Homework 4

Problem 1

(a) "chgrp GROUP FILE/S"
(b) "setfacl -m u:user1:permissions" —> setfacl sets ACLs (Access control lists) of file and directories, while getfacl shows ACLs.
(c) use ls -l FILE to view the permissions, then "chmod u+s serviceA_user1.log"
(d) Setuid, Because it gives instant access to attackers. In Discretionary access control (DAC) access is controlled based on identities of individual users and groups. A key weakness lies in their discretionary nature, which allows the owner of a resource to set or modify its permissions. Another weakness is the unlimited access allowed for the administrator or root user.
, Role-based access control based upon principle of least privilege.
(e) Logrotate is a system utility that manages the automatic rotation and compression of log files. If log files were not rotated, compressed, and periodically pruned, they could eventually consume all available disk space on a system. As long as it is editable only by root, there is no security concerns.

Problem 2

1. Objects and rights to access them (Access Control List)
2. Domains (users) and their rights (Capability List)
(a) An **access list** is a list for each object consisting of the domains with a nonempty set of access rights for that object. A **capability list** is a list of objects and the operations allowed on those objects for each domain.
- Each column in the access matrix can be implemented as an **access list** for one object. The resulting list for each object consists of ordered pairs <domain, rights-set>, which define all domains with a non-empty set of access rights for that object.
- In **capability list,** each row is associated with its domain.

(b) Access lists: Easy to know the access right of a given subject. * Easy to revoke a users access right on all objects.
Capability lists: good for granting rights to a user.

Problem 3

(a) The **fseek()** function is the random access function. For example, when reading structures to a file you can use **fseek()** to set the file position indicator to read a specific structure in the file. As it is only a read function, I don't think that any malicious user can misuse fseek() only by reading it even it has access to the file. **Need-to-know** principle is useful in limiting the amount of damage a faulty process or an attacker can cause in the system. Another thought is, that if runs with root permissions, it will gain complete access to the system.

(b) A race condition is an undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time, but because of the nature of the device or system, the operations must be done in the proper sequence to be done correctly.
-**Race condition** occurs when multiple threads read and write the same variable i.e. they have access to some shared data and they try to change it at the same time. In such a scenario threads are "racing" each other to access/change the data.

- **Lock-free** algorithm that provides protection from race conditions without requiring the over- head of locking.
-**Linked file allocation** method can be used only with sequential access. It does not work effectively with random access.

Problem 4

DRAM is a volatile memory, this means that once you store anything in it, after shutting down your PC, it simply fades away.

Concurrent **reads** from a file (whether from multiple threads -- assuming from separately opened descriptors -- or from multiple processes) is well-defined and permitted on all modern major operating systems. It is only concurrent **writes** to a file which are ill-defined and which you should not attempt to do without locking (unless you are appending to the file, like a log, and the OS makes such concurrent writes well-defined).

Problem 5

(a)
The way of communicating a message from one process to another process is called a **signal**.
When we use the signal command to send a signal to a process that is owned by other users (like root), then we would need admin rights and have the privilege to use the 'sudo' command.
We can replace **setuid** programs using file capabilities, i.e., instead of giving a privilege program the root privilege, we can assign a set of required capabilities to the program. This way, we can enforce the principle of **least privilege**.

(b)
- Linux divides the privileges traditionally associated with superuser into distinct units, known as **capabilities**, which can be independently enabled and disabled. Capabilities are a per-thread attribute.
- The goal of capabilities is divide the power of superuser into pieces, such that if a program that has one or more capabilities is compromised, its power to do damage to the system would be less than the same program running with root privilege.

Problem 6
- An **access list** is a list for each object consisting of the domains with a nonempty set of access rights for that object. A **capability list** is a list of objects and the operations allowed on those objects for each domain.

1. It is an ACL-based approach because we can assume each invited guests as files that need to be specified based on their identities.
2. A **capability** is a token, ticket, or key that gives the possessor permission to access an
3. entity or object in a computer system. A capability table is bound to a domain, whereas an ACL is bound to an object. Thus it is a capability based approach, the personal access can be granted to the matched students in the list.
4. It is an ACL-based approach, because we need to check if the object (here the parking tickets) is obtainable by cash or not.

5. It is an ACL-based approach because it is bound to object (here the room keys) so the keys with special permissions will have access to the specified rooms.

Problem 7

(a) The weakest point is that the network system is used which makes it easier for attackers to gain unauthorized access to the system and to the files e.g. during a ssh connection, becoming man-in-the-middle.

(b) There are many ways to strengthen the weakest points: encryption, authentication . . .

(c) The problem is how to keep the password secret within the computer. The UNIX system uses secure hashing to avoid the necessity of keeping its password list secret. Because the password is hashed rather than encrypted, it is impossible for the system to decrypt the stored value and determine the original password. Although the passwords are hashed, anyone with a copy of the password file can run fast hash routines against it—hashing each word in a dictionary, for instance, and comparing the results against the passwords. If the user has selected a password that is also a word in the dictionary, the password is cracked. And because systems use well-known hashing algorithms, an attacker might keep a cache of passwords that have been cracked previously. Therefore, the salt value is added to the password to ensure that if two plaintext passwords are the same, they result in different hash values which makes the attack much slower.

(d) - Address Space Layout Randomization (**ASLR**) attempts to solve this problem by randomizing address spaces — that is, putting address spaces, such as the starting locations of the stack and heap, in unpredictable locations. Address randomization, although not foolproof, makes exploitation considerably more difficult. ASLR is a standard feature in many operating systems, including Windows, Linux, and macOS.

- Another approach is to place the user data and the system files into two separate partitions. The system partition is mounted read-only, whereas the data partition is read–write. This way system partition is protected because it is not easy to cause damage.

Problem 8

(a) If you didn't write it, you can't trust it.

Rootkit viruses are originally the back-doors on UNIX systems meant to provide easy root access. When malware infects the operating system, it can take over all of the system's functions, including those functions that would normally facilitate its own detection. So no, the last compiled version of the compiler will not be free of the devils additions.

(b) Same answer as (a)

(c) Example: Virus:Win32/Induc.A is a virus that infects Delphi library source files. Any executables compiled/linked by the Delphi compiler on the affected machine will contain the malicious code.

Solution: Enable rootkit detection in the settings, run a full scan with that, then a full scan with Windows Defender.