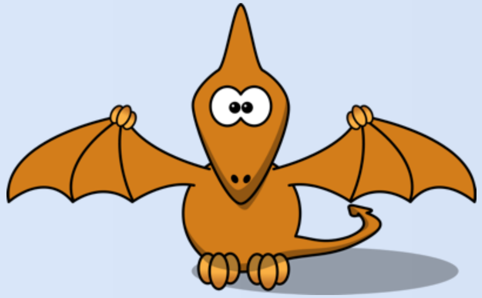
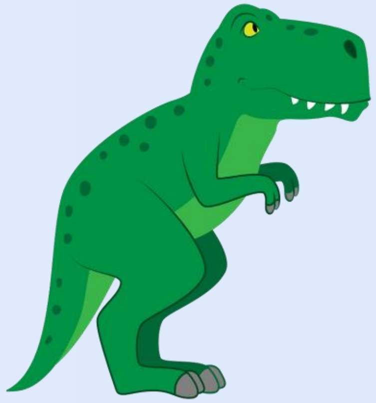


Chapter 16-17.

Security & Protection



Operating System Concepts (10th Ed.)

Part 7. Security and Protection

- ***Security***
  - ensures the ***authentication*** of users to protect
    - the integrity of the *information*, both *data* and *code*.
    - the *physical resources* on the computer system.

- ***Protection*** ensures that only processes
  - that have gained proper ***authorization*** from the operating system
    - can operate on *memory segments*, the *CPU*, and *other resources*.
  - *controls access* to a system
    - by *limiting* the types of *access permitted* to users.

16.1 Security Problem

- Computer resources
  - may *be misused accidentally* or *purposely*.
  - hence, we need a mechanism to *guard against* or *detect attacks*.
  - *cryptography* is a key security enabler.

- *Security* involves *guarding* computer resources against
  - *unauthorized* access,
  - *malicious* destruction or alteration,
  - and *accidental* introduction of *inconsistency*.

16.1 Security Problem

- Security Violations
  - *threat*: the potential for a security violation, *accidental*.
  - *attack*: an attempt to break security, *intentional* (malicious).
- The types of Security Violations:
  - Breach of *confidentiality*.
  - Breach of *integrity*.
  - Breach of *availability*.
  - *Theft of service*.
  - *Denial of service* (DoS): Distributed DoS (*DDoS*).

16.1 Security Problem

- Four Levels of Security:
  - *Physical*: physically secured against the *entry by intruders*.
  - *Network*: networking is *harmful* in terms of security.
  - *Operating System*: must decrease the *attack surface* and avoid *penetration*.
  - *Application*: may contain *security bugs*.

