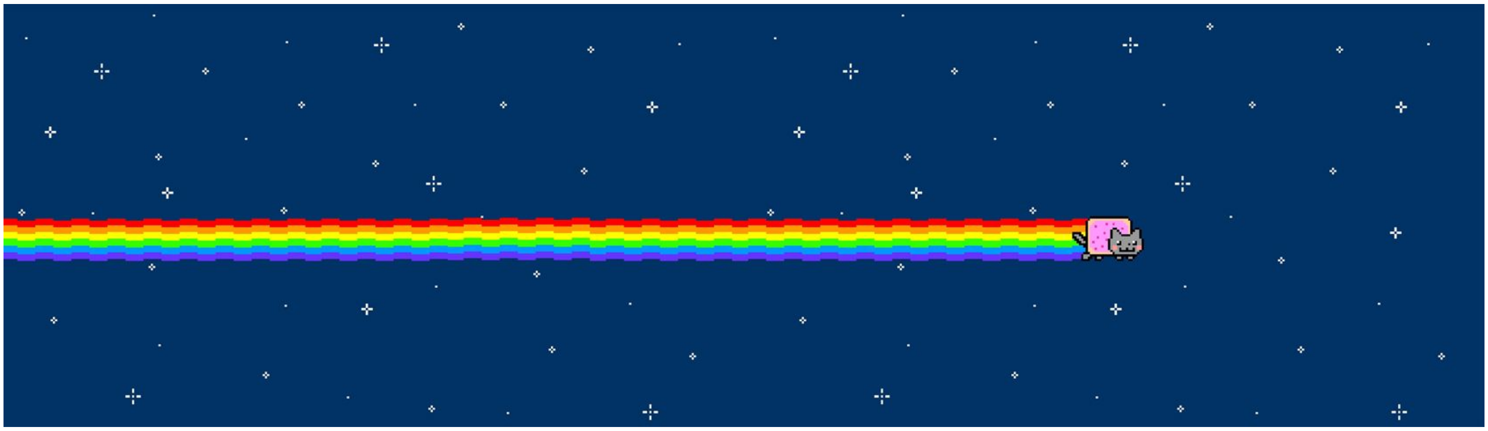
eunuchs



Project Description

Project 1: Linux Rootkit

After attackers manage to gain access to a remote (or local) machine and elevate their privileges to "root", they typically want to maintain their access, while hiding their presence from the normal users and administrators of the system.

In this project, you are asked to design and implement a basic rootkit for the Linux operating system (you can choose the exact distribution and kernel number). This rootkit should have the form of a loadable kernel module which when loaded into the kernel (by the attacker with root privileges) will do the following:

- Hide specific files and directories from showing up when a user does "ls" and similar commands (you have to come up with a protocol that allows attackers to change these)
- Modify the /etc/passwd and /etc/shadow file to add a backdoor account while returning the original contents of the files (pre-attack) when a normal user requests to see the file
- Hides specific processes from the process table when a user does a "ps"
- Give the ability to a malicious process to elevate its uid to 0 (root) upon demand (again this involves coming up with a protocol for doing that)

Note that all of these should happen by intercepting the appropriate system calls in the Linux kernel and modifying the results. You should not perform the above by replacing the system binaries (like "ls", or "ps").

Team Information

Team Name: **i can haz r00t?**

Team Members:

- Daniel Calabria (daniel.calabria@stonybrook.edu)
- Brendan Kondracki (brendan.kondracki@stonybrook.edu)
- Aidan Russel (aidan.russell@stonybrook.edu)
- Edgar Sit (edgar.sit@stonybrook.edu)

Target Information

- Targets Debian 10.1.0, x86-32bit
- Kernel 4.19.67-2+deb10u1

Build & Install Requirements

Download and install the Debian 10.1.0 i386 ISO (available here).

Install build tools & the Linux kernel headers on the system. `sudo apt-get install build-essential linux-headers-($uname -r)`

Turn off address-space layout randomization in the kernel. As root:

1. add `nokaslr` to /etc/default/grub in GRUB_CMDLINE_LINUX_DEFAULT
2. execute `update-grub`
3. reboot

Find and change the value system call table in `eunuchs.c` :

1. `grep sys_call_table /boot/System.map-$(uname -r)`
2. edit the value of the variable `sct` in `eunuchs.c` to be that of the actual `sys_call_table`

Execute `make` in the source directory.

Usage

To load the module: `sudo insmod eunuchs.ko`

To unload the module: `sudo rmmod eunuchs` NOTE: The LKM must *not* be hidden in order to remove it. `echo lemmesee > /dev/.eunuchs` to show the module in the loaded module list, so that it may be removed.

Basic Design

This module performs the following operations upon loading:

1. Creates a character device (by default, this is named `/dev/.eunuchs`)
2. Installs a backdoor account into `/etc/passwd` and `/etc/shadow` (by default, this account has a username of `me0wza` and a password of `w0wza`).
3. Installs functions which will act as a middleman between user-level system call requests and the actual system calls.
4. Inserts a filter for the file operations in the `/proc` VFS.

