EPFL Computer Security Summary

Matthias Zeller

June 2020

Contents

1 Access control

= determining whether a principal has authorized access to an object. An authorization is a tuple (principal, access, object).

Terminology

- **Subjects** (often = principal) = entity within an IT system
- **Object** (=asset) = resources that subject may access or use
- **Operation** = read, write, append, execute

1.1 Access Control Matrix (ACM)

- Abstract representation of all permitted triplets (subject, action, access right)
- Must be dynamic
- 2nd role: determine which subjects can provide permissions on objects to other subjects
- Must implement security policy without ever violating it
- Not suitable for direct implementation
  - Not scalable
  - Very sparse → inefficient
  - Error prone: no global view

Graham-Denning Model Extends matrix with:

- Ownership: each object has owner = subject that creates the object
- Controllership: each subject has a controller = subject that creates the subject
- Allows delegation and revocation

1.2 Types of access control

- **Mandatory access control** (MAC): central authority (military, hospital, baking)
- **Discretionary access control** (DAC): object owner assign permissions

1.3 How to store an ACM

1.3.1 Access Control List (ACL)

- Store the access control matrix by column
- Permissions are associated to **objects**
- Stored with the resource
- Easy to determine who can access a resources
- Difficult to see all rights of a user
- Difficult permission delegation
- Difficult to check at run-time, confused deputy problem, not always clear which subject accesses the resource
- Unsuited for systems with many subjects that come and go → large dynamic ACL, two alternatives:
  - Role-based access control
  - Group-based access control

1.3.2 Capability

- Store ACM by row
- Permissions associated to subjects
- Stored with the subjects
- Easy to visualize all subject permissions
- Easy to delegate
- Revoking permissions on one object is hard
- Hard to ensure legitimate access right transfer

1.4 Role-based access control (RBAC)

- Assign permissions to roles, assign roles to subjects, subjects select an active role
- Rationale: some subjects are similar to each other and have the same rights
- Alternative to ACLs systems having many subjects that come and go
- Problems:
  - Role explosion, temptation to fine grain roles
  - Limited expressiveness: problem with least privilege
  - Difficult to implement separation of duty, pb with separation of privilege

1.5 Group-based access control

- Assign permissions to access objects to groups, assign subjects to groups, subjects have the permission of all their group
- Rationale: some permissions are always needed together (e.g. access to sockets and access to network interface)
- Alternative to ACLs systems having many subjects that come and go
- Supports negative permissions, they're always tested first

1.6 Example: UNIX/Linux access control

- Principals and groups:
  - User (= principal) identities UIDs, group identities GIDs
  - User information
  - Users belong to ¿= 1 group

- Security architecture
  - Discretionary access control
  - Each user owns a set of files
  - All processes of a user run with that user's privileges = ambient authority
  - 3 "groups": owner, group, other
- Super users
  - Special root user account
  - Can access everything, no access deny
  - Almost no security checks
  - Root is in the Trusted Computer Base (TCB)
- Access control Lists
  - Files have ACLs attached: each file has a owner UID and GID assigned, 9 permission bits
  - Different semantics between files and directories
  - 3 attributes: suid, sgid, sticky
- Access control implementation:
- Special rights: suid, sgid

1.7 Ambient authority

- Implicit subject and authority, only operation and names are specified
- Simplifies program design and usability
- Problem with least privilege
- ACL generally considers ambient authority, permissions usually checked for the user running the program
- Capabilities: they themselves contain the identity of the principal, no ambient authority
- **Confused deputy**: privileged principal used by another less privileged principal to improve its access. To avoid those problems:
  - Re-implement access control in the privileged process: expensive, cumbersome, error-prone
  - Let privileged process check authorizations
  - Use capabilities instead of ACLs

1.8 Access contorl implementation

Should use reference monitor: check everything. Do NOT check accesses all over the program. This violates the least common mechanism principle. This supports economy of mechanism and complete mediation.