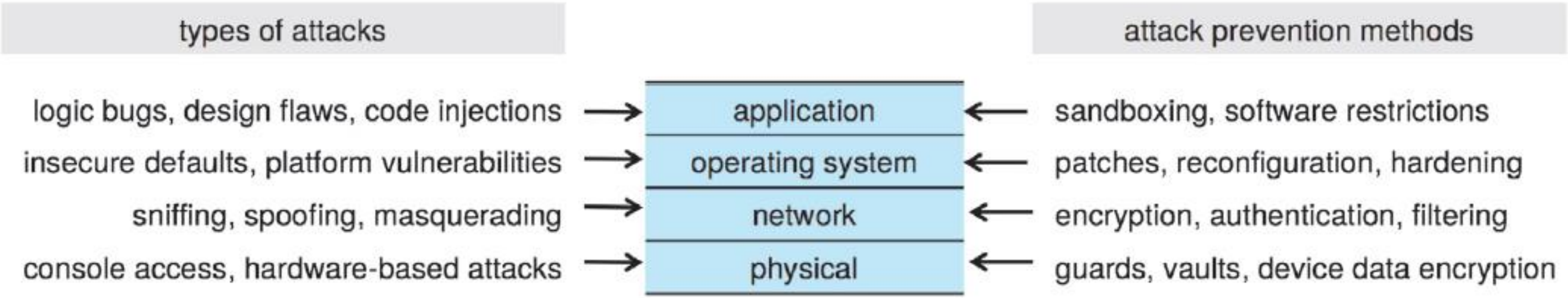


Figure 16.1 *The four-layered model of security.*

16.2 Program Threats

- *Security Holes* in *Programs* (or *Processes*)
  - *Malware* is
    - a software designed to exploit, disable or damage computer systems.
    - Trojan horse, spyware, ransomware, backdoor, or logic bomb.
  - *Code Injection*
    - Most software is not malicious, but nonetheless pose serious threats
    - due to a *code-injection* attack.
  - *Viruses* and *Worms*
    - A *virus* is a fragment of code embedded in a legitimate program.
    - *self-replicating* and are designed to *infect* other programs.
    - *Worms* use a network to *replicate* without any help from humans.

16.3 System and Network Threats

- *Threats* in the System and the Network:
  - Attacks via Network Traffic
    - *sniffing*: an attacker remains passive and *intercept* network traffic.
    - *spoofing*: either *masquerading* as one of the parties, or becoming a fully active *person-in-the-middle* intercepting and possibly modifying transactions between two peers.
  - *Denial of Service*:
    - does not aims to gain information or stealing resources
    - but rather to disrupt legitimate use of a system or facility.
  - *Port Scanning*:
    - is not itself an attack but is a means for detecting a vulnerability.
    - often part of a fingerprinting to deduce the type of the OS in use.