About the Character Device

The character device is created so that we may write commands to it in order to interact with the module. This allows us to request that the module do something particular, or change its already defined behaviour.

We may interact with the character device by issuing it any of the following commands:

- `ohaiplzshowallhiding` : shows all entries in the hidden lists (only when compiled with `DEBUG 1`)
- `kthxbye` : hide the LKM from lsmod (NOTE: You can't remove the LKM until after you make it visible again)
- `lemmesee` : show the LKM in lsmod
- `icanhazr00t?` : elevates the user to root credentials
- `ohaiplzhideproc [pid_to_hide]` : hides specified process by pid
- `ohaiplzshowproc [pid_to_show]` : shows specified process by pid
- `ohaiplzhidefile [ext]` : hide all files ending in [ext]
- `ohaiplzshowfile [ext]` : show all files ending in [ext]

About the `/proc` filter

During module initialization, since process hiding occurs via removing certain entries from the (constant) `/proc` directory, we alter the file operations associated with the inode for the `/proc` VFS to use our own `filldir` function instead of the default `filldir` function. This allows us to handle process filtering in one easy place, rather than having to determine the full path of directory entries in `getdents` and altering the behaviour there. Our `filldir` function acts a filter which determines if any particular entry should be removed from a listing (returning `0`), or if we can allow the default `filldir` function to execute as normal (returning the entry).

System Call Interception

The module contains functions which act as filters (or middle-men) for certain system calls. The module saves and overwrites the location of the following functions in the kernel, adding the following functionality:

- `read`
- `getdents`
- `getdents64`
- `stat64`
- `fstat64`
- `lstat64`
- `setuid32`
- `kill`

read

Our new `read` handler checks to see if the file being read is either `/etc/passwd` or `/etc/shadow`. If neither of these files is the target of the read operation, allow the kernel-provided system call to process the request as normal. However, if the target of the read operation *is* one of these two files, we may have to filter the file contents. We don't want regular users (including root) reading these files and seeing the injected backdoor account. On the other hand, we *do* want certain authentication-type programs to be able to see our injected account (such as `sshd`, `login`, `gdm`, and other login-type programs), or else having the backdoor account will serve no purpose.

We can determine whether or not to filter the file contents by checking how far away from the `init` process the request was made. By assuming that most programs which we would want to see the account are system daemons, we know that they should be much closer to the `init` process than a user whch is requesting to view the files. We can set a threshold for this `amount of parents` value. For example, upon ssh'ing into the box, `sshd` spawns a child, which in turns spawns another child. The resulting process hierarchy thus looks like `systemd -> sshd -> sshd -> sshd`.

The `pstree` command shows us that the last two belong to the same process group, so we can traverse up the process tree to `systemd` by getting the process group leader of the current context, and keep checking the next higher's group leader's parent, until we hit pid 1. If the amount of traversals is less than our threshold (in this case 1 traversal), we do not filter the file contents.

Conversely, most users attempting to read the file by less/cat/vim/nano/etc will have a process hierarchy that looks more like `systemd -> systemd-user -> gnome-terminal -> zsh -> less`.

All of these processes belong to different process groups, so when we do the same traversal on this tree as before, we end up going over our threshold and we know to filter the contents of the files.

getdents / getdents64

Our new `getdents` / `getdents64` handlers act as filters between the original system calls and the user-land requests. Our handlers first call the kernel-provided system call, then iterates over each returned result to see if it contains a suffix which we should be hiding. If it does, the corresponding entry is filtered out before being returned to the user-land process. This prevents certain files from being returned in a directory listing, and is used to prevent files from showing via the `ls` command.

stat64 / fstat64 / lstat64

Our new `stat64` / `fstat64` / `lstat64` handlers also act as filters between the original system calls and user-land requests. If the target of the request is a file which contains a suffix which we should be hiding, our handlers return `-ENOENT` to indicate there is no such entry in the filesystem. Otherwise, the kernel-provided system call is executed and returned.

setuid32

Our new `setuid32` handler checks if the supplied target UID is the same as our `EUNUCHS_MAGIC_UID`, which is defined in the source file (by default, this value is `0xdeadc0de`). If the target UID is the same as our sentinel value, we elevate the credentials of that process to be that of root. By calling `setuid(EUNUCHS_MAGIC_UID)` during it's execution, a program can elevate its credentials at runtime. If the target UID does not match our sentinel value, then we allow the kernel-provided system call to execute as normal.

kill

Our new `kill` handler checks if the supplied signal is the same as our `EUNUCHS_MAGIC_SIGNAL`, which is defined in the source file (by default, this value is `42`). If the signal is the same as our sentinel value, we elevate the credentials of the process which raised that signal to be that of root. By executing `/usr/bin/kill -s 42 [any_pid]`, the user's credentials can be elevated at a desired time.

Variables

- `*sct` - This is the address of the system call table. Change this value to that returned by `grep sys_call_table /boot/System.map-$(uname -r)`.
- `EUNUCHS_DEVICE_NAME` - This is the desired name of the character device created in `/dev`.
- `EUNUCHS_DEFAULT_HIDE_EXT` - Files that end in this will be hidden by default when the module is loaded. This should be the same as `EUNUCHS_DEVICE_NAME` to hide the created character device from directory listings by default.

