COMP0141

Security

This course is about "understanding information security: how attacks work, how they can be prevented, and the importance of 'thinking about the human'… and above all, how to think with a security mindset".

1 What is Security?

Safety protects against unintentional harm, while *security* protects against intentional threats.

In *CS* security,
- correctness,
- safety, and
- robustness

must hold even against a powerful adversary.

The *Bruce Schneier* Security Mindset A habitual practice of thinking how a system can be subverted or broken instead of just how it *normally* works fine; thinking like an attacker. One will never notice most security problems without thinking like this.

1.1 Security Properties

The **CIA Triad** are 3 core properties in security:
Confidentiality data being inaccessible by the unauthorized
Integrity data being correct (not altered by the unauthorized) over its lifespan
Availability data being accessible whenever required

Other properties beyond CIA:
Privacy (note: confidentiality belongs to data, while privacy belongs to individuals)
Authenticity (note: integrity pertains to the data itself, while authenticity pertains to its source)
Anonymity (note: privacy is about hiding the action, while anonymity is about hiding the author)
Non-repudiation individuals being unable to deny any messages they've sent
Plausible deniability ability to credibly deny actions (a stronger form of the negation of non-repudiation)
Forward secrecy compromise of long-term keys does not compromise past session keys (and so past traffic)

1.2 Security Definitions

To *design* a secure system, one first needs to *define* the criteria for security.
Two philosophies:
Binary A system is either secure or insecure. Cryptography uses a binary model.
Risk management A system is secure to some acceptable level

Binary definitions are hard to get right (and consequences can be tremendous). Risk management, however, often presents a lot of trade-offs (e.g. between availability and integrity).

Cost-wise, risk-management is usually more suitable than a binary system.

1.3 Threat Modeling

Threats, *vulnerabilities*, *likelihood*, *impact*, and *cost* are used to create a *threat model*.

Systems are not just *secure*, but secure under *some* threat model.
A **system is secure** if an adversary, constrained by some given threat model, cannot violate the security policy.

1.3.1 Threats

An adversary has access to a set of *resources* to:
- observe (sniff packets)
- disrupt (jam radio comms)
- influence (social engineering)
- modify (alter packets in transit)
- control (influence trusted users)

A *strategic* adversary uses resources *optimally* (to maximize their objective subject to constraints).

STRIDE is a model to analyze threats:

Spoofing impersonation (Authenticity)
Tampering modification of data (Integrity)
Repudiation claiming (whether honestly or not) a lack of responsibility (Non-repudiation)
Information disclosure unauthorized access (Confidentiality)
Denial of service exhausting / overwhelming a server's ability to provide service (Availability)
Elevation of privilege gaining higher access privileges than authorized (Authentication and Access Control)

1.3.2 Vulnerabilities

Vulnerabilities are what *enable* a threat. Adversaries *exploit* vulnerabilities to cause harm.

1.3.3 Likelihood

The likelihood of failure. This can vary depending on the state of or input to the system. It also depends on who the adversary is (their expertise and motivations).

1.3.4 Impact

The consequences in the event of a successful attack. Once again, depends on many state variables and the extent of the attack itself.

1.3.5 Protection

The initiatives taken to achieve security. Always has some *cost* and some finite effectiveness.

1.4 Human-centered Security

One has to consider **goals**: why are people using this system?

Usability How well a (specific) user (in a specific context) can use a product to achieve a *defined goal* effectively, efficiently, and satisfactorily.

Time is a major hidden cost.

Of course, usability and security usually come at a trade-off against each other.

Usability can be evaluated by
- cognitive walkthrough by usability experts
- user studies
- telemetry

Common problems and ways to improve:
- Users lack intuition: provide education and training
- No one to maintain the security of personal devices: make security invisible
- Difficulty in estimating risks: education
- Users avoid secure methods due to complexity: make security the path of least resistance

2 Design Principles & Background

2.1 Design Principles

General considerations when designing secure systems.

The following 8 principles spell out ELLF COPS.

Economy of Mechanism Keep designs as simple and small as possible. Simpler = easier to understand, less likely to contain flaws.

Least Privilege When configuring access control, provide each user with the minimum access required for them to do their job.

Least Common Mechanism Minimize mechanisms (state, services, channels) shared across users or security domains. Shared mechanisms are inherently trusted by all parties using them and are prone to leaking covert channels.

Fail-safe Defaults The "default" state (e.g. during unforeseen errors) should not violate security principles. E.g., using a white-list instead of a black-list for access control.

Complete Mediation Every access to every object should be checked for authority.

Open Design A system's security should not depend on an adversary's ignorance of its design. (security by obscurity)

Psychological Acceptability A system's secure mechanism should be at least as easy as not using it. (path of least resistance)

Separation of Privilege No single condition (or single party) should suffice to grant access; require multiple independent checks.

Additional principles:

Defense in Depth Not allowing a single vulnerability to compromise the entire system.

Design for Updating Vulnerabilities will always be discovered, so make your system updatable.

Prudent Paranoia Don't underestimate the effort adversaries will go to. "Just because you're paranoid doesn't mean they aren't after you."

Privacy Promotion Treat user privacy as a first-class design goal, not a side-effect of confidentiality. Collect the minimum data necessary, retain it for the minimum time, and prefer techniques (e.g. local processing, end-to-end encryption) that keep data out of operator hands.

2.2 Networking

The TCP/IP stack (4 layers, bottom-up):

Link physical / local-segment frames (Ethernet, 802.11). MAC addresses.

Network end-to-end packet routing (IP). IP addresses; routers forward by longest-prefix match.

Transport connection abstractions. TCP is reliable, ordered, handshake-based; UDP is best-effort and stateless.

Application everything else (HTTP, DNS, BGP, …).

DNS maps human-readable domain names to IP addresses via a hierarchical, distributed lookup (root → TLD → authoritative). Mostly UDP/53; cached aggressively. Trust-based — classic DNS has no integrity (cf. DNSSEC, DoH).

Packets each layer wraps the layer above with its own header. Hosts care about all four; intermediate routers normally inspect only up to Network.

Routing between networks (autonomous systems) by BGP, an Application-layer protocol over TCP. Each router announces reachability for prefixes; specificity (longer prefix) wins.

2.3 Math

(not sure why this is in this section)

$x \mid y$ means x is a factor of y (think: x "goes into" y).

