Sahil Khanna (sk5xvh)

Task 1: Kernel-level rootkit:

For this task we followed a tutorial to build a linux kernel rootkit. Our goal was to build a Linux Kernel Module (LKM) that will allow the user to get instant root on whatever machine it is installed on. The code that we copied from the link provided accomplishes this using a character device driver. Since the Kernel is a space that has a lot of access in a machine, if we find a way to access it, we can exploit it to change user permissions. Users do not have a way to communicate with the kernel space by default. The character device driver was used to pass information between the user and the LKM. The LKM allows the user to access the kernel space and therefore manipulate the permissions, etc. As for the code itself, the file operations allow for data transfer between the user and LKM. root_open is called when the user opens the device. root_init is called at initialization. Here is where we set up our character device driver with its major number and this is also where we register it. root_read allows for data transfer from the LKM to the user. root_write is the function where we update our permissions in the kernel. It is the function where the user is able to send data to the LKM. If the keyword “g0tR0ot” is inputted by the user, the user is granted a new set of ‘credentials’ with all the credential values being set to zero. The credentials are UID, GID, EUID, EGID, SUID, SGID, FSUID, and FSGID.The credentials are then stored in the new_cred object. The memory is then freed using kfree(data). There are a bunch of if/else statements to ensure that credentials are only granted if the first seven characters of user input is “g0tR0ot”. In the root_exit method, we destroy the character device driver and unregister it and then we exit the program.

Task 2: What Else?:

1. Hide files/directories: rootkits can hide files/directories with a specific prefix inputted by the user and prevent them from appearing in “ls” or “find” commands
2. Hide processes: rootkits can hide processes from view with the /reptile/reptile_cmd hide <pid> command in the git example given on the task page.
3. Hide User: the rootkit could modify the output of commands such as getuid. It could also hide a certain user in the databases where all the users are stored
4. Hide TCP/UDP connections: rootkits can hide TCP/UDP connections from view in the output of commands that look at those connections. The git example uses a hidden_conn_list variable that holds all the connections that need to be hidden when commands to view connections are called.
5. Hidden boot persistence: The rootkit will persist despite system reboots, etc. by hiding files and processes and hooking system functions.
6. File content tampering: the rootkit can replace file functions with its own versions, modifying the content before returning it
7. Some obfuscation techniques: these are some techniques that make the rootkit more difficult to detect such as encryption and modifying access of users/files to stay hidden
8. ICMP/UDP/TCP port-knocking backdoor: When receiving a specific packet, the program would open a backdoor port.
9. Full TTY/PTY shell with file transfer: a rootkit can be used to create a Full TTY/PTY shell with file transfer by intercepting calls
10. Client to handle Reptile Shell: provides an interface for the user to interact with the rootkit and provide commands.
11. Shell connect back each X times (not default): every x times the rootkit is executed, it will connect back to an IP address via a port
12. Conceal other malware: Rootkits can conceal malware files and programs such as keyloggers and computer viruses
13. Zombie computer: Rootkits can be used to control all functionality of a machine and can convert a machine into a zombie computer.
14. Intercept https requests and leak data: If a rootkit gets access to the networking layers, then it could potentially perform a man in the middle attack
15. Modify file/directory permissions/privileges: Once root-level permissions have been established, the rootkit can use chmod to edit permissions.
16. Detecting attacks: Hide the presence of attack detection software, making it difficult for attackers to see a honeypot for example.

17. Anti-theft protection: Machines can be equipped with rootkits that report periodically over the network to a separate machine so that in case it is stolen, the information can be wiped remotely.
18. Alter boot/startup process: The rootkit can alter the boot process for the machine by putting code in the kernel or modifying the startup program
19. Resource Hogging: The rootkit can create many processes and allocate large amounts of memory, slowing down the computer and possibly leading to a crash.
20. Keylogger: A rootkit can install a keylogger at the kernel level capturing all keyboard output. It monitors the input and output devices of the machine.