Computer Security in the Real World

(Lampson 2004) talks about how despite many security successes, the actual realistic picture is one of a large number of unsecured systems. The reasons behind this are three-fold:

- **Cost** - True costs of attacks are often obscured due to embarrassments, as such corporations are unable to accurately determine cost-benefit.
- **Interconnectivity** - Anyone is able to attack a networked system.
- **User convenience**.

In addition to this, the only successful products - firewalls and antivirus systems - require minimal setup, and suggest the **bad configurations** lead to bad security, as much as anything else. Anything run on a bad configuration that is vulnerable should also be considered vulnerable. All of these thing require audits to be anything close to effective.

Real-world systems are not particularly secure. To break into physical locations it requires only minor determination. What stops large amounts of criminal activity is risk versus reward. Much the same thing happens with corporations; they rely on insurance and the legal system to recover and punish attacks, as opposed to taking preventative measures.

Computer security takes this a step further, by having connected systems. This means that attacks can come from any locale, and can often be daisy chained (via infections).

Security policies are based on four things:

- Secrecy (Confidentiality).
- Integrity.
- Availability.
- Accountability.

The reason security systems cannot be developed cohesively is that these four principles conflict, and different organizations favor different principles. Additionally, inside organizations user groups often want conflicting things.

When implementing these policies, there are two problems they are searching for:

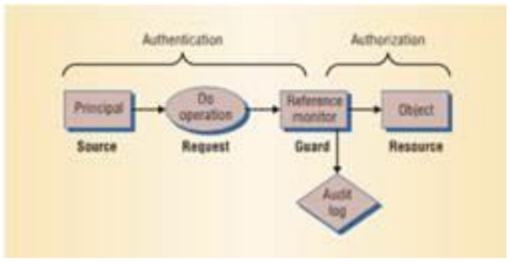
- Buggy or hostile code.
- Buggy or hostile agents - Both those that are giving bad instructions to programs, and those that spoof or tap into restricted systems.

There are five broad defense strategies that can be implemented:

- **Isolate** - Exclude all users.
- **Exclude** - Firewalls.
- **Restrict** - Sandboxing, such as permission settings on an Android phone.
  - **Honeypots** - Restrictive in the sense that humans have a tendency to focus in, and can easily miss over details.
- **Recover** - Backups/insurance.
- **Punish** - Audits/law enforcement.

The last two are not specifically computer security functions, and as such; corporations feel much more at home using these tools, than the first 3.

The **access-control model** outlines how these strategies work in concert:



The **guard** acts as the system that handles both authentication (identification), and authorization (permission checking). The guard should be separate from the object, in case the object gets corrupted. In some situations the guard should check that the object itself isn't nefarious (e.g. in cases of SQL injection). In this model, the guard is known as a **warden**.

The **information-control model** checks for corruption of the principle due to the object being delivered. This is often used in cases where malware is purposely infecting a system for research. **Taint control** wipes the machine every so often to remove any cases of corruption.

Solving these broader problems require a number of things. Firstly, configurations and systems need to be simplified for both users and administrators (users more so). Administrators need to deal with users uniformly, as not to get bogged down in the details. However, this can restrict user freedom, and cause additional problems. One way to simplify is to group users into **roles** that have a single set of privileges. **Access Control Lists** define both the roles and privileges.

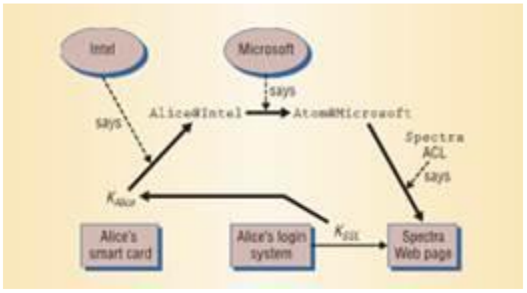
One prediction that is completely wrong, is the suggestion that type-safe systems eliminate many low level security flaws.

Trusted Computing Bases

The **star property** is used in highly secretive situations, where the central principle only has read access, and the outer principles only have write access. This form of **hierarchical security** allows for information to flow in, but not out.

A trusted computing base is a less strict version of the star property. Where the central principle has write access, and is designed to be a central point that should not be compromised. Lateral trust between levels of the hierarchy is also important.

Chains of Trust



A chain of trust is where "Principle P speaks for principle Q about T". This means that Q has delegated responsibility to P regarding T. This happens when a verifier states that this is acceptable. This means that P can talk to everyone in its network regarding Q, so that Q doesn't have to speak to each device individually.

The reasons this happens is that lots of direct contact requires safely exchanging keys.

Instead of verification of programs, sandboxing or virtual machining via **hypervisor** can be done when a verifier is not available - say with new programs.