$(\exists a \in \mathbb{Z} : z = ax + y) \iff z \equiv y \pmod{x}$
$z \equiv y \pmod{x} \iff x \mid (z - y)$

$\gcd(a, b) = \max(d : d \mid a \text{ and } d \mid b)$

2.3.1 Euclidean Algorithm

Efficiently calculate $\gcd(a, b)$.

```
def gcd(a, b):
  while b:
    a, b = b, a % b
  return a
```

2.3.2 Extended Euclidean Algorithm

Bezout's identity $\forall a, b \in \mathbb{Z} : \exists x, y \in \mathbb{Z} : \gcd(a, b) = ax + by$

Efficiently calculate x, y s.t. $\gcd(a, b) = ax + by$

```
def egcd(a, b):
  x, x1 = 1, 0
  y, y1 = 0, 1
  while b:
    q, r = divmod(a, b)
    a, b = b, r
    x, x1 = x1, x - q * x1
    y, y1 = y1, y - q * y1
  return a, x, y
```

2.3.3 Modular Arithmetic

All ring axioms (associativity, commutativity, distributivity) are preserved for $+$ and $\times$.
Exponentiation works in the exponent *modulo* $\varphi(n)$, not modulo n: when $\gcd(a, n) = 1$, $a^x \equiv a^{x \bmod \varphi(n)} \pmod{n}$ (Euler's theorem).

For further stuff regarding **commutative rings** and **modulo addition groups** or **modulo multiplication rings**, see https://akioweh.com/shared/COMP0147#modular-arithmetic.

Multiplicative Inverse a^{-1} is an inverse of a under some modulo n iff. $a^{-1} \times a \equiv 1 \pmod{n}$.

An element x of some $\mathbb{Z}/n\mathbb{Z}$ has a multiplicative inverse iff. $\gcd(x, n) = 1$ (i.e., x and n are coprime).
If $\gcd(a, n) = 1$, then $ax + ny = 1$ by Bezout's identity, and so $ax \equiv 1 \pmod{n}$, giving x as the inverse of a.
The proof in the other direction is literally the same thing reversed.

Multiplicative inverses are unique. Evidently, the extended Euclidean algorithm computes them efficiently.

A *field* is an abelian group under addition whose *nonzero* elements form an abelian group under multiplication, with multiplication distributing over addition. (0 has no multiplicative inverse.)

Prime Order Finite Field $\mathbb{Z}/p\mathbb{Z}$ where p is prime. (Note: there are also finite fields of prime-*power* order p^k for $k > 1$, but those are *not* $\mathbb{Z}/p^k\mathbb{Z}$ — that latter ring is not even a field.)

3 Confidentiality

3.1 Cryptography

3.1.1 Math

The security of a cryptography system is always tied to some *hard* problem, e.g. the discrete logarithm.

Important algebraic structures:
- large prime order finite fields $F_p \quad (p \geq 2^{1024})$;
- multiplicative rings modulo large $n = pq \quad \mathbb{Z}/pq\mathbb{Z} \quad (p, q \geq 2^{1024}$ and are primes).

Finite fields' value in encryption comes from the discrete log problem; multiplicative rings' value come from the factoring problem.
Discrete log is also hard in composite multiplicative rings (in fact, at least as hard as factoring), but prime fields admit cleaner subgroup structure for DL-based protocols.

Discrete Logarithm Given b, y, and prime p, find x s.t. $b^x \equiv y \pmod{p}$.

Factoring Problem Given $N = pq$ (different odd primes), find p and q.

RSA Problem (Rivest, Shamir, Adleman) Given N, e, and y, find b s.t. $b^e \equiv y \pmod{N}$. ($N = pq$, different odd primes.) *Believed* to be as hard as the factoring problem.

One-Way Functions A function that is easy to compute, but (believed to be) very difficult to invert or find two inputs with the same output. Modular exponentiation (whose inverse is the discrete log *problem*) and integer multiplication of primes (whose inverse is the factoring *problem*) are candidate one-way functions.

3.1.2 Classic Ciphers

Most classic ciphers are *substitution* ciphers: each plaintext letter is replaced by another according to some rule. They split into *monoalphabetic* (one fixed substitution) and *polyalphabetic* (the substitution varies by position).

3.1.2.1 Monoalphabetic Substitution

A single permutation $\sigma \in S_n$ (over an alphabet of size n) is applied to every letter. Key space is $n!$ but trivially broken by frequency analysis on any non-trivial ciphertext.

Caesar shift cipher special case where σ is a cyclic shift. Key is a single element of $\mathbb{Z}/n$; only n possible keys (brute-forceable).

3.1.2.2 Polyalphabetic Substitution

The substitution varies by position; the parent class of Vigenère, running-key, and Enigma. Defeats simple frequency analysis on the ciphertext as a whole, but periodic schemes leak structure once the period is found (Kasiski examination, index of coincidence).

Vigenere Cipher rotates through a fixed-length sequence of k independent Caesar shifts. Key is in $(\mathbb{Z}/n)^k$. Periodic, so vulnerable once k is recovered.

Running Key Cipher Vigenère with the key drawn from a long, structured source (e.g. a book). No fixed period, but the key is non-random. Statistical attacks on plaintext-on-plaintext still apply.

Enigma rotor machine implementing a position-varying substitution with very long effective period; broken via known-plaintext (cribs) and operational mistakes, not a flaw in the substitution principle.

3.1.2.3 Beyond Substitution

One-time Pad Each plaintext letter is combined with a corresponding key letter via modular addition. Requires a secret key at least as long as the plaintext. Information-theoretically unbreakable if the key is truly random, secret, and used only once. (Vigenère with a truly random key as long as the message *is* a one-time pad.)

3.2 Symmetric Encryption

- stream ciphers: a key + nonce/IV generates a keystream that XORs with arbitrary-length plaintext.
- block ciphers: a primitive that encrypts fixed-size blocks under one key. Encrypting longer messages requires a *mode of operation* (CBC, CTR, GCM, ...); some modes pad, others (e.g. CTR) don't.

AES (Advanced Encryption Standard, formerly Rijndael) the de-facto symmetric block cipher. 128-bit blocks; 128 / 192 / 256-bit keys with 10 / 12 / 14 rounds respectively. Each round (except the last) is SubBytes (non-linear S-box over $\mathrm{GF}(2^8)$) → ShiftRows → MixColumns → AddRoundKey; round keys come from a key schedule. Has dedicated CPU instructions (AES-NI) on modern hardware. Security is unproven but well-scrutinized.

3.2.1 Key Exchange

The Diffie-Hellman (DH) key exchange protocol allows two people to create a shared secret key over an insecure communication channel without ever sending the key itself.