16.3 System and Network Threats

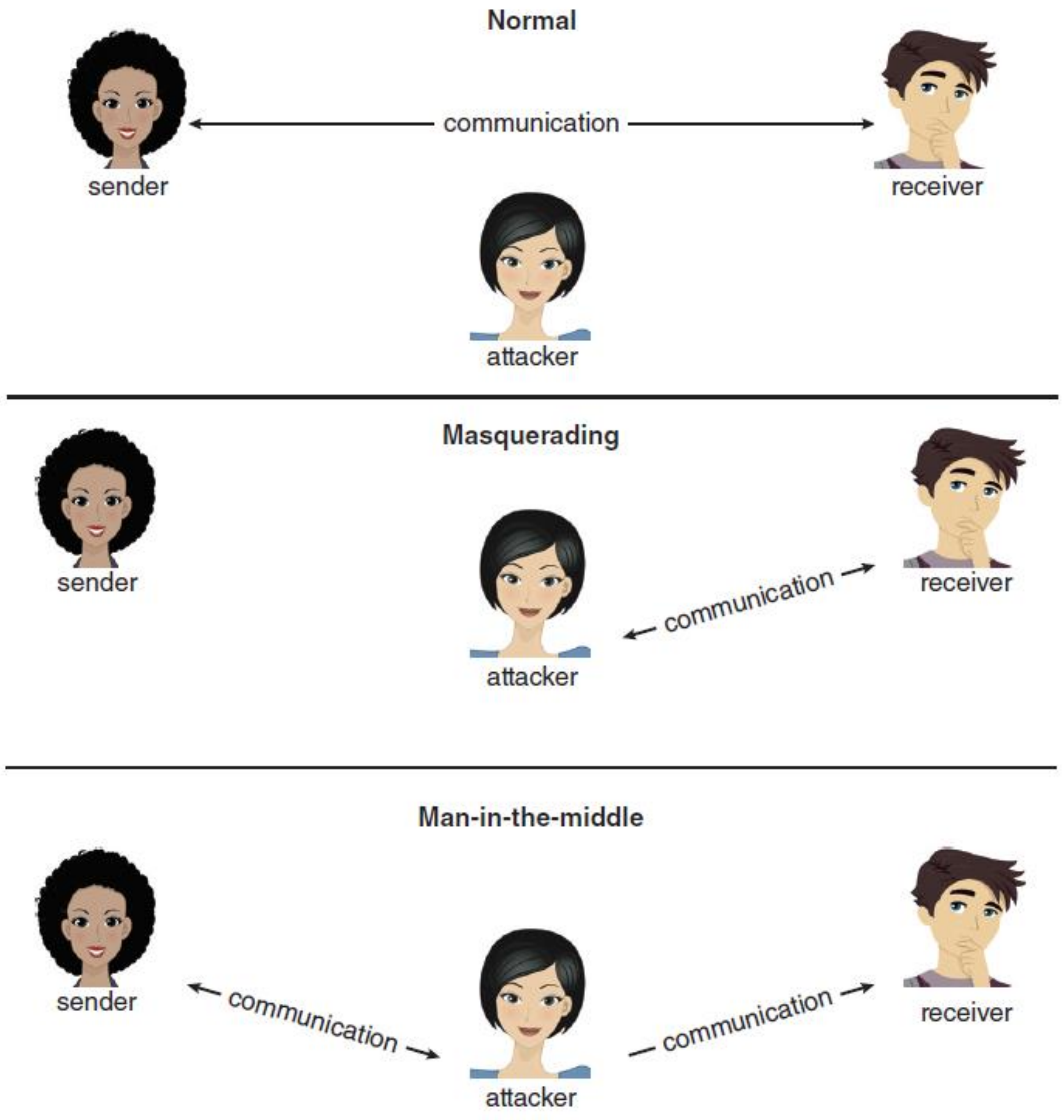


Figure 16.6 *Standard security attacks. (original source from Shutterstock)*

16.4 Cryptography as a Security Tool

- ***Cryptography***:
  - In a networked computer,
    - there exist potential *senders* and *receivers* of network *message*.
  - Then, how is the OS able to *trust* the message?
    - The job of *cryptography* is to eliminate *the need to trust the network*.
  - Based on secrets called ***keys***:
    - selectively distributed to computers in a network,
    - used to *process* (*encrypt or decrypt*) the messages.

16.4 Cryptography as a Security Tool

- ***Encryption***:
  - enables the sender to ensure that
    - only a receiver possessing a key can read the message.
  - The components of an encryption algorithm:
    - a set of *keys*: K.
    - a set of *messages*: M.
    - a set of *ciphertexts*: C.
    - an *encryption* function $E: K \to (M \to C)$.
    - a *decryption* function $D: K \to (C \to M)$.

16.4 Cryptography as a Security Tool

- The essential property of encryption algorithm:
  - Given a *ciphertext* $c \in C$, a computer can compute m,
    - such that $E_k(m) = c$, only if it possesses k.
  - Thus, a computer *holding k*
    - *can decrypt* ciphertexts to the plaintexts used to produce them,
  - but, a computer *not holding k*
    - *cannot decrypt* ciphertexts.
  - Note that the ciphertexts are generally exposed,
    - it is important that it be *infeasible to derive k* from the ciphertexts.

16.4 Cryptography as a Security Tool

- Two main types of encryption algorithms:
  - ***symmetric***:
    - the *same key* is used to encrypt and to decrypt.
    - therefore, the secret of k must be protected.
  - ***asymmetric***:
    - there are *different keys* to encrypt and to decrypt,
    - that is, the *public key* and the *private key*.

- ***Symmetric Encryption***:
  - In communication via symmetric encryption,
  - the key exchange can take place
    - directly between two parties
    - or via a trusted third party (to say, a certificate authority)
  - Commonly used symmetric encryption algorithms are
    - DES: Data Encryption Standard
    - AES: Advanced Encryption Standard

