4. Human Weakness, Physical Security and Hashes

COMP6441 • KC Notes

4.1 Human Weakness: Problem

- **Humans are the weakest part to security**
  - **Greed**: corruption in police, bank tellers, abuse of trust and power
  - **Fear, emotion**: humans act and think irrationally
  - **Laziness**: humans do not like repetition, and routine checks may just be ticked off
  - **Pride, anger, curiosity, ignorance, overload of information**
  - Compounded by normalised behaviour – "this was always how it has been"
- **Costa Concordia** disaster and **South Korea's Sewol ferry** disaster
  - In the former, the captain of the boat left first and didn't think to evacuate the passengers first
  - In the later, the captain told everyone to stay on the boat
- Elaborate setups that are only **security theatre** – only *looks* secure
- Other disasters with systematic failures that need to be stopped, (e.g. child abuse, refugee and detention centre conditions)

4.2 Human Weakness: Response

- The response to human weakness is **training and drilling**
  - **Rick Rescoria** found the evacuation procedures for the World Trade Centre inadequate – trained and drilled his company's employees on evacuation
  - When the plane crashed, he evacuated and orchestrated the evacuation of around 2,000 people
- Similarly, training is needed for people to **act rationally when security is exploited**
  - Train people to stop tailgaters
  - Magicians and how they create distractions and trick you psychologically

4.3 Physical Security

- Having a secure communication protocol is useless **without protecting physical access**
  - Latest CIA leaks targeted physical access, e.g. televisions, optic fibres
  - Other physical access including stealing, key logging, microphones
- **Tamper-proof vs tamper-evident** – prevent tampering or know of tampering
  - E.g. ballot boxes with security tags need to be tamper-evident
  - ATMs need to be tamper-proof to prevent access to ports

4.4 Hashing

- **Hashing**: ensuring that a **message has integrity** (has not been tampered with) and this follows with **authentication** (message comes from owner)
    - Prevent a **man in the middle attack**, where someone could **change or replay** a message
    - Example: poker machine where a light beam reads the number of coins falling, but could be tampered with by covering up the light beam
- **Nonce**: a **number used once** that **prevents replay attacks**, e.g. the time of day
    - Time of day requires **confidentiality** – an alternative is a variable size or small fixed length appended to the string
- **Cryptographic hashing**:
    1. Sender and receiver decide on a secret, and sender appends secret to his message m.
    2. Sender hashes his plaintext secret and message and **sends the plaintext message m and hash h(m).**
    3. The receiver can **confirm by appending the hash to the plaintext and comparing hashes**.
    - Cryptographic hashes must be easy to go from m to h(m), but very difficult to go from h(m) to m
        - Passwords can be stored as hashes and you can compare hashes to verify user
- Attacks:
    - **Pre-image attack**: if given the hash h(m), you find the message m
    - **Birthday/collision attack**: if you find two messages m_1 and m_2 that have the **same hash h(m)**
        - The **birthday paradox** (not actually a paradox but counterintuitive) – as long as there are ~24 people, there is more than 50% chance that **at least two people share the same birthday** (number of pairs grow quadratically)
    - **Second pre-image attack**: given both the message m_1 and hash h(m), **you find an m_2 with the same h(m)**
        - Different from collision attack as **you are given more information**
- Current hashing algorithms:
    - MD5 – too small, easy to brute force. Not collision or pre-image resistant
    - SHA0, SHA1, SHA2 – all developed with the NSA, the first two considered **broken**
    - SHA3 not developed with the NSA
- **Broken**: once a hashing algorithm can be attacked faster than brute force.
- **Length Extension Attack**: because most hashes are iterative and take part by part to hash, so you could add to the end of a hash
    - **HMAC** solves this by applying **hash(k || hash(key || message))**