1. **Public parameters**
   Alice and Bob agree on prime p and a generator g of a large prime-order subgroup of $(\mathbb{Z}/p)^*$. (Using a full-group generator leaks one bit of each secret via the Legendre symbol.)
2. **Private Keys**
   Alice picks secret a.
   Bob picks secret b.
3. **Public Keys**
   Alice computes $A = g^a \bmod p$ and sends it to Bob.
   Bob computes $B = g^b \bmod p$ and sends it to Alice.
4. **Shared Secret**
   Alice computes $S = B^a \bmod p$.
   Bob computes $S = A^b \bmod p$.
   Both results are identical due to, well, math.

This is secure as 1 the discrete logarithm problem makes it hard to deduce a or b from A and B, and 2 the *Computational Diffie-Hellman* (CDH) assumption makes it hard to compute S from A and B alone. (CDH is implied by DL but not proven equivalent.)

Note that DH does not *authenticate* the parties and is thus vulnerable to man-in-the-middle (MitM) attacks. An attacker could sit in the middle, pretend to be Bob to Alice and Alice to Bob, establishing two separate secrets. This allows eavesdropping (and integrity violations) invisible to Bob and Alice.

3.3 Asymmetric Encryption

Asymmetric encryption (aka. public-key cryptography) mainly solves logistical constraints of symmetric encryption.
A problem of symmetric encryption arises in many-to-many communication setups: the number of keys in the system scales quadratically.

Encryption Oracle A black-box entity that encrypts any plaintext on demand.

Decryption Oracle A black-box entity that decrypts any ciphertext on demand.

Chosen Plaintext Attack (CPA) Adversaries that have access to an encryption oracle. E.g., influencing the message of a sender and then snooping the ciphertext in transit.

Chosen Ciphertext Attack (CCA) Adversaries that have access to a decryption oracle.

Indistinguishability from CPA (IND-CPA security) A security property where CPAs do not allow an adversary to distinguish between the ciphertexts of one chosen plaintext from another.

Indistinguishability from CCA (IND-CCA security) A security property where CCAs do not allow an adversary to distinguish between the ciphertexts of one chosen plaintext from another. This is strictly stronger than IND-CPA.

RSA:
- choose 2 large primes p and q
- let $N = pq$
- choose Public Key e s.t. $\gcd(e, \varphi(N)) = 1$ ($\varphi(N) = (p-1)(q-1)$)
- compute Private Key d s.t. $e \times d \equiv 1 \pmod{\varphi(N)}$ (the inverse of e). (Modern implementations use Carmichael's $\lambda(N) = \mathrm{lcm}(p-1, q-1)$ instead of $\varphi(N)$, yielding a smaller d; both work.)

(N, e) is the public key; (N, d) is the private key.

$\mathrm{Enc}(M) = M^e \bmod N$
$\mathrm{Dec}(C) = C^d \bmod N$

NOTE: "textbook" RSA encryption (above) is *not* IND-CPA secure: it's deterministic (same $M \Longrightarrow$ same C) and malleable. Real schemes randomize via padding (OAEP).

3.3.1 random details

One-way functions' existence depend on P ≠ NP, which is *unproved*. (P ≠ NP is necessary but not sufficient: OWFs need average-case hardness, not just worst-case.)
Even many symmetric ciphers that do not depend on one-way functions, like AES, are also *unproved* regarding their security. In practice, the agreement of an algorithm's security is purely social (based on the lack of vulnerabilities found after "significant" scrutiny)

Positive indicators (e.g. padlock icon when secure) are easily ignored or spoofed; *negative* indicators (warn when security is *absent*) are preferred — users react to alerts, not the absence of them.

4 Integrity

...the state of being unaltered (by the unauthorized).

Digital signatures provide authenticity (and integrity), addressing the MitM vulnerability identified earlier in DH key exchange.

Authenticated Encryption encryption + signature / message authentication checks to achieve both confidentiality and integrity (+ authenticity).

4.1 Digital Signatures

For RSA specifically, the setup mirrors asymmetric encryption with the "private" and "public" keys swapped: one uses the private key to compute the *signature* of a message by treating the message as a ciphertext and "decrypting" it, and then anyone can use the public key to re-"encrypt" the signature and check that it matches the original message. (This key-swap intuition is RSA-specific; ECDSA, EdDSA, and hash-based signatures don't follow this pattern.)

For RSA, follow the same keygen procedure, then
(N, e) is (still) the public key; (N, d) is the signing (private) key.

$\text{Signature}(M) = M^d \bmod N$
$\text{Verify}(M, \sigma)$: check that $\sigma^e \equiv M \bmod N$

Existential Unforgeability under Chosen Message Attack (EUF-CMA security) An adversary cannot produce a valid signature for *any* new message, even after adaptively obtaining signatures on messages of their choice from a signing oracle.

NOTE: "textbook" RSA (what we have here) signatures are *not* EUF-CMA. Forgery without oracle: pick σ, set $M = \sigma^e \bmod N$ and the (M, σ) pair is valid (although no control over M's content). Forgery with oracle: RSA is multiplicatively homomorphic, so $\sigma_1\sigma_2$ signs M_1M_2 for any two pairs of message-signatures. Real schemes sign $\text{pad}(\text{hash}(M))$, not raw M.

4.2 Message Authentication Codes

Like signatures, but uses shared-key primitives (e.g. HMAC, CMAC). Verification requires the same key, so MACs prove integrity + authenticity to anyone holding the key but cannot establish non-repudiation.

4.3 Hash Functions

Cryptographic Hash Function A hash function with pre-image resistance, second pre-image resistance, and collision resistance. Uniformity and the avalanche effect are typically also required to achieve the resistances.

Avalanche Effect ...where even small changes in input yield large changes in output.

Uniformity The property where the output distribution is uniform against random inputs.

Pre-image Resistance Given y, it is hard to find x s.t. $H(x) = y$.

Second Pre-image Resistance Given x, it is hard to find $x' \neq x$ s.t. $H(x) = H(x')$.

Collision Resistance It is hard to find any non-equal pair x, x' s.t. $H(x) = H(x')$.

Strong avalanche effects typically imply good uniformity...

Applications of hash functions:
- checksums
- message authentication codes
- digital signatures
- blockchains
- password storage

$\text{HMAC}(K, M) = H\Big((K \oplus \text{opad}) \parallel H\big((K \oplus \text{ipad}) \parallel M\big)\Big)$

4.4 Mapping to STRIDE

Keyed hashes (MACs) or signatures could prevent Spoofing, Tampering, Repudiation, and Elevation of privilege (in STRIDE). (Unkeyed hashes alone don't authenticate; an attacker can recompute them.) Encryption (in addition to powering signatures) could prevent Information disclosure and Elevation of privilege (in STRIDE).