16.4 Cryptography as a Security Tool

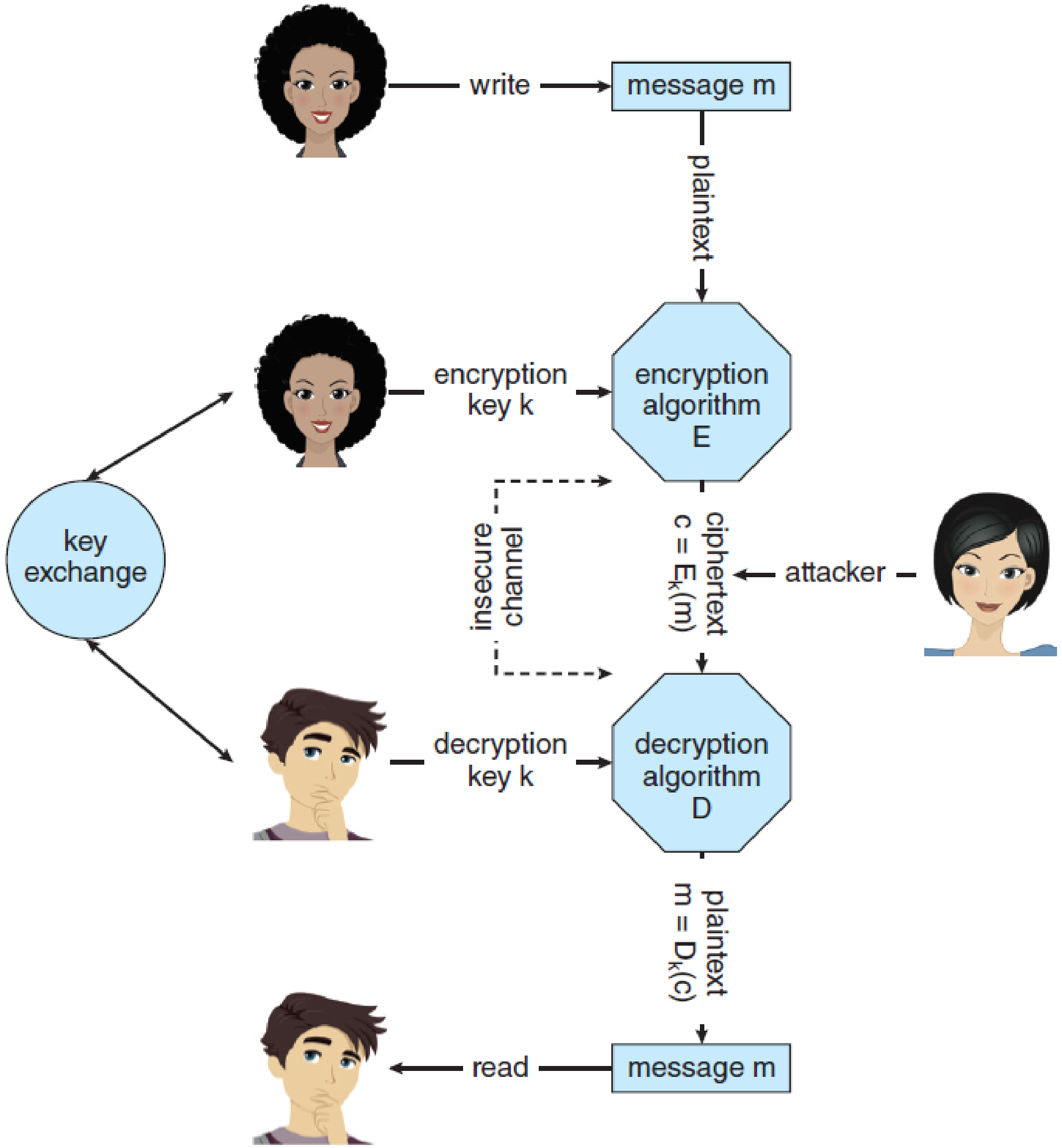
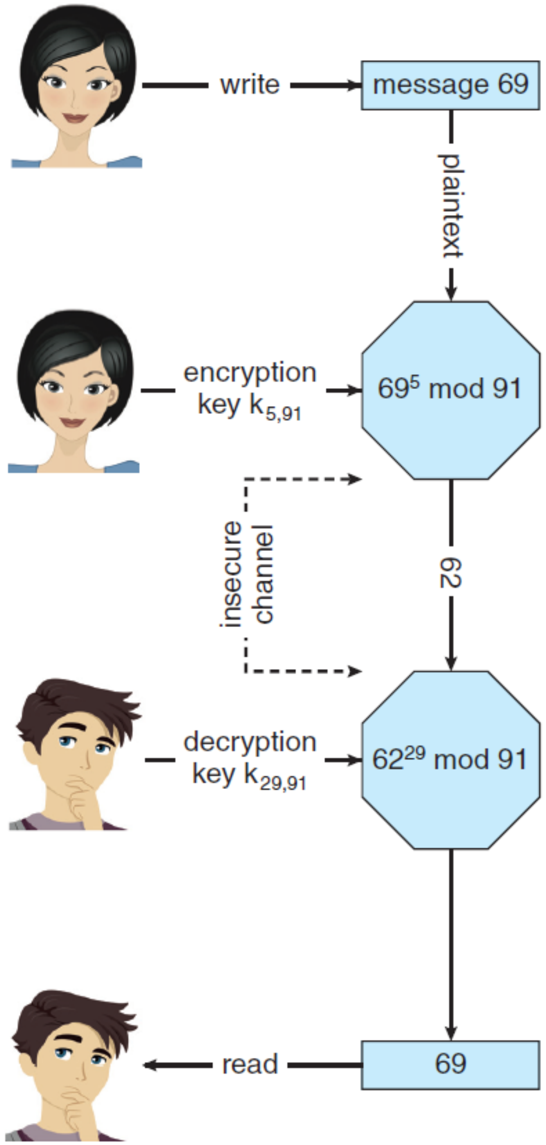