2 Security models

For instance, when considering the type of access control (MAC, DAC), we can wonder about the basic security principles behind these policies. We can ask: "Are there things owners of objects are *not* allowed to do ?". Yes → MAC, no → DAC.

The lecture focuses on the MAC security policy models. These policies determine all relations between subjects and objects. The user and owner of an object has no rights to establish access policy of the object.

2.1 What are Security Models?

- Not a policy or mechanism itself
- Design pattern to reason about security properties to build security policies
- Doesn't provide details of the policy
- A security model might or might not be appropriate for a specific security property
- First security model: Orange Book, covers (mainly) needs of government for confidentiality

2.2 The Bell-LaPadula Model (BLP)

- **Multilevel** security approach
- Tackle the **confidentiality** problem
- Family of models subject to refinement and extensions

2.2.1 Overview on BLP Components

- Subjects S and Objects O
- Access to objects: 4 values, execute, read, write, append (i.e. can't read, only add content at the end)
- System state
  - Level functions: mapping subject/objects to classifications
  - Access Control Matrix
  - Current access set: current authorized triplets in the system (subject, object, attribute)

2.2.2 Level function for objects: Classification

- Classification: total order of labels (*e.g. Unclassified, Confidential, Secret, Top Secret*)
- Categories: compartments of objects on one topic (*e.g. Nuclear, Crypto*)
- Security level = (Classification, {set of categories})

Dominance relationship

- Dominance relationship: level (l1, c1) dominates level (l2, c2) if:
  - $l1 \geq l2$ ($\geq$ means more secret than)
  - $c2 \subset c1$
- Representation: Lattice
- Properties: dominance is transitive, top and bottom levels, no total order (*i.e. some have no relationship*)

Clearance level

- Clearance = maximum security level a subject has been assigned
- Current security level: subjects can operate at lower security levels
- level(S_i) must dominate current_level(S_i)

2.2.3 BLP properties

Simple security property

- SS-property: if (subject, object, w/r) is a current access, then level(subject) dominates level(object)
- In other words, a subejct can read objects with lower classification than its clearance
- Also called No Read Up (NRU)
- Not sufficient though: just forbidding people to read up doesn't prevent malicious code to high level classified information to lower levels (violates the ss-property)

Star property

- *-property: if subject has simulatneous observe (r,w) access to O_1 and alter (a,w) access to O_2, then level O_2 dominates O_1
- In other words, a subject cannot write in clearance levels lowers than theirs, so they cannot leak information
- Due to ss-property (no read up), the subject can only append to higher levels

Discretionary property

- ds-property: if an access (subject, object, action) takes place it must be in the access control matrix
- Information should only be accessed on a need-to-know basis → least privilege principle
- Using DAC, the system can protect integrity of objects by forbidding subjects from writing
- This is basically saying that we use an ACM

2.2.4 Basic Security Theorem

- If all state transitions are secure, and the initial state is secure, then every subsequent state is secure regardless of the inputs
- If for any individual access, the ss-property, the *-property and the ds-property hold, then any sequential composition security holds
- ⇒ a system can be analyzed in terms of single step transitions of state
- Remind: a state is the current access permissions, the ACM and the clearance level

2.2.5 BLP Problems

- Confidentiality-oriented: does not consider integrity or availability
- State-based + single transition model: too low-level, not expressive
- the 3 security properties are not sufficient to ensure confidentiality

Illegal information flow If level(S_1) is TS (Top Secret), level(S_2) is C (Confidential):

1. S_2 creates O_2 → level(O_2) = TS
2. S_1 reads C and either:
   - changes the object level
   - leaves object level untouched
3. S_2 attempts to access to O_2 in C → success or failure leaks 1 bit of information

2.3 Covert channels

- Covert channel: any channel that allows information flows contrary to the security policy
- A covert channel is enabled by shared resources accessed by subjects with different clearance levels
- Least common mechanism: the more resources are shared, the harder it is to eliminate covert channels
- Eliminate covert channels:
  - Isolate the high level: once information is up don't let it go down. But not practical
  - Add noise to communications: add fake interaction between high and low levels such that real intended covert channels cannot be distinguished from random noise
  - With 1 bit/second of leak, OK for documents, not ok for cryptographic keys
  - DoD policy: store cryptographic keys on dedicated hardware

2.3.1 The tranquillity property

- Tranquillity: classification/clearance does not change during execution
- Illegal flows are enabled by actions in the system not rules by the BLP rules: changes in object/subject levels, changes in the ACM
- But static is not practical

2.4 Declassification

- With no write down, how a subject communicates with lower level subjects ?
- Declassification: remove classification label, under the control of the security policy

- Enables communication with lower layers
- Cannot be made inherently safe, manual process
- Very typical and necessary
- Hard to rule out covert channels (how to know the object does not contain secrets?)

Difficulty of declassification in practice

- Microsoft Word revision history contains deleted text
- PDF redaction by overlying graphical elements, the text is on the file!
- Strategic adversary!

2.5 NRL pump

- System that implements BLP by inserting a router that physically separates communications between high and low levels
- High can read low, but can't write low
- There is a covert channel: acknowledgments.
  - Low levels need ACKs from high levels to know if information is delivered
  - These ACKs could be used as covert channels (e.g. by introducing delays)
  - Avoid problem: the pump buffers the ACK and delivers it after a probabilistically based time (???)

2.6 Integrity

- Integrity also matters!
- Preventing fraud (e.g. in commercial services) is about protecting integrity
- Public key cryptography requires high-integrity for confidentiality
- Properties of integrity: for (a,b) inputs of an operation yielding a result, the adversary cannot
  - influence inputs from honest parties
  - exchange the operation for another one
  - change the result
  - perform the operation more than once
- The system should operate as if no adversary tries to tamper with it

2.6.1 Principles of Integrity mechanisms

Separation of duties Require multiple principals to perform an operation (separation of privilege principle). Examples: accountant records payments/income (the two must match), both sides of a transaction keep a record and must match (legal receipts), two officers requires to launch a missile.

Rotation of duties Allow a principal only a limited times on a particular role & limit other actions while in this role. This decreases opportunities of principals to tamper with the action. Examples: guards appointed for a single random shift to guard a bank safe.

Secure logging Tamper evident log to recover from integrity failures (compromise recording principle). Consistency of log across multiple entities is the key. Needs separation and rotation of duties principle. Examples: logs of transactions inside ATM machine, notaries maintain logs of transactions and contracts, bitcoin.

2.7 The BIBA Model

- Multilayer security approach
- Tackles the **integrity** problem
- Family of models subject to refinement and extensions
- Subjects and objects are assigned an integrity level
- Two strict key rules
- Simple integrity: **no read-down**, protect higher integrity principals from being corrupted by lower integrity level data
- *-integrity: **no write-up**, prevent lower integrity principals from being corrupting high integrity data
- Examples: in bank, director establishes rules and every employee reads, employees cannot rewrite rules. Computer: Web application open in the browser should not write to the file system (at most /tmp)

2.7.1 BIBA variant : Low-water-mark for subjects

- Low-water-mark policy for subjects: subjects start processes at their highest integrity level. When accessing an object, its current level is lowered to the lowest of the two: current-level(s), level(o)
- Temporary downgrade for the session: hard to avoid label creep. Example: mitigate impact of a network Trojan.

2.7.2 BIBA variant 2: Low-water-mark for object

- Low-water-mark for objects: once object has been written by a subject, it takes the lowest level of the object or subject
- Dangerous! Only allows for integrity violation detection, the file cannot harm but tampering has happened
- Idea: if subject pollutes an object by writing on it, this object is downgraded to the level of the subject
- Option to prevent tampering: replicate, and once operation happended, either sanitize or upgrade

2.7.3 BIBA Additional actions

Invocation

- Subjects can interact with objects at other levels, but still protecting the object
- For instance, a Director acts as a protection (run checks or deny operations) for the object that the Teller needs to write on.
- Simple invocation: high level invoke low level