4.5 Digital Certificates

X.509 certificates are used to prove the ownership of a public key. To communicate with a given domain, you need its public key. X.509 is important to ensure any public key you receive is actually associated with the claimed domain and did not come from an impersonator (Authenticity and MitM).

5 Human-centered Security

5.1 Thinking Socio-Technically

To build truly robust security systems, one must avoid viewing technical and social aspects in isolation. A *socio-technical* approach combines

- technical elements: authentication, encryption, identity management systems, etc.
- social elements: people, workplace culture, usage context, user limitations…

Security is a process, not a product.

- Design *with* humans, not just *for* humans: understand user needs and limitations through testing, feedback, and observation.
- Understanding interactions: how people interact with technology "in the wild" is often very different from various idealized constructs
- Managing constraints: the user can*not* do a lot of things, like remembering a long random password that changes every week

A critique of human-centered design is that while it improves systems for those involved in the design process, it may inadvertently make it worse for those who are not.

5.2 Humans are (not) the Weakest Link

A "myth" is that humans are the *weakest link* in security. However, research shows that users do *care* about security and intend on doing the right thing. Instead, users fail due to:

- a lack of awareness (of threats)
- not being told correct and *actionable* mitigation strategies (despite threat awareness)
- the system having low usability, forcing *shadow security* practices (e.g. writing passwords down)

Primary vs. Secondary Tasks Security is almost never a user's *primary* task; their goal is to complete their work (e.g. messaging friends, accessing the web), rendering security a *secondary* task that often even becomes a *barrier* to the primary goal.

5.3 Users vs. Management

- Users are not the enemy:
  Security departments often view users as a liability. They should instead be treated like *partners*; security policies must be compatible with *existing* practices; otherwise, they will be circumvented as users need to continue performing their primary tasks.
- Security managers are not the enemy:
  Security managers generally also do (intend to) consider user needs. However, they're often inhibited by restrictive organizational structures; mutual distrust develop between users and IT departments, making policies ineffective and unenforceable.
- Security Policies:
  Policies are often developed in isolation without consulting the users.

5.4 Designing Usable Systems

A system that lacks usability is fundamentally insecure as users will misuse or circumvent it.

The 5Es of Usability (from Whitney Quesenbery) Effectiveness, Efficiency, Engagement, Error Tolerance (allowing users to recover from mistakes), and Ease of Learning.

NOTE that the slides incorrectly attributes these to the Nielsen Norman Group…

Affordances & Signifiers An *affordance* is a relationship between an object and the actions it permits (a door affords pulling); a *signifier* is the perceptible cue that communicates the affordance (the pull handle). Systems should make secure actions both afforded and signified.

Case studies:

- Access-controlled doors: a standard swing door with a badge reader is technically secure, but socially insecure (coworkers will politely hold the door open for each other, violating policy). A physical turnstile that only allows one person at a time designs *around* the social limitation.
- Encryption: PGP 5.0 was technically brilliant but very unusable, causing users to expose keys or send plain text by accident. In contrast, WhatsApp integrated end-to-end encryption transparently, leading to mass adoption of this secure system.
- Passwords: professionals want complex, unique, expiring passwords. Users want to log in and do their work. Forcing such high complexity ends up with users doing worse things (*shadow security practices*) like writing passwords on sticky notes.

5.5 Human Limitations & Cognitive Biases

Memory limitation:

- short-term / working memory: 4 chunks at a time (Cowan, 2001; Miller's classic "7 ± 2" is now considered an over-estimate)
- long-term memory: relies on habits and requires periodic refreshers

Cognitive Biases:

Optimism Bias "This won't ever happen to *me*."

Anchoring Bias The first piece of information disproportionately shapes subsequent estimates. E.g., the first quoted "risk level" frames how seriously a user takes everything that follows.

Status Quo Bias "I did it before, so I'll do it again." Users stick with prior choices and habits.

Consensus Bias "Everyone probably reuses passwords like I do, so it's fine." (false-consensus: assuming others share your behaviors)

Hyperbolic Time Discounting "I'll worry about security tomorrow; I have to finish my work today."

Attention and Inattentional Blindness:

- Users experience "security fatigue". Frequent warnings or false positives cause users to become desensitized and ignore them.

Inattentional Blindness One only perceives what one focuses on. ("Gorillas in our midst" experiment.) Users often do not notice security red-flags because they never focused on security to begin with (but rather the work they are doing).

5.6 Security Awareness & Behavior Change

Awareness campaigns should move beyond just making people scared; they should provide simple actionable steps.

Fogg Behavior Model Users need Motivation, Ability (system usability), and a Prompt to change behavior.

MINDSPACE Framework Factors influencing behavior change include Messenger, Incentives, Norms, Defaults, Salience, Priming, Affect, Commitments, and Ego.

Simulated phishing attacks can train staff to spot malicious indicators and may identify organizational vulnerabilities. However, adversaries continually adapt, requiring frequent training or obsolescence. This can also backfire by making employees ignore genuine warnings due to desensitization. This also creates profound distrust between employers and employees.

Punishing poor behavior (a **deterrence** strategy) is often ineffective as users circumvent policies out of necessity, not malice.

Sanctioning leads to non-reporting; users would rather hide their mistakes out of fear, stopping the IT from spotting vulnerabilities or responding to damage.

5.7 Accessible Security

Security should be designed universally. Designing only for the *average* user leaves massive vulnerabilities for marginalized populations.

6 Availability & Malware

6.1 Importance of Availability

(recall the definition of Availability from the CIA triad.)

If availability is compromised,
- operational disruption
- poor user experience -> reputational damage
- material (commonly financial) damage
- cascading systematic effects

Primary threats to availability
- hardware failure: reliability engineering issues; solved via redundancy
- malware
- Denial of Service attacks: intentional attacks, usually externally, without malware within the system

Many fundamental internet protocols (TCP/IP, BGP) were built on trust (initially used only in institutional research). They lack mechanisms to prevent / mitigate many forms of denial-of-service attacks. Core redesigns are necessary, but impossible to deploy globally.

6.2 Denial of Service (DoS) Attacks

Denial of Service Attack Preventing *authorized* access to a system or resource, usually by exhaustion achieved via techniques like *amplification*.

Resource Exhaustion A mechanism to execute DoS: intentionally consuming a finite resource until none is left for legitimate users. *Flooding* is the volumetric subcase; algorithmic-complexity attacks exhaust without flooding.
Volumetric: saturating (network) link bandwidth (e.g. UDP floods).
Protocol: saturating protocol-state limits in software (e.g. TCP connection tables, using SYN floods).
Application: exhausting compute resources (e.g. RAM, CPU, database connections).