Figure 16.7 *A Secure communication over an insecure medium. (original source from Shutterstock)*

16.4 Cryptography as a Security Tool

- ***Asymmetric Encryption***:
  - a *breakthrough* in cryptography, first described by *Diffie and Hellman*.
    - *anyone* can *encrypt* a message to the receiving entity,
    - *only that entity* can *decrypt* the message, *no matter* who is *listening*.
  - The ***RSA*** algorithm: *Rivest, Shamir, and Adleman*.
    - the most widely used asymmetric encryption algorithm.



더 이상의 자세한 설명은 생략… 하지 않고,
주니온TV에서 다음 영상을 참고하도록 한다.

https://youtu.be/joYFr6y9uXE

Figure 16.8 *Encryption and decryption using RSA asymmetric cryptography. (original source from Shutterstock)*

16.4 Cryptography as a Security Tool

- ***Authentication***:
  - The *encryption* offers a way of
    - constraining the set of possible *receivers* of a message.
  - Whereas an *authentication* offers a way of
    - constraining the set of possible *senders* of a message.
  - An authentication is also useful
    - for providing that a message *has not been modified*.

16.4 Cryptography as a Security Tool

- Components of an authentication algorithm:
  - a set of *keys*: K.
  - a set of *messages*: M.
  - a set of *authenticators*: A.
  - a *secure(hash)* function $S: K \to (M \to A)$.
  - a *verification* function $V: K \to (M \times A \to \{true, false\})$.

16.4 Cryptography as a Security Tool

- The property of an authentication algorithm:
  - Given a *message* m,
    - a computer can generate an authenticator $a \in A$,
    - such that $V_k(m, a) = true$, only if it possesses k.
  - Thus, a computer *holding k*
    - *can generate authenticators* on messages
    - so that any computer possessing *k* can *verity* them.
  - however, a computer *not holding k*
    - *cannot generate authenticators on messages.*