Cryptography

Cryptography is the science of hiding information from untrusted parties. It predominately deals with the confidentiality portion of security, but can also be used for integrity. Cryptography is designed to protect against a number of malicious events:

- **Interception** - Attacker reads a message not meant for them.
- **Interruption** - Attacker prevents communication (e.g. HTTPs has to keep track of sessions, so opening many sessions locks up the system via DoS).
- **Modification** - Attacker modifies a message sent between two parties.
- **Duplication** - Attacker sends multiple copies of a message to a party.
- **Fabrication** - Attacker creates a message that appears to - falsely - be from one party.
    - **Impersonation** - Attacker pretends to be a legitimist party.
- **Stegocommunication** - The two parties communicating are hostile.
- **Repudiation** - One party says they did not send or receive a message.

There are two broad ways that cryptography secures systems:

- **Symmetric Encryption** - The decryption and encryption keys are either the same, or mathematically related, so that knowing one allows a person to determine the other.
- **Asymmetric Encryption** - The decryption and encryption keys cannot be determined from one another without large computational effort. This is often the result of un-invertible functions, such as hashing.

Ciphertext may be broken in a number of ways:

- Discovering the underlying "secure" algorithm, if in fact the system in question is not a real cryptosystem, but instead masquerading as one.
- Social Engineering.
- Brute-force on small key-spaces.
- Using plaintext/ciphertext pairings to determine what ciphertext results in some action. This problem is inherent when keys are reused, and in general keys will cryptographically weaken over time - hence the need for expiry.

Public Key Cryptography

Public key (PK) cryptography is a form of asymmetric communication; where either the encryption or decryption keys are freely given to the public known, as the public key (p), and the other key is kept private (p):

- **Public encryption key/private decryption key** - The public can encrypt and send you messages that only you can decrypt.
    - $\{P\}_A$ - Plaintext *P* encrypted with private key *A*.
- **Private encryption key/public decryption key** - The public can decrypt your messages - this verifies they are from you personally.
    - $[P]_A$ - Plaintext *P* signed by the private key *A*.

Giving out both keys results in a completely insecure system, and is party why this system isn't used by every user, it can be quite complicated.

Public Key Infrastructure

The **Public Key Infrastructure** (PKI) is a set of systems in place to manage and distribute public keys. They are mutually trusted principles that certify that public keys are from specific individuals. However, they have a number of problems:

- It is hard to fully verify the identity of the public key holder.
- It is often hard to determine which person holds which key, due to such things as many people having the same name.
- Users may not actually inspect certificates, rendering the whole system useless. This is especially important in terms of **man-in-the-middle attacks**. In this situation if a user isn't paying attention to the certificate the malicious party can use certificates not related to either party.

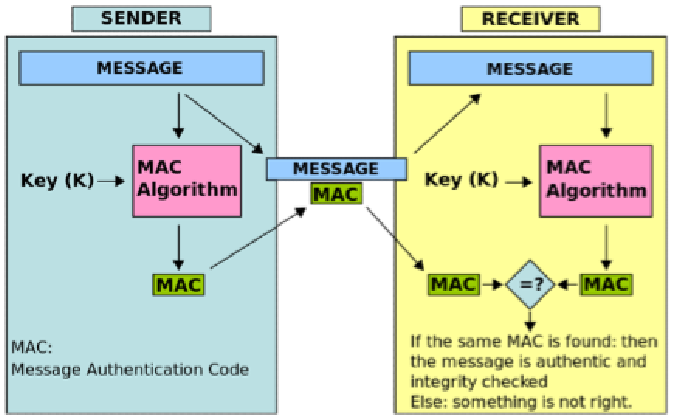
When public key infrastructure is down, the browser will revert to using any keys it has cached. However, this results in the possibly of using expired keys that may have expired due to malicious use.

Key Continuity Management is an alternative to CAs. In this system, parties decide to accept initial keys, and these are then used for further communication. Problems arise in this system due to:

- Difficultly making sure introductions are trustworthy.
- Difficultly with expiring keys.

Message Integrity

One function of cryptography is the assurance of integrity. Message Authentication Codes (MAC) use a shared secret key to hash a message. The message is then sent with its resulting hash. If the receiving party notices that the hash they receive from doing their own computation is different, they can be reasonable sure tampering has taken place.


SENDER
MESSAGE
Key (K)
MAC Algorithm
MAC
MAC:
Message Authentication Code
MESSAGE
MAC
RECEIVER
MESSAGE
Key (K)
MAC Algorithm
MAC
=?
MAC
If the same MAC is found: then the message is authentic and integrity checked
Else: something is not right.

Another function of MAC encoding is to check for file degradation either during downloading (via an MD5 hash), or during normal processes of storage (e.g. bit-rot).