As the Application layer is usually more shielded from the outside world, application layer attacks are often enabled by internal vulnerabilities (e.g. a memory leak in service software).

Vulnerability-based DoS A mechanism to execute DoS: exploiting vulnerability or logical flaw to stop a service completely.
Hard Crashes: using (e.g. malformed) inputs or vulnerabilities to trigger a software crash.
Infinite Loops: using a vulnerability to exhaust CPU.
Account Lockout: intentionally triggering a security mechanism (e.g. repeatedly entering incorrect passwords) to stop the actual user from accessing their account.

As should be evident, some DoS vectors are created by security mechanisms themselves, like account lockouts.

Permanent DoS (PDoS) stuff like physical destruction. Also includes ***Phlashing***: exploiting firmware-bricking vulnerabilities; and vulnerabilities that cause permanent data deletion.

6.2.1 Network DoS

(here we specifically focus on DoS in the TCP/IP stack.)

6.2.1.1 Link Layer

802.11b Wi-Fi (Ethernet / physical links are relatively more secure)

- Jamming: simple continuous transmission on the 2.4GHz freq (with enough power to drown out others)
- Network Allocation Vector (NAV) Bug: setting the 15-bit channel-reservation field to its max value prevents other nodes from transmitting
- De-authentication Bug: "de-auth" packets can be sent without authentication, so one can repeatedly kick users off an Access Point.

6.2.1.2 Network Layer

IP

- Smurf Attack: (Data Flood) attacker spoofs the source IP with victim's IP and sends ICMP Echo Requests (pings) to a network's *broadcast address*. All devices on that network replies, to the victim. This effectively multiplies traffic by the size of the broadcast network.

BGP (note: BGP is technically an Application Layer protocol (it sits on top of TCP))

- Route Hijacking (BGP Trust Vulnerability): BGP relies on trust. (Pakistan Telecom, 2008) tried blocking YouTube domestically by using BGP to route traffic to black hole. They used a `/24` address mask, which was more specific than YouTube's actual `/22` mask, and since specific paths are preferred over broad ones, it caused all YouTube traffic globally to get sent to Pakistan.

6.2.1.3 Transport Layer

TCP

TCP's 3-way handshake (SYN → SYN/ACK → ACK) is exploitable.
- Low-rate SYN Flood: attacker sends victim SYN packets with random spoofed source IPs. The victim stores state in memory for each packet, waiting for ACKs that never arrive, exhausting the half-open connection backlog queue. Mitigation: SYN cookies.
- Massive SYN Flood: a botnet sends a crapton of SYN packets, saturating network links.

6.2.1.4 Application Layer

- DNS Amplification: attacker sends a DNS request spoofed with the victim's IP address using EDNS extensions to resolver. The packet itself is 60 bytes, but the response is 3000, creating 50x *amplification*.
- Memcached Attack: (Github, 2018) amplification exploit using public memcached caching servers.
- TCP Connection Flood: Botnets complete full TCP handshakes and moves the flooding to application connections (e.g. HTTP requests).

More on memcached:
memcached is a in-memory caching software, like an older, simpler version of Redis. Misconfigured (passwordless) exposed servers allowed anyone to 1. "save" a large value 2. "get" this large value, except with a spoofed address so the large response goes to the victim. memcached used to use UDP by default, making the spoofing trivial (does not require connection handshake like TCP).

6.2.2 Notable Attacks

- Estonia (2007): Massive political attack hitting government/bank sites using ICMP and TCP SYN floods. Estonia mitigated it by blocking all foreign web traffic.
- Root DNS Servers (2007): Botnets attacked the 13 root internet servers.
- Slammer Worm (2003): Exploited MS SQL Server via a single 376-byte UDP packet. Saturated internet links globally in under 10 minutes.
- Mirai (2016): Botnet comprised of millions of poorly secured IoT devices (cameras, fridges) running default credentials. Attacked Dyn DNS, knocking out Twitter, Reddit, and Netflix.

6.2.3 DoS Mitigation Strategies

6.2.3.1 Architectural Defenses

Proxies

- Content-Delivery Networks (CDNs) & Scrubbing Proxies: (e.g. Cloudflare, Akamai) distribute load globally. Proxies sit in front of the web server, handle the handshakes, and only forward fully established TCP connections to absorb SYN floods. CDNs remove the vast majority of the load from the origin server to absorb even application-layer attacks.
- SYN Cookies. When the server receives a SYN packet, instead of saving the connection details in memory, it encodes them into a 32-bit Initial Sequence Number (ISN): a coarse timestamp counter, a small MSS index, and a MAC of the client's IP/port + a server secret. The server sends a SYN-ACK back with this ISN as the TCP header's Sequence Number. If the client is legitimate, it will respond with an ACK packet. The client's "Acknowledgement Number" is always the server's Sequence Number + 1. The server subtracts 1 from the acknowledgement number, re-calculates the hash using the packet's info and compares. If they match, the connection is legitimate and only then will the server allocate memory for it.

6.2.3.2 Rate Limiting & Filtering

- Client Puzzles: a proof-of-work mechanism requiring clients to solve moderately hard computational problems (very similar to crypto mining). A drawback is of course that the computational power varies greatly from professional workstations to mobile devices…
- CAPTCHAs: application-layer human verification
- Ingress Filtering: ISPs drop outgoing packets if the source IP is not from within the network (indicating spoofing). Of course, as long as one ISP does not do this, then its users can more easily perform spoofing.

6.2.3.3 Source Identification

Regarding source IP spoofing: discovering the true origin of spoofed packets to block them at the source.

- Edge Sampling: (aka. Probabilistic Traceback) routers probabilistically mark a small proportion of packets with their IP edge data (start, end, distance) (into the 16-bit IP identification field). The victim statistically reconstructs the probable path. The expected number of packets needed is $E(X) < \frac{\ln(d)}{p(1-p)^{d-1}}$ where d is the path length, p = marking proportion.

6.3 Botnets

Botnet A network of compromised machines (aka. bots/zombies/drones) controlled remotely by a *botmaster*.

Botnets are used for *Distributed* DoS (DDos), spamming, email harvesting, keylogging, crypto-mining, hosting proxy *stepping-stones*, etc.

Command & Control (C&C) The architecture used to control a botnet:
Centralized: push style (IRC channels) or pull style (HTTP polling). Single point of failure.
P2P: (local) network relay chains. More robust and stealthy. Uses custom protocols or blends in with standard ones.
Domain Generation Algorithms: (aka. Domain Flux) C&C is located at some domain that is not communicated or within the bot itself, but is generated using some deterministic algorithm. Super stealthy when done right.

