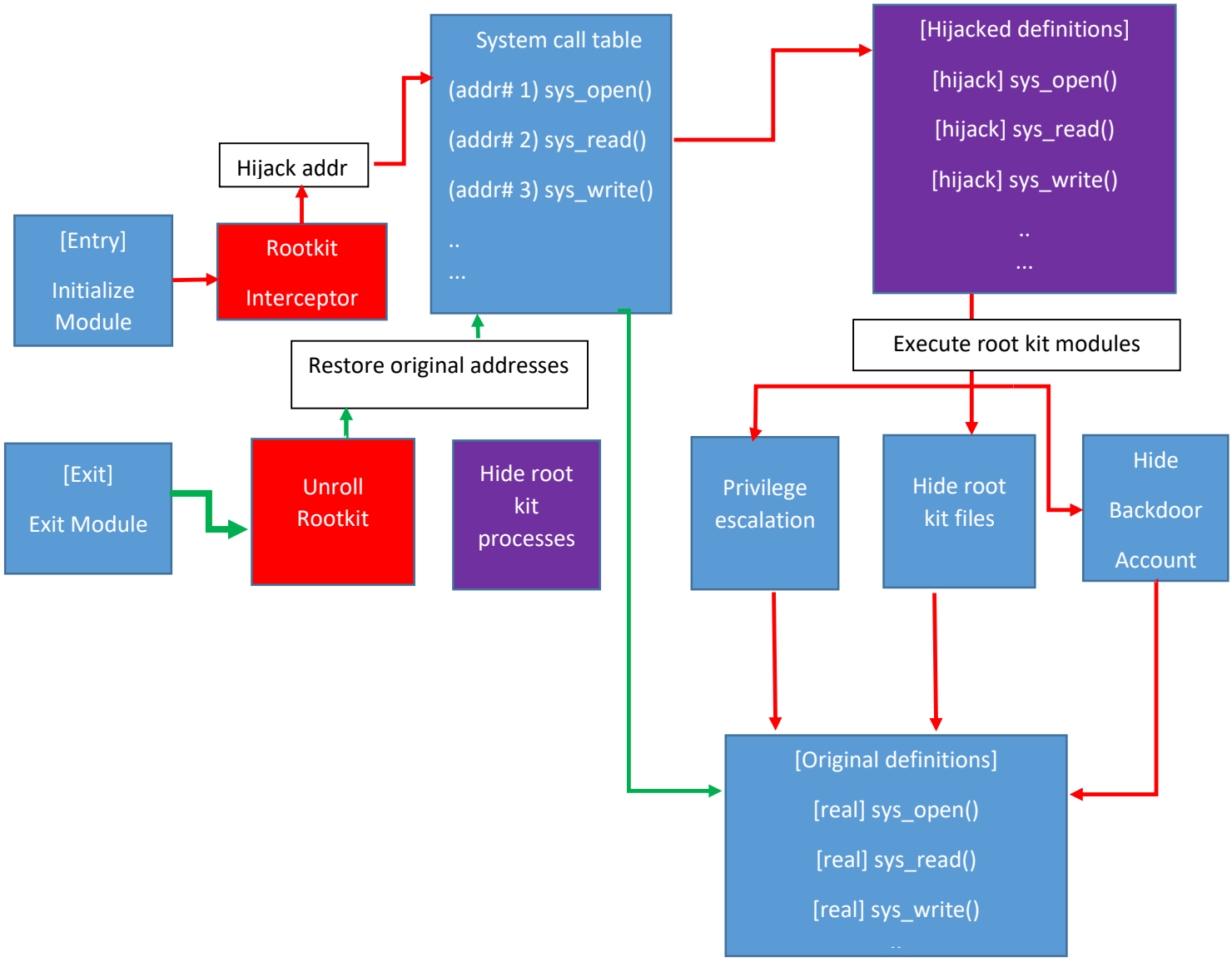
[Meghana D - 110739940]
[Prasoon Rai - 110921754]
[Shubhada Patil - 110284246]
[Siddharth Kawar - 110739852]

1. Introduction:

Rootkit according to Wikipedia is a collection of computer software, typically malicious, that is designed to enable access to a computer or areas of its software that would not otherwise be allowed (for example, to an unauthorized user) and often masks its existence or the existence of other software. After attackers manage to gain access to remote machine, they want to keep access to the machine intact without exposing themselves. In this project, we have implemented a rootkit as a loadable kernel module which when loaded in the kernel will do the following:

- Hide the malicious process from showing up when a user does 'ps' or related command.
- Modify the /etc/passwd and /etc/shadow file to add a backdoor account and hide the backdoor entry when a user does cat.
- Give the ability to a malicious process to elevate its user id to root upon demand.
- Hide specific files and directories from showing up when a user does "ls".

