CSEN 1001: Computer and Network Security
Project Report
Linux RootKit

Abdelrahman Khaled Amer
34-9791 T.09

Ahmed Ayman Abdelhamid
34-3944 T.16

Mark Rofail
34-13137 T.10

Muhammad Khattab
34-14154 T.12

Omar Yasser Abdelraouf Ali
34-4675 T.16

2 May 2019

Introduction

Motivation

Linux has been one of the most secure operating systems ever since it was first created a little over 25 years ago.

However, attackers can use RootKits to allow themselves complete access over the machine they choose. To do so, first they need to install the RootKit onto that user's PC, and install their desired modules/malware. When that is done the attacker has full control.

We chose to implement the Linux RootKit as we believed it was probably the most fun project out of all the proposed projects because it covers a lot of ground that we haven't previously discussed in other courses.

Linux RootKit

A RootKit is a piece of software that an attacker uses to provide himself/herself access to to a system or machine, that they are otherwise unauthorized to access, where they can act maliciously by going through private files, intercepting messages, adding other malicious software and locking the original user's data away for ransom or other reasons.

RootKits usually are very specialized in the sense that they can only target a specific kind of system, that is because it is very difficult to generalize the different types of systems and different practices used in making this systems, For example a RootKit might be able to handle an EXT file system (Linux), but not an NTFS file system (Windows).

A Linux RootKit is a RootKit that is made to specifically target the Linux operating system.

Project Description

For our RootKit we were asked to and have succeeded in implementing a RootKit that has the following features:

- Gets root access
  This ensures that the RootKit doesn't have to ask the user for permission to install anything it wants. (It becomes the a super user)
- Hides its modules
  This ensures that the kernel modules installed by the RootKit (and the RootKit itself) cannot be detected by the user.
- Hides a process
  This ensures that a process chosen by the RootKit cannot be detected by the user.

Tasks

The project was split into 4 smaller tasks and each task was given to a team member in order to make things fair and speed up the process. The distribution was as follows:

Task	**Team Member(s)**
Research and resource collection	Ahmed Ayman Muhammad Khattab
Root Access	Mark Rofail
Module Hiding	Omar Yasser
Process Hiding	Abdelrahman Khaled Ahmed Ayman

Methodology

Root Access

In order to load and remove kernel modules, root access is required. Once the root access is acquired, it will be possible to use `insmod` and `rmmod`.

Berkley Pocket Filter is used to allow a user-space program attach onto any socket. The reason behind this is to allow or disallow certain types of data through. This is accomplished by defining a filter code, then the verifier checks the code for threats and then deploys it. The intuition here is to bypass the verifier to achieve our goal of hacking the system.

The verifier runs or emulates the BPF code after performing the check. The two statements `(u32) r9 = (u32) -1` and `if r9 != 0xffffffff` are the ones that we will try to exploit. At runtime, the int value `0xffffffff` will be first cast to an unsigned int value and then assigned to an unsigned 64-bit value, which means that the new value stored in `r9` is `0xffffffffffffffff`. This causes the condition to succeed and jump to `pc + 2`. Since the verifier never saw that this branch would be possible, it did not check for malicious code there. Hence, that is where we put the malicious code. What is left is to leak the value of frame pointer via `bpf_lookup_elem`, from which we can get the address of the top of the stack. Since the struct `task_struct` is stored in the `thread_info`, which is located on the top of the stack, we just need to leak the value of `task_struct->cred` and set `task_struct->cred->uid` to 0. After this step, the root access is gained.

Process Hiding

Sometimes an attacker would like to hide a process from the active user. Perhaps they would like to hide a key logger, or a crypto currency mining software or any other malicious software that does something without the user's consent.

The user can easily view the processes that are currently active and choose to terminate ones that he/she believes could be dangerous or useless. It is in the attacker's best interest to make these processes invisible to the user so that they may continue their attack. To do so the attacker must first understand how the Linux displays the active processes to the user.

The user can use the `ps` command in any terminal to view the processes that are currently running in that terminal (this command makes use of the `procfs` or the process file system). By prefixing the `strace` command to the `ps` command we can have the terminal print all the system calls that were made to run the `ps` command.

Of the system calls that are displayed we are only interested in two specific lines. The line that says `open("/proc",...) = {id}` which opens the `/proc` folder that contains a list of all currently active processes and gives them an id so that they can be used later, and the line that says `getdents({id},...)` which uses that id to print to the terminal.

Each process in the `procfs` has pointers to functions that are called when ceratian actions are executed on it. For example, it will call a function when a read operation is performed on it, or another different function when a write operation is performed on it.

If we look in the Linux OS kernel source tree files we will find that the `getdents` function calls a function called `iterate_dir` which in turn calls the function `iterate_shared`. `iterat_shared` takes in a function pointer as one of its parameters and calls that function when the read occurs. For the sake of simplicity, let's call that function `f_read`.

If we wish to hide the process, then all we have to do is create our own version of the `iterate_shared` function and the `f_read` function and implant them into the Linux kernel without the user noticing. That way we can stop the user from viewing any and all processes that we don't want him to view.

We implemented this by creating our own `iterate_shared` function that returned a 0 when given the process id of the process we didn't want the user

to view. Details on how to run this are in the section **How To Run** below.

Module Hiding

This feature is considered with hiding a module from a list of modules. In our case, it is of extreme importance to run this functionality before doing anything with the rootkit. This is in order that owner would not notice any suspisicous activity.

Two crucial locations are to be considered in order to hide a module. The first location is `"/proc/module"`. This location has the modules on the system, storing the addresses of the modules and their states. The second one is `"/sys/module"`, which contains information on the currently loaded kernel objects.

To hide modules from the first location, the method `list_del_init(&__this_module.list)` was used to delete the entry for the module. Similarly, `kobject_del(&THIS_MODULE->mkobj.kobj)` was used to delete the entry from the second location.

How to run

Needed Software

Ubuntu 16.04 Desktop , Preferably run it on a VM.
Kernel version : 4.15.0-45 Generic.

Hiding the module

1. Compile the make file inside hide module folder.
   ```
   $ sudo make
   ```

2. Insert the module using
   ```
   $ sudo insmod mhide.ko
   ```

3. Search for the module using
   `$ lsmod | grep mhide`
   You will notice it is hidden

Hiding a process

1. create any process for example
   `$ sleep 600 &`
2. run `$ ps` to get the process id
3. Copy the PID to the process hide/phide.c file in the variable `*proc_to_hide`
4. Compile and insert the module same as hiding the module
5. Run the command to list processes
   `$ ps`
   You will notice the process is no longer visible

Gaining super user access

1. Compile the make file inside privilege_escelation folder.
   `$ sudo make`
2. Insert the module to gain the root access

Bibliography

[1] https://github.com/brl/grlh/blob/master/get-rekt-linux-hardened.c

[2] https://ricklarabee.blogspot.com/2018/07/ebpf-and-analysis-of-get-rekt-linux.html

[3] http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction

[4] http://turbochaos.blogspot.com/2013/09/linux-rootkits-101-1-of-3.html

[5] https://yassine.tioual.com/index.php/2017/01/10/hiding-processes-for-fun-and-profit

[6] https://github.com/m0nad/Diamorphine

[7] http://average-coder.blogspot.com/2011/12/linux-rootkit.html

[8] https://d0hnuts.wordpress.com/2016/12/21/basics-of-making-a-rootkit-from-syscall-to-hook