6.3.1 Takedown Strategies

- Attack C&C Infrastructure: seize domains or take down IRCs
- De-peering: disconnecting hosting services that "don't care" about malicious services from the wider internet.
- Honeypots: deploying deliberately vulnerable bait targets to get infected; researchers can then study / reverse-engineer malware.

6.4 Malware

Malware is no longer written for fun, fame, or vandalism (as in the 80s/90s); most malware now are sophisticated profit-driven criminal efforts.

6.4.1 Taxonomy

Parasitic (Need a host program)
- Viruses
- Trojan horses
- Logic bombs

Self-contained / Standalone
- Worms
- Rootkits
- Spyware

6.4.2 Parasites

Feature	Viruses	Worms
Independence	Parasitic	Standalone
Activation	Requires human interaction (open file or run program)	Autonomous
Replication	By attacking itself to other files on the *same* system	By sending copies of itself to *other* systems over a network
Speed of Spread	Moderate, limited by users	Rapid, no realistic limit
Defense	AV Scanners, Sandboxing	Intrusion detection systems (IDS), Firewalls, Low-level exploit protection (e.g. ASLR)

6.4.2.1 Viruses

A virus injects its code into a host program. When the host runs, the virus runs first, then transfers control back so the host appears to behave normally.

Lifecycle: *dormant* → *propagation* (infect more hosts) → *triggering* (condition met, e.g. date) → *execution* (payload).

Concealment techniques to evade signature-based AV:
Encrypted payload encrypted under a per-instance key. Only the small decryptor stub is constant (and thus signaturable).
Polymorphic the decryptor itself is regenerated each infection (different register choices, junk instructions). No two copies share bytes.
Metamorphic no encryption at all; the entire body is rewritten each generation (instruction substitution, reordering).

6.4.2.2 Worms

A worm is standalone: it scans for vulnerable hosts over the network and propagates without user interaction. Spread is bounded by scanning strategy and bandwidth, not by users.

Scanning strategies:
- Random: pick IPs uniformly. Simple, slow start, wastes packets on dark space.
- Hit-list: pre-computed list of known-vulnerable hosts seeds the initial wave for explosive early growth (Slammer-style).
- Topological: harvest addresses from the infected host (address book, peer lists). High hit rate but limited reach.

Spread famously follows a logistic curve; the early phase is exponential, capped by the size of the vulnerable population.

6.4.3 Trojans, Spyware, and Rootkits

Trojan Horse Malware disguised as legitimate, useful software. Relies entirely on social engineering. Cannot self-replicate. Often opens backdoors.

Spyware & Keylogger Malware that secretly monitors user behavior. Keyloggers capture keystrokes to steal passwords, card numbers, etc.

Rootkits Advanced malware designed to remain completely invisible and maintain privileged access.
User-space Rootkits: replaces standard utilities (e.g. `ls`, `ps`, `syslogd`)
Kernel-space Rootkits: modify privileged kernel memory directly; near-impossible to detect or remove from within the running OS.
There's also lower-level malware on the bootloader level.

7 Authentication

Authentication is the process of proving you are who you claim to be (before being granted access to non-public resources). It is distinct from *access control*, which concerns what exactly you can do, once authenticated.

Three general factors:
- What you **know**: private knowledge, e.g., PINs
- What you **have**: physical or cryptographic possessions, e.g. keys (mechanical or electronic), 2FA authenticators
- What you **are**: non-informational properties of the user, e.g. biometrics

7.1 What you know – Passwords

7.1.1 The Password Design Dilemma

A good password system must simultaneously satisfy contradictory requirements:
- easy to *memorize*, but hard to *guess*
- easy to *use*, but hard to be *stolen*
- easy to store *securely*, but also *retrievable**

*Whether retrievability is desired is debatable :p

7.1.2 Password Storage

Naive plaintext
- storing (username, password) verbatim
- a data breach leaks everything

Hashed
- store (username, $H(\text{password})$)
- on login: compute $H(\text{entered password})$ and compare to stored hash
- assuming one-way hash function, data breach does not immediately reveal actual passwords
- problem: same password = same hash, enabling precomputed lookup tables (and the more space-efficient *rainbow tables*, which trade time for space via hash chains and reduction functions) to recover passwords after a breach

Salted hash
- store (username, $H(\text{password} \parallel \text{salt})$, salt)
- salt is a random value
- now even same passwords produce different hashes, rendering rainbow tables useless (new table needed per-hash; increasing hash size makes this infeasible)
- modern best practice: use a *slow* password-hashing KDF (Argon2, bcrypt, scrypt) so brute-force becomes prohibitively expensive even per-user

7.1.3 Hashed Password Attacks

Dictionary Attack take wordlist, hash each on the fly, compare against password hash / search in list of hashes.

Rainbow table dictionary attack but fully pre-computed. Usually uses a much larger dictionary for higher hit-rate.

Rainbow table with dictionary size d: hash d strings… profit.
With x-bit salt: now need to hash $d \times 2^x$ string = salt combos… sad.

With a 32-bit salt, there are $\sim 4 \times 10^9$ possible salt values requiring as many rainbow tables.
Salting defeats universal precomputation, significantly nerfing rainbow tables. Dictionary attacks on a per-user basis still works.

A small (~10) GPU cluster can perform $\sim 10^{11}$ (SHA-512) hashes per second, making it possible to crack hashes (by exhausting dictionaries) on the hourly timescale. (`bcrypt` is significantly slower, at $\sim 10^6$/s.)
Regardless of a slow hash function or not, by NOT choosing a password that could end up in a dictionary (e.g. pseudo-random strings), you get cracking-resistance for free.

7.2 What you have – Cryptographic Tokens

7.2.1 Challenge-Response Auth

Alice and Bob have a previously-established, persistent pair of symmetric keys.

Challenge procedure:
- Alice wants to authenticate with Bob
- Bob provides Alice a challenge, a random string
- Alice computes a MAC (or encryption) of the challenge under the shared key, sends back
- Bob recomputes/decrypts to verify

The challenge is random to prevent replay attacks.
This can also be extended to happen both way for bidirectional auth.

7.2.2 One-Time Passwords (OTP)

…prevents replay attacks by definition.

Hash-chain OTP (Lamport scheme):

Setup (server-side):
1. user generates random secret w
2. user computes $H^t(w)$ (hashes t times) and sends to server
3. server stores value and stores counter $j = 0$

Authentication (i-th login)
1. user sends their name, i, and $H^{t-i}(w)$
2. server checks that $i = j + 1$ AND $H(\text{submitted value}) = \text{stored value}$
3. update stored value to submitted value and increment j

Pros:
- server does not store any secret; can be compromised and will be ok

Cons:
- you get exactly t logins; need re-setup afterwards. (The t-th login reveals w itself, so re-setup must precede running out.)