2. Project schematic diagram:



3. System Specification

To use our module, you need to have Linux Kernel - 4.4.0-31-generic running on 32bit-X86 architecture.

4. Major Design Decisions:

As soon as we insert our module into the system we scan for the system call table address in the main memory. We do a linear scan of the memory and compare every address with the known address of the **sys_close** system call (which is always exported). As soon as we find the match, we know that we have found the address of the system call table and then store the current function pointer of the system calls that we want to hijack and replace it with our modified function (*_hijack). We have mainly modified three system calls

1) ***long sys_getdents64(unsigned int fd, struct linux_dirent64 __user *dirent, unsigned int count)***
2) ***long sys_read (unsigned int fd, char __user *buf, size_t count)***
3) ***long sys_chdir(const char __user *filename)***

Apart from this we have also modified proc_iterate function for hiding malicious process when ps related commands are executed.

int proc_readdir_new(struct file *file, struct dir_context *ctx)

a) **Hiding Files and Directories**: In order to hide files, we have hijacked the ***getdents64*** system call. Please make a note that there are two system calls getdents and getdents64. We have hijacked getdents64 because it was the one which was being called on 32bit Ubuntu 14.04 based on linux 4.4.0-31-generic. Here, before returning dirent structure back to userspace, we scan the buffer and look at each record. If any record contains the substring "rootkit" in directory name, we skip writing that record in the user struct dirent buffer. So, this way directory records containing substring "rootkit" are never returned to user. Therefore, such files and directories never shown up on ls and related commands.

b) Adding Backdoor: In order to add the backdoor account, we have added the hardcoded string
"devil:x:1002:1002:devil lurking,,,:/home/devil:/bin/bash"
"devil:1xyz$jYhggMsejqNh4czc7hl8N/:17121:0:99999:7:::\n"
to "/etc/passwd" and "/etc/shadow" respectively. The **normal user** can see the contents of the /etc/passwd but the user cannot change it. Also the normal user cannot read the contents of the /etc/shadow. If we execute "cat" command on "/etc/passwd" the backdoor will not show up, provided that module is loaded.

c) Hiding malicious Processes: In order to hide processes, we have hacked the /proc filesystem. We have modified procfs filesystem's **iterate** function to use our helper function **filldir**. This **filldir** function is responsible for filling the directory information in the user dirent buffer that is provided when the call is made to getdents system call. We skip records of our malicious

processes while filling user's dirent buffer. We identify our malicious processes/executables which contains the keyword "hack" in their name. We could have simply reused already modified **getdents** system call but that increases the execution time of the ps command and it is very noticeable. So, in order to not prolong the execution time of the ps command we chose to modify the procfs filesystems iterate method.

int proc_iterate(struct file *file, struct dir_context *ctx)

d) **Privilege escalation:** In privilege escalation, we modified the ***chdir*** system call.

long sys_chdir(const char __user *filename)

To elevate to root privilege, process makes a call to this system call by passing "abrt" string as filename and upon successful execution it will become root.

Finally, when we remove our module, we restore all system calls to their original state.

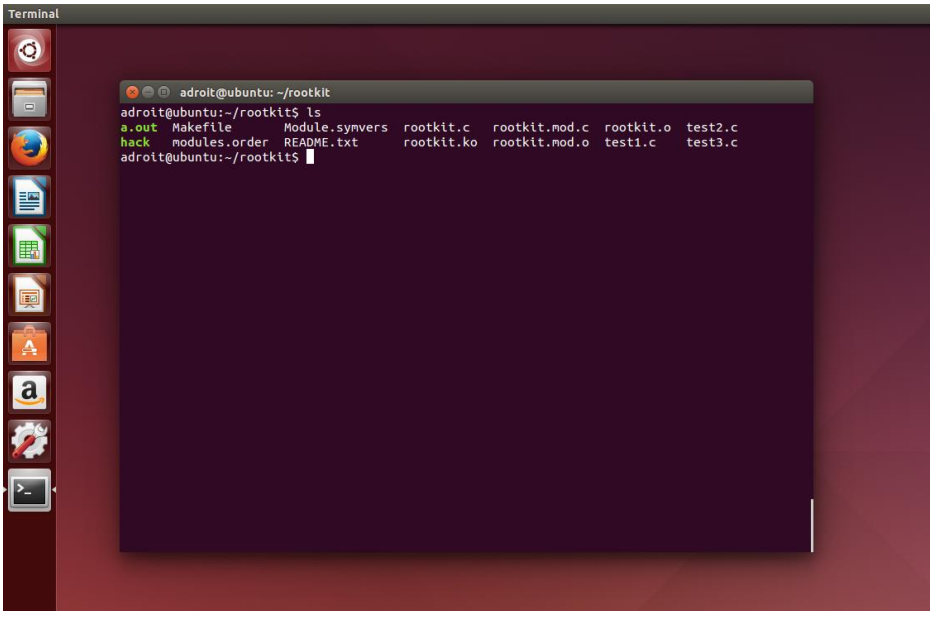
5. Module wise summary and execution outputs:

5.1 Hide root kit files and directories:

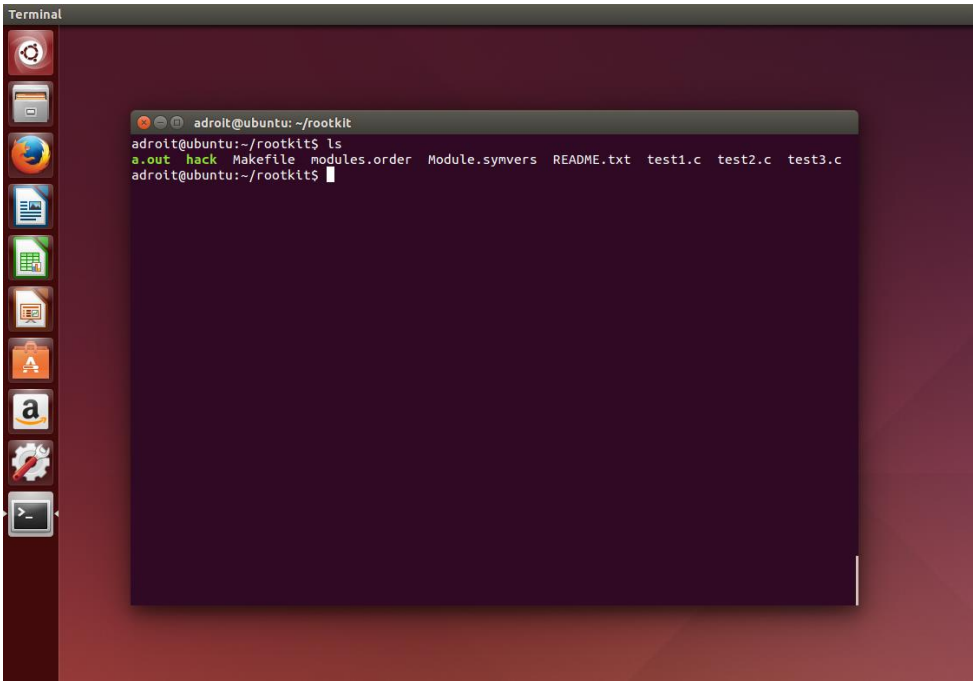
- This module is responsible for hiding files installed by the attacker as part of rootkit.
- Once active, files and folders with substring 'rootkit' do not list upon executing 'ls' command.

