1 Class #18

This section discusses *Protection* and *Security*. Roughly, *protection* is to be in a state of defence from an attack or invasion. *Security* is a state of freedom from damage, risk or apprehension.

1.1 Protection

For computer systems, *protection* is defined as a mechanism to control access of programs, processes and users to system defined resources. This mechanism only guarentees that the object to be protected will in fact be protected. This needs to be combined with a *policy* which indicates what are the resources that actually need to protected. This usually boils down to two goals:

- Facilitate error detection and limit errors from propagating except at specific interface points.
- Prevent malicious intentional violations by a user

It is easy to see that when there is a single user on a system, both these issues are pretty much irrelevant. The main focus of research into these issues has been multi-programming systems, although it is increasingly a concern for multi-machine networked environments.

Typical examples of security violations include:

- Information disclosure
- Information destruction and modification
- Unauthorized use of service
- Prevent denial of service

A typical example is a **service** that is provided by an **owner** of the service to a bunch of **users**. The following should be ensured by this mechanism:

- No user should be able to steal the service by making a copy
- No user should be able to destroy or otherwise damage the service.
- No user should be able to employ the service without the owner's permission.
- The owner should be able to revoke the access of an authorized user to the service.
- No user should be able to prevent an authorized user from accessing the service.
- The service should not steal, destroy or compromise any information that the user did not explicitly supply.
- A service should not be able to disclose any sensitive information revealed to it by the user.

Usually, all this can be summarized under the **need-to-know** principle.

An operating system can be thought of as a collection of objects on which certain operations can be performed. There are *processes* in the system which try to extract services from the objects. A *protection domain* can be thought of as a set of pairs, where each pair consists of an **object** and an accompanying **access right**. The **access right** identifies which of the operations on the object can be performed by a particular process. When a particular process is in a particular protection domain, it can only perform the operations on the objects it is allowed to. To be able to perform any additional operations, the process must *change* its protection domain.

For example, in UNIX, a process in the user mode belongs in a certain protection domain, whereas a process executing as the super-user is in a different protection domain. By carefully defining the operations that are permitted in each protection domain, the operating system can restrict transfers between domains. In UNIX, the *setuid* bit of a process has to be set for the prorcess to be able to execute as root.

1.2 Access Control Matrix

Given that any domain can perform a (possibly incomplete) set of operations on any of the objects in the operating system, it is reasonable to represent this information as a **matrix**. The rows of the matrix correspond to protection domains and the columns correspond to objects. An entry at row i and column j stores what operations protection domain i can perform on object j. In particular, operations can be **control operations** such as allowing a process to change entries in the matrix, etc. In addition, **domain transition rules** can also be encoded in this setup by having columns corresponding to domains themselves.

Access control matrices can be implemented in one of two ways:

- Each column can be stored with the corresponding object. This mechanism is known as an **access contol list**. However, this approach has two drawbacks:
  - Determining the set of rights for a domain is expensive, since every object in the system must be searched to get this information.
  - Every access to an object must be checked for validity, and since the columns can be quite long, this can be quite expensive as well.
- Each row can be stored with each domain. This approach is called the **capability list** approach.

On most systems, the ACL is searched on the first access to an object. After the first access, a capability is created for it in the domain accessing it and is used for subsequent accesses to the object.

1.2.1 Access Revocation

Another problem that one must deal with is the issue of **access revocation**. When a protection domain's access to an object is revoked, the domain cannot continue to perform the operations that it was performing on the domain. It is easy to see that with the **capability list** approach, revocation can be quite hard. In particular, one must search exhaustively and locate all outstanding capabilities, to prevent rogue accesses to an object. There are several possible solutions to this problem:

Re-acquisition: Periodically, all capabilities of a domain are deleted. Process must reacquire all its capabilities when this happens.

Back pointers: A pointer can be maintained from every object to every outstanding capability for that object. However, this requires the object to be involved in every capability transaction operation.

Indirection: In this approach, an entry in a capability list instead of pointing to the object directly, points to an entry in a table. The entries in the table point to the objects. When the capability of a domain to an object is to be modified, only the table entry corresponding to the object has to be modified. In addition, selective revocation is possible, since two different entries in the table could be pointing to the same object and it is possible to change only one of them.

2 Authentication

Authentication is the process of verifying that the person at "the other end" is really who he claims to be. Typical mechanism used for authentication purposes is the *password* mechanism. However, the passwords have a limited number of charcters and can be guessed with a sufficient amount of computing power. So, *one-time passwords* are another popular mechanism to avoid this problem, where a password can be used to gain access precisely once.

It is a bad idea to store the password in clear text. The UNIX solution is to store only encrypted versions of passwords. The encryption mechanism is not invertible easily. In other words, any given word can be encrypted extremely efficiently. However, given an encrypted word, it takes exponential time (in the size of the encrypted password). However, this still leaves open the potential attack from a user who captures a password file and runs a decryption script for a large amount of time in an attempt to capture other people's password.

Authentication is not a complete solution, however. It is possible for remote programs to execute on a local machine and behave unexpectedly. This is captured in the following types of behaviours:

Trojan horse is a program that misuses its access rights when it runs.

Trap door is a program that exploits some feature of some software that has not been advertized, and which a user can exploit to break into a system.

worm is a program that spawns itself repeatedly to degrade system performance

virus is a malicious fragment of code embedded in a legitimate program

2.1 Encryption and Digital Signatures

Encryption is the process by which a mapping is constructed from input text to new text. The mapping must be easily constructed, however, the encrypted text must be extremely hard to invert.

Puzzle 1 *Given two people A and B who need to exchange some information. All they have two do the communication is a box, a key and a lock each. They can only send stuff to one another in the box by a courier who is completely untrustworthy and who will steal anything from an unlocked box. Find a way for A and B to actually exchange a top secret paper in this situation.*

A digital signature is usually something appended to the end of a message to verify the authenticity of a message. It should have two important characteristics.

Message dependence: Clearly the signature should be message dependent, else someone else could add stuff to or remove stuff from a message without changing the signature or the authenticity. (this possible modification is known as *electronic cut-and-paste*.

Sender dependence: Obvious

Here is a mechanism that satisfies a digital signature protocol: Let A and B be two communicating parties. Let D_A and D_B be two decryption functions known only to A and B respectively. Similarly, let E_A and E_B be two encryption functions known to A and B only. If these algorithms satisfy certain properties, then using the approach of solving the puzzle above, we can find a way for A and B to communicate.

2.2 How to share a secret

A (k, n) threshold mechanism is defined as follows: Let D be an item of data to be communicated and let it be broken down to n components. Let this breaking down satisfy the property that to recover the initial datum,

Knowledge of k or more pieces of D makes D computable.

Knowledge of less than k pieces of D leaves D completely undetermined.

Theorem 1 *Given k points in the plane, there is exactly* **one** *polynomial of degree k-1 passing through all the points.*

Then, pick a random $k-1$ degree polynomial, say Q(x). Instead of sending out the pieces of D by themselves, break up D into n parts, and for each part, actually send the pair $< i, Q(i) >$. This mechanism ensures that fewer than k pairs captured will leave D completely undetermined, but k pairs captured will determine D completely.