Time-based OTP:

shared secret + current universal time (bucketed into a few-second window) = temporally temporary codes.
Requires clock synchronization. Not strictly *one-time*: the same code is reusable within its time window.

7.2.3 What You Are – Biometrics

Biometric A measurable property of an individual from which distinguishing and repeatable features can be extracted for automatic recognition.

- Distinguishing: reliable tell two different people apart
- Repeatable: reliably recognize the same person as, well, the same person (i.e. must produce consistent features across measurements)

Two-Phase Process:

1. Enrollment
   1. capture multiple raw samples
   2. extract digital feature template
   3. store template
2. Authentication
   1. capture new sample
   2. extract features
   3. compare to stored template (fuzzy match)

Problem: threshold for fuzzy? Too loose = insecure; too tight = many false lockouts.

Modalities:

Modality	Usability	Maintainability	Match Accuracy	Feature Stability	Notes
Fingerprint	ok	ok	good	good	need to check for "liveness" to prevent amputation attacks; "gummy-bear" attacks possible
Hand Scanner	good	good	ok	good	hard to train the model
Iris Scanner	ok	good	good	ok	patterned/coloured contact lenses can cause issues
Retina Scanner	poor	good	excellent	excellent	invasive (NIR through pupil); rarely deployed outside high-security

Pros:
- nothing to remember
- passive, can't lose it
- cannot be shared or lent
- unique (assuming perfect accuracy and a suitable biometric)

Cons:
- revocation is impossible
- invasive: ties an authentication medium to an individual, uniquely
- not secret (e.g. fingerprints can be lifted)
- false acceptance: by nature of fuzzy… (recall birthday paradox)

More…
Stored templates are a privacy risk
Biometrics are implicit identification
User confidence massively declines as soon as one attack succeeds

7.2.4 Multi-Factor Authentication (MFA)

Combine ≥2 factors from different categories (know / have / are) so compromising one does not break authentication. Two factors of the *same* category (e.g. password + security question) is not MFA.

Common second factors, weakest to strongest:

SMS-OTP code sent by SMS. Vulnerable to SIM swap, SS7 interception, and phishing (the code is just a shared secret in transit).

TOTP authenticator app time-based code on a device. Phishable (attacker proxies the code in real-time) but not interceptable over the carrier.

Push approval app prompts "approve this login?". Phishable via *MFA fatigue* (spam approvals until the user taps yes by accident).

FIDO2 / WebAuthn hardware-backed public-key challenge-response. The authenticator signs a challenge bound to the actual origin (anti-phishing) and never releases the private key.

7.2.5 Attacks

Phishing trick the user into entering credentials at an attacker-controlled site. Defeats most "what you know" and many "what you have" factors; FIDO2/WebAuthn defeats it by binding to origin.

Replay capture a valid auth message and resend it later. Prevented by freshness (random challenges, timestamps, nonces).

Credential Stuffing replay (username, password) pairs leaked from one breach against unrelated services. Effective because of password reuse.

Password Spraying try a few common passwords against many accounts (avoiding per-account lockouts that defeat brute force).

SIM Swap social-engineer the carrier into porting the victim's number, intercepting SMS-OTPs and recovery flows.

MFA Fatigue spam push approvals until the user taps "approve" out of annoyance or confusion.

Adversary-in-the-Middle real-time proxy that relays login forms (and OTP codes) between victim and the legitimate site, capturing the resulting session.

8 Access Control

Authentication alone is insufficient – not all users, even authenticated, should be granted access to *everything*.
Authentication is a *coarse* form of access control; it is a boolean check on your ability to access a system *at all*, whereas this is about *fine-grained* access control.

8.1 Core Model

Component	Meaning
Subjects (S)	Users of the system (commonly modeled as *accounts*)
Objects (O)	Resources on the system (e.g. files)
Access Rights (R)	Extent of access (e.g. execute, read, write, append)

An access right can be characterized by two dimensions: *alteration* and *observation.*

	no alteration	**alteration**
no observation	execute	append
observation	read	write

(This is an abstract model: "execute" assumes the runtime doesn't leak file contents to the subject, even though concrete OSes typically read the binary into the subject's memory.)

Reference Monitor The component of a system that enforces access control decisions. (e.g. operating system, hotel staff)

8.2 Access Control Matrix

A simple access control model.

...subjects as rows, objects as columns, and cells containing the corresponding access rights.

E.g.:

	`file1`	`file2`	`file3`
Alice	read, write		read
Bob		read, write	read, write

1. Subject requests to access and object
2. *Reference monitor* looks in the matrix
3. Grant or deny access accordingly

Problem: access control matrices are not very space-efficient, requiring `#user` × `#files` entries.

8.3 Access Control List

Basically a sparse representation of an access control matrix. A bit less inefficient.
Each object stores a list of subjects who have rights (and the specific rights in question).

8.4 UNIX Permissions

An implemented access control model with major industrial use.

Traditional UNIX permissions are an ACL-*like* approach (not true ACLs — POSIX extended ACLs are a separate, later addition). Each file only stores permissions for three generalized subjects:
- owner
- group
- world

The *owner* refers to one specific, concrete subject (user).
The *group* refers to a predefined collection of subjects.
The *world* is any other subject that is neither the owner nor belongs to the group.

It is important to note that the "owner" just refers to some arbitrary user. UNIX does not have *ownership* semantics; that is, being the *owner* does not semantically grant any additional permissions.
(Owners do retain administrative privileges outside the `rwx` bits, e.g. `chmod` and `chown` to themselves, since these go through dedicated syscalls rather than the access matrix.)

The set of access rights (for any of these indirect subjects) are
- read (`r`)
- write (`w`)
- execute (`x`)

This results in a total of 9 boolean values, stored as 9 bits, and commonly represented as a 9-character string: [owner-r][owner-w][owner-x][group-r][group-w][group-x][world-r][world-w][world-x], where a corresponding character is '-' if that permission is denied. E.g.:
- `rwxrwxrwx` : everyone has all permissions
- `rwx------` : owner has all permissions, everyone else has none.
- `---r-xr-x` : the "owner" has no permissions, but everyone else can read or execute.
- `rwx---rw-` : the owner has all permissions, the group has none, and everyone else can r/w.

The reference monitor checks a subject against the owner, group, and world permissions **in order**, and applies the first match.
Hence, one can achieve *negative* permissions; e.g. if the owner has less permissions than world.