5.1.1 Execution outputs:

1. Before loading rootkit, all the files are visible:



2. After installing rootkit module, files containing substring “rootkit” don’t list upon ls:

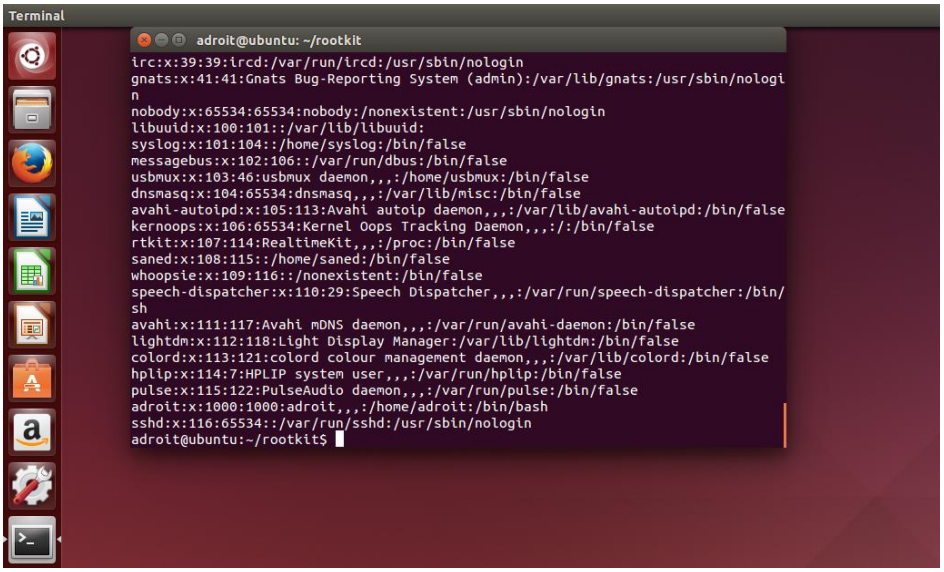


5.2 Hide back door account:

- This module is responsible to hide the password entry of the attacker.
- Once active, /etc/passwd does not show attacker’s entry upon executing ‘cat’ command.

5.2.1 Execution outputs:

1. After installing rootkit module, user devil_lurking is added, but not visible on ‘cat /etc/passwd’:



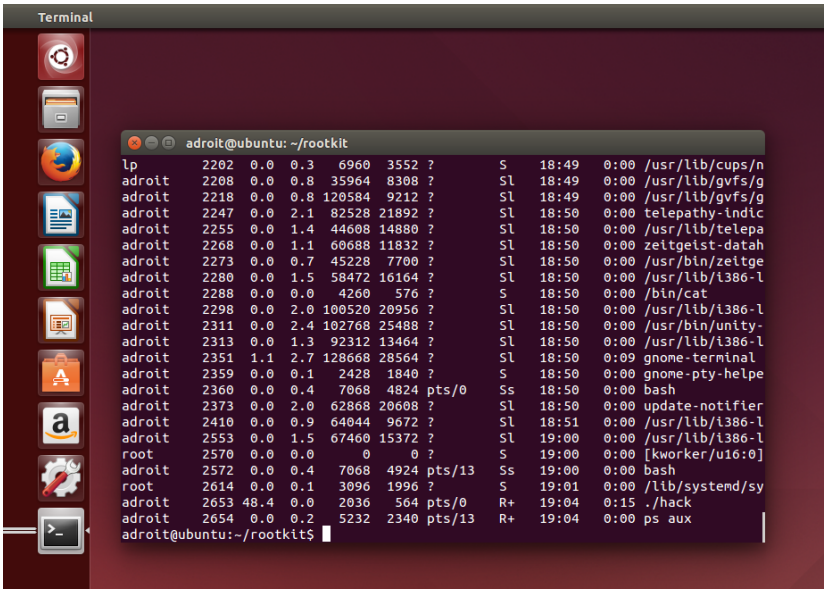
5.3 Hide root kit processes:

- This module is responsible for hiding attacker's processes.
- Upon execution, 'ps' command does not list specific processes when module is loaded.
- If there are child processes spawned by attacker processes, they get hidden as well.

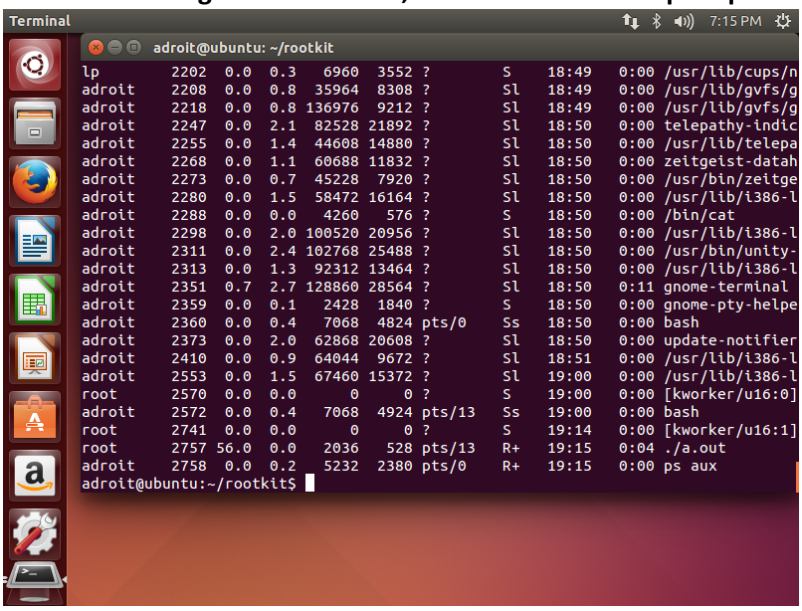
5.3.1 Execution outputs:

Experiment 1: Test file : test3.c executed as "gcc test3.c -o hack ", "./hack"

1. Before loading rootkit, attacker's process 'hack' is visible:



2. After installing rootkit module, 'hack' does not list upon 'ps' command:



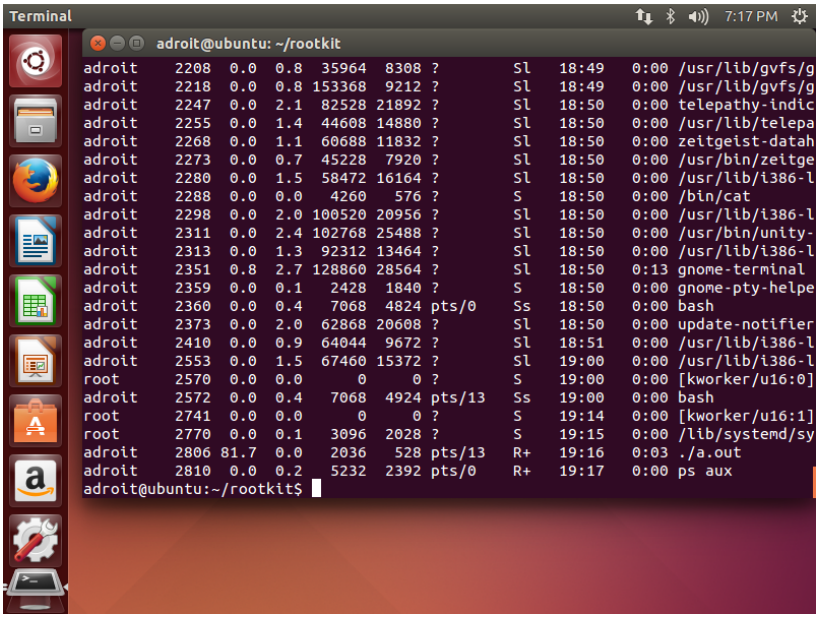
5.4 Privilege escalation:

- This module is responsible to enable the attacker acquire root privilege.
- Once active, the attacker would be able to gain root access.

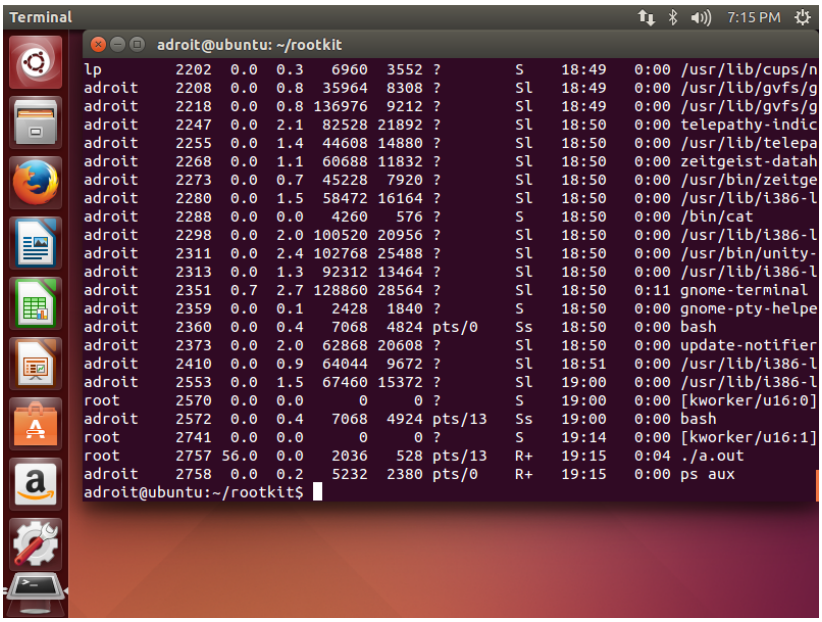
5.4.1 Execution outputs:

Experiment 1: Test File : test1.c compiled as "gcc test1.c" and executed with "./a.out"