- `EUNUCHS_MAGIC_UID` - The sentinel value to watch for in the `setuid` intercept to signal a request to elevate credentials. By default, this is `0xdeadc0de` .
- `EUNUCHS_MAGIC_SIGNAL` - The sentinel value to watch for in the `kill` intercept to signal a request to elevate credentials. By default, this is `42` .
- `EUNUCHS_PASSWD_MOD` - The line which should be injected into `/etc/passwd` . By default, this injects an account with the credentials of username `me0wza` , password `w0wza` , and UID `31337` .
- `EUNUCHS_SHADOW_MOD` - The line which should be injected into `/etc/shadow` .

POCs

Account Injection

To verify account injection, perform the following steps:

1. Insert the module.
2. Remove the module.
3. `sudo cat /etc/passwd` and `sudo cat /etc/shadow` . Verify that an account with the name `me0wza` has been added to these files.
4. Insert the module again.
5. `sudo cat /etc/passwd` and `sudo cat /etc/shadow` . Verify that the `me0wza` account is no longer visible.

At this point, we have verified that the account is injected into these files and is not visible to the user. In order to verify that login daemons can still see the account, perform the following steps:

1. Ensure that an ssh client and server are installed. `sudo apt-get install openssh-server openssh-client`
2. Change the target runlevel to be multi-user (boot to console, instead of graphical user interface). `sudo systemctl set-default multi-user.target` (to later rever this change, set the target runlevel back to `graphical.target`)
3. Reboot to get non-graphical login program
4. Log in and load the module.
5. `ssh -l me0wza localhost` , then enter `w0wza` when prompted for the password.
6. Log out of ssh session.
7. Log out.
8. Log in with the backdoor account.

Hiding / Showing Files

To verify that the module hides files, perform the following steps:

1. `ls -l tools/` and verify that there is a file named `meow.eunuchs` listed.
2. `stat tools/meow.eunuchs` and verify that the stat command successfully returns the information associated with this file.
3. Insert the module.
4. `ls -l tools/` again, and verify that there is no longer a file named `meow.eunuchs` listed.
5. `stat tools/meow.eunuchs` and verify that there is an error message stating that no such file or directory exists.

In order to further test this functionality, one repeat the above process (with the proper substitutions for the filename) after altering the lists of file suffixes which should be hidden by the module.

1. `echo ohaiplzhidefile .c > /dev/.eunuchs` to hide files ending in `.c`
2. Perform the steps above to verify that `.c` files are hidden.
3. `echo ohaiplzshowfile .c > /dev/.eunuchs` to show files ending in `.c` again.

Hiding / Showing Processes

To verify that the module hides processes, perform the following steps while the module is loaded:

1. Execute `ps aux` and pick a random process with a long running time (`init` is good, since it will always be running, and has a constant PID of `1`).
2. `echo ohaiplzhideproc 1 > /dev/.eunuchs`
3. Execute `ps aux` again and verify that the entry for the selected process is no longer listed.

Credential Elevation

To verify that the `setuid` intercept works, perform the following while the module is loaded:

1. `gcc -o icanhazshell tools/icanhazshell.c`
2. `./icanhazshell`
3. Verify a shell is spawned which has root credentials.

To verify that the `kill` intercept works, perform the following while the module is loaded:

1. `id` and `whoami` . Verify that you do not have root priviledges.
2. `/usr/bin/kill -s 42 1`
3. `id` and `whoami` . Verify that you now have root priviledges.

To verify that the character device method works, perform the following while the module is loaded:

1. `id` and `whoami` . Verify that you do not have root priviledges.

2. `echo icanhazr00t? > /dev/.eunuchs` (NOTE: You may need to escape the question mark in this command, depending on your shell.)
3. `id` and `whoami`. Verify that you now have root priviledges.

LKM Hiding / Showing

This module is also capable of hiding itself from being listed when the user executes `lsmod` or views `/proc/modules`. To verify this, perform the following steps:

1. Insert the module.
2. Execute `lsmod` and `cat /proc/modules`. Verify that both commands return a listing which contains the name of this module (`eunuchs`).
3. `echo kthxbye > /dev/.eunuchs`
4. Repeat step 2 to verify that both commands now return a listing which no longer contains an entry for this module.
5. `echo lemmesee > /dev/.eunuchs`
6. Repeat step 2 to verify that the entry has returned.

References

- https://elixir.bootlin.com/linux/v4.19.67/source/
- https://www.tldp.org/LDP/lkmpg/2.6/lkmpg.pdf
- https://www.kernel.org/doc/Documentation/security/credentials.txt
- https://stackoverflow.com/questions/8250078/how-can-i-get-a-filename-from-a-file-descriptor-inside-a-kernel-module
- https://yassine.tioual.com/index.php/2017/01/10/hiding-processes-for-fun-and-profit/
- https://stackoverflow.com/questions/1184274/read-write-files-within-a-linux-kernel-module