For **directories**, "read", "write", and "execute" are not semantically intuitive:
- read = list contents of directory (allows `getdents(dir)` syscall)
- write = create, rename, or delete the contents, (*only* in conjunction with execute)
- execute = "traverse" (allows the directory as a *path component* to syscalls, e.g. `stat` on contents)

...then, there's also the interaction between read and execute for directories:
- read (no execute) = can list all files; names, but cannot `stat` them
- execute (no read) = semantic "dark read" access; can access files by direct path, but cannot list
- read *and* execute = semantic "read" access; can discover and access contents (according to their permissions), but cannot create/rename/delete contents
- write (no execute) = does nothing; you need execute to do stuff like `touch dir/newfile`

Sticky Bit The `10th` UNIX permission value. When set on a directory, only the owner can rename or delete contents, even if others have write permission on the directory. This is useful for shared directories like `/tmp` where all users need `rwx` but shouldn't be able to delete each other's files.

Root User:
- default owner of all system files
- has all permissions regardless of the permission bits
- critical in multi-user systems, protecting users from themselves (messing up system) and each other

`sudo`:
- temporarily grants a user the privileges of another user (root by default)
- principle: don't run as root all the time; elevate only when needed

8.5 Design Principles

- Least privilege: set permissions to minimum needed; don't make files globally readable by default.
- Separation of responsibilities: different roles get different permissions.
- Complete mediation: every file access goes through the reference monitor; no caching of permissions that could become stale.
- Fail-safe default: no permissions should mean deny by default
- Defense in depth: use other mechanisms in addition to e.g. UNIX system permissions
- Open design: reference monitor source should be auditable
- Psychological acceptability: Intuitive system
- Economy of mechanisms: Keep the TCB (below) small

8.6 Trusted Computing Base (TCB)

Trusted Computing Base (TCB) Every component (hardware or software) upon which the *security policy* relies. If any TCB component is compromised, the security policy may be violated.

For access control, the TCB includes:
- TPM chip: hardware that verifies the integrity of low level software (bootloader, kernel) (*secure boot*)
- the kernel itself
- any `setuid` programs (below)

8.7 Unix Processes & Privilege System

The system complementary to the UNIX Permissions model.

Process Isolation
- Processes cannot access each other's memory (by default)
- Each process runs with the permissions of its associated *User*
- A process can access any file its user has access to

Specifically, user accounts are identified by and represented as numeric IDs.

There are three User IDs (UIDs) for a process:
- Real UID (RUID): identifies who the process *belongs* to
- Effective UID (EUID): what the reference monitor uses for permissions
- Saved UID (SSUID): used to store the value of the EUID before modification

All processes are spawned by a parent (except `init`); on process creation:
- RUID is inherited from parent's RUID
- EUID is inherited from parent's EUID, or set to file's owner if `setuid` bit is set (see below)
- SSUID is set to the same as EUID

NOTE that the above is `exec*` syscalls; `fork` copies UIDs unconditionally.

Changing UIDs:
- Root:
  - `setuid(x)`: changes all 3 to x
  - `seteuid(x)`: changes EUID to x
- Unprivileged users:
  - `setuid(x)`: changes EUID to x only if x is RUID or SSUID (or EUID)
  - `seteuid(x)`: same as `setuid`

8.7.1 Elevating Privileges (`setuid` bit)

Unprivileged users often need *elevation* for specific operations (e.g. `passwd` modifies `/etc/shadow`, which only root can r/w).
The setuid permission bit (the `12th` bit) on an executable file means when the file is executed, the process's EUID (and SSUID) is set to the file's owner rather than inheriting the parent's (default).

(The setgid bit is the `11th` bit, and works similarly but for groups instead of users.)

Pros of UNIX model:
- simple
- flexible enough for most access control policies

Cons:
- ACLs are coarse-grained (only 3 subjects)
- cannot differentiate between processes run by the same user; you cannot have different processes have different permissions without running them as different users
- nearly all system operations require root, creating a large attack surface

8.8 Access Control Policy Types

Mandatory Access Control
- Central authority determines access
- Even if you "own" a file, you cannot grant access to someone else unless the system policy allows it, the discretion is taken away from the user
- Very secure at the cost of being rigid and difficult to manage/configure

Discretionary Access Control
- Access is at the discretion of the owner, if you create it, you own it, and decide who else can touch it
- *nix style
- Prone to user error as the system won't enforce anything (can make sensitive file world-readable)

Role-Based Access Control
- Permissions are tied to roles ("Manager", "HR", or "SysAdmin" rather than User IDs)
- May implement MAC or DAC

8.9 Graham-Denning Model

Graham-Denning Model A formal access control model defining 8 *operations* on an access control matrix to characterize how it evolves over time.

Below, X's (and S's) are objects, and O's are objects, and R's are access rights.
A right R can additionally be marked as "transferable", denoted R^*.
"A only if (...)" means A can only be performed if the right represented by the tuple is in the reference monitor's database (access control matrix).
Similarly, "A generates (...)" means A creates the corresponding entry in the access control matrix.
(A, B, R) means subject A has right R ("owner" or "control") on subject/object B.

X creating O generates $(X, O,$ "owner").
X can delete O only if $(X, O,$ "owner").

X creating S generates $(X, S,$ "owner") AND $(X, S,$ "control").
X can delete S only if $(X, S,$ "owner").

X can grant R or R^* on O to S only if $(X, O,$ "owner").
X can transfer R or R^* on O to S only if (X, O, R^*).

X can revoke R on O from S only if $(X, O,$ "owner") OR $(X, S,$ "control").
X can read rights on O for S only if $(X, O,$ "owner") OR $(X, S,$ "control").

Note that the "subject" (item #2) of a table entry can be *either* a subject or an object.

Grant vs. Transfer: only *owners* can grant rights, whereas any holder of a *transferable* right (R^*) can transfer it. Note that X retains R^* even after "transferring" it.

8.10 Access Control in Organizations

Idea: effective access problem in practice, especially in larger organizations, is difficult.

Correctness requirements:
- no gaps: every resource must be covered by policy
- no conflicts
- no unintended restrictions: *primary goals* must not be impeded

Logistical challenges:
- Information asymmetry (between different employees): e.g. security admins not knowing someone was fired
  - Insider attacks often result from the failure to revoke access of former employees
- Efficient updates: people's roles change, and access needs to be kept up-to-date

8.11 Modern Models

Android has a unique and effective access control / security model:

- per-application (per-user) permissions (e.g. GPS access in a mobile app): higher granularity
- runs SELinux which enforces MAC over all processes (every process runs as unique user)
- uses per-app sandboxing: apps cannot access outside its own data by default
- runs the user as non-root by default (and in fact never grants the user root)

9 Security Examples