1. Before loading rootkit, user is listed as running user for process ./a.out ('adroit' in our case):

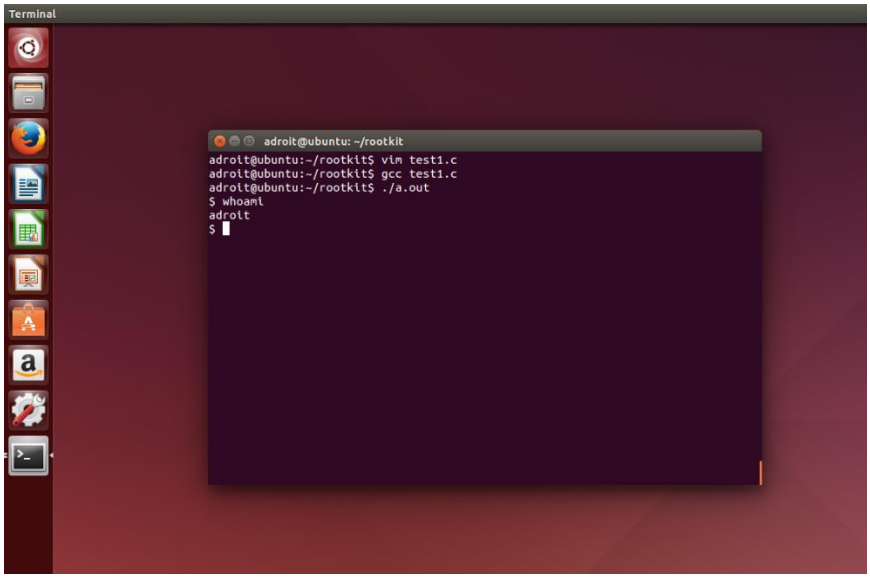


2. After loading rootkit module, user name lists as 'root' for process ./a.out:

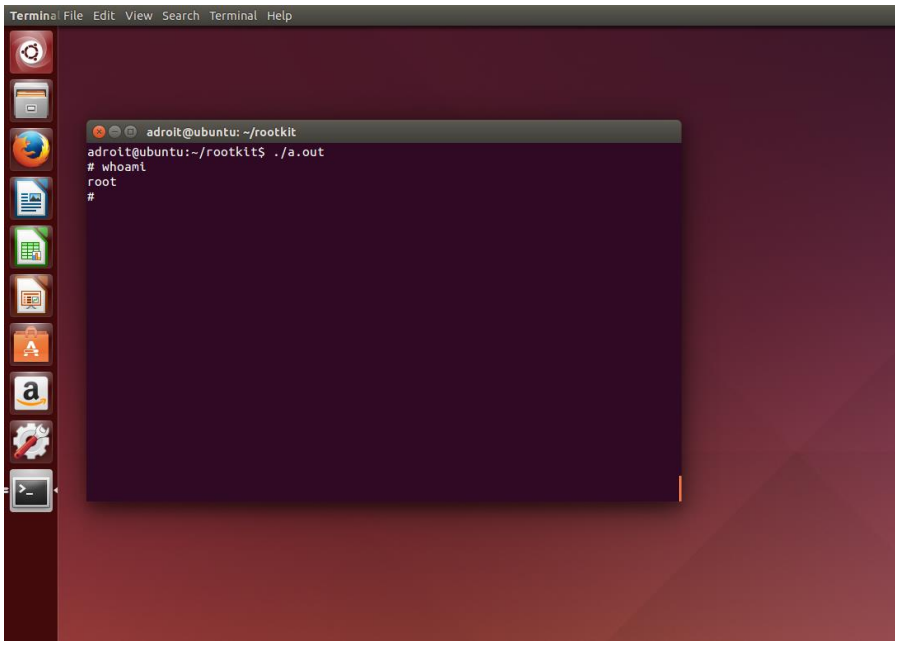


Experiment 2:

1. **If rootkit is not loaded, executing test4.c does not escalate privilege ('whoami' outputs the running user):**



2. **After loading rootkit and executing test4.c, privilege is escalated to root.**



6. Limitations

a) While removing our module, one needs to remove the backdoor account manually, which can be done easily by executing “sudo userdel -f devil”.

b) After removing our module, closing the terminal results in minor freezing of the window. We are not certain about the root cause and are exploring possible solutions.

7. References

- http://www.cbs.dtu.dk/cgi-bin/nph-runsafe?man=getdents64
- https://kernelnewbies.org/Documents/Kernel-Docbooks?action=AttachFile&do=get&target=procfs-guide_2.6.29.pdf
- https://linux.die.net/lkmpg/x710.html
- https://www.kernel.org/doc/Documentation/security/credentials.txt
- The Linux Kernel Module Programming Guide
- http://phrack.org/issues/58/6.html
- http://turbochaos.blogspot.com/2013/09/linux-rootkits-101-1-of-3.html
- http://stackoverflow.com/questions/26000691/system-call-interception-via-loadable-kernel-module