Hello World

TASK 1: Kernel Level Rootkit

The *root.c* file is a rootkit that gives nonroot users root access. In this particular case, the name of the module that was inserted into the kernel is *ttyR0*. The *root.c* file essentially uses a magic word, in this case, "g0tR0ot" to trigger the rootkit. The code will import all the relevant header files, define the name and device of the module and setup the function prototypes for initializing, exiting, opening, reading, and writing the module. It will also initialize the `V(x)` function which will be used for setting the id's of the user and declares a `majorNumber` (used by kernel to detect which module to use), class, and device driver. It sets up the module info and maps functions with their respective pointers in the file operations `struct` object. In detail, when the module is first inserted into the machine, it will create a major number and register a character device for the kernel to use in order to identify the machine and will print those them to the kernel log. Next, it will register the device class and its device driver. The type of device class can be found in the `#define` variables towards the top of the code. If any stage of the registration fails, the module will unregister the device class or device driver, throw a pointer error, and will print out an error message. The rootkit will allocate memory into the kernel for a possible input from the user. When a user writes to the module, the module will check if the input is equal to the magic word. If it is equal, then the module will check if preparing credentials is possible. If it not possible, the module will not do anything; however, if preparing the credentials is possible, then the module will set the user's *uid*, *gid*, *euid*, *egid*, *suid*, *sgid*, *fsuid*, and *fsgid* to 0 (the id of root). It will commit those credentials for the current user giving the user root access. Then, it will do some garbage collection and return the the length of the input string. Finally, the module will destroy the device and exit.

TASK 2: What Else?

1. *root access*: by writing a known string to a character, one can allow a non-root user to gain root access
2. *file hiding*: hides files and directories that are used
3. *process hiding*: hide processes that are happening on the machine
4. *self hiding*: hide the rootkit itself
5. *hide transport connections*: hide tcp/udp connections to the machine
6. *hidden boot persistence*: allows user to save data changed back tot he USB storage device instead of leaving leaving information in system memory
7. *file content tampering*: allows one to interfere or modify file content for causing damage or unauthorized alterations
8. *obfuscation*: allows the root make itself/code obscure or unclear to decipher
9. *ICMP/UDP/TCP port-knocking backdoor*: can correctly specify a specific sequence of closed port connections to gain authentication
10. *tty/pty shell with file transfer*: establishes a connection with a machine a enables input into its shell
11. *reverse shell*: enables one to make the target machine to a remote host
12. *eavesdropping*: enables an actor to eavesdrop on users and intercept personal information
13. *file execute*: allows actors to remotely execute other files on target computers
14. *zombify*: appropriates the compromised machine as a zombie computer for attacks on other computers
15. *attack detections*: can detect attacks and move forward with protocols
16. *anti-theft protection*: can have BIOS-based rootkit that will report to central authority and disable or wipe information in the event that it is stolen

17. *tracking*: allows the actor to keylog the computer
18. *disable security applications*: can disable security applications or malware detection systems making your system vulnerable
19. *drop malware*: can drop additional malware to a system
20. *memory dumps*: a rootkit can perform a memory dump from any specific directory