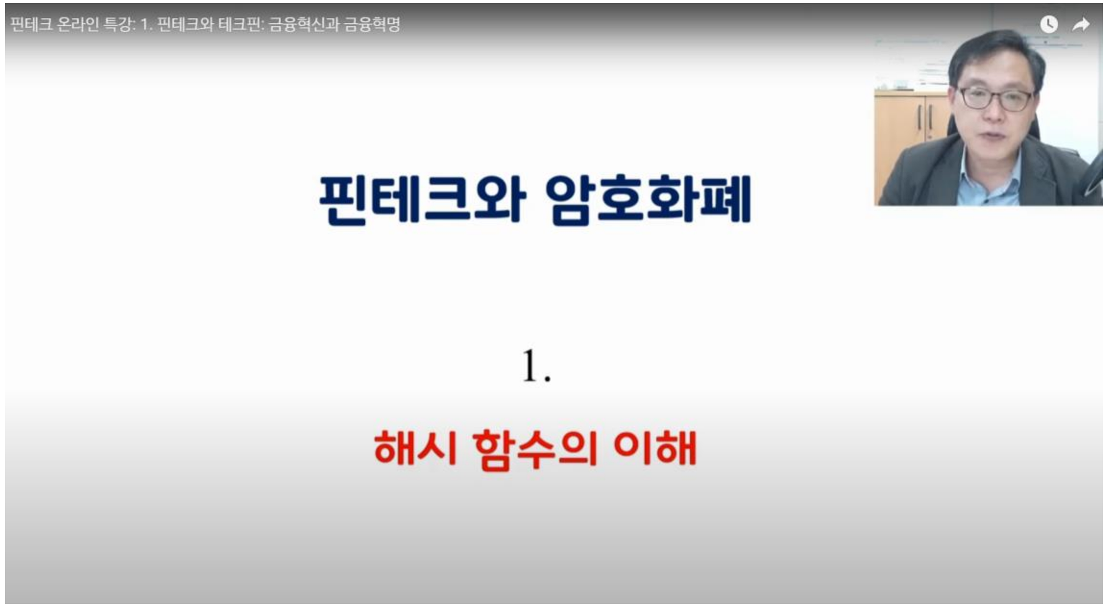
16.4 Cryptography as a Security Tool

- Two types of authentication algorithm:
  - A *hash function* $H(m)$ creates a small, *fixed-sized* block of data,
    - known as a *message digest* (or *hash value*), from a message m.
  - *MAC*: Message-Authentication Code
    - a cryptographic checksum is generated, using a secret key.
  - *Digital Signature Algorithm*:
    - produces a digital signature that enables anyone to verify the authenticity of the message.
  - Digital signatures are very useful,
    - since it is computationally *infeasible* to derive the private key from the public key. (to say, the *BitCoin* on the *BlockChain*)

16.4 Cryptography as a Security Tool

- 해시 함수가 비트코인에서 어떻게 사용되는지 알고싶다면,
- 주니온TV에서 다음 영상을 참고하도록 하자.

- 다 볼 필요는 없고, 37:55 에서 54:31 까지만 보면 된다.
- 고정 댓글에 좌표도 찍어 두었으니, 정성을 봐서라도….



https://youtu.be/RBvKQBQkYz

17.1 Goals of Protection

- ***Protection*** involves
  - controlling the (*authorized*) access of *processes* and *users*
    - to the resources defined by a computer system.
  - to increase the *reliability* of any complex system
    - that makes use of *shared resources* and is
    - connected to *insecure communication platforms*, such as the Internet.

17.2 Principles of Protection

- ***The Principle of Least Privilege.***
  - a key, time-tested guiding principle for protection.
  - Processes, users, and even systems should be given
    - just enough *privileges* to perform their tasks.
  - For instances,
    - UNIX user & file *privileges*: *root*, *sudo*, *chmod*.
    - *permissions:* gives a chance to *mitigate* the *malicious attacks*.

17.5 Access Matrix

- *Access Matrix*:
  - The general model of protection
    - can be viewed, abstractly, as an *access matrix*.

domain \ object	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Figure 17.5 *Access matrix.*

17.5 Access Matrix

- In an Access Matrix scheme:
  - the *rows* of the matrix represent *domains*
    - and the *columns* represents *objects*.
  - *ACL*: *Access Control List*
    - Each entry in the matrix consists of *a set of access rights*.
  - This scheme provides us with
    - the *mechanism* for specifying a variety of *policy*.

17.11 Other Protection Methods

- Sandboxing:
  - involves running processes
    - in environments that *limit what they can do*.
  - A process runs with the *credentials* of the user that started it
    - and has *access* to all things that *the user can access*.
  - For example,
    - *Java* and *.NET* impose sandboxing in the level of virtual machines.
    - *Android* enforces sandboxing as part of their mandatory access control.

17.11 Other Protection Methods

- Code Signing:
  - At a fundamental level,
    - how can a system ***trust*** a *program* or a *script*?
  - What if the code or the script is changed?
  - *Code signing*:
    - the *digital signing* of programs and executables
    - to confirm that they have not been changed since they were created.
  - Code signing is used
    - for *operating system* distribution, *patches*, and 3rd party tools.
    - also for *mobile apps* in the Android or iOS platforms.

The End:

- The Last Question:
  - *Do we need a brand new Operating System?*
  - Let us consider a computer system which is
    - totally different from the classical *von-Neuman architecture*.

1. ***Quantum Computers***
   - for processing the *Quantum Information*.
2. ***World Computers***
   - running on *the Block-Chain*, to say, the *Etherium* platform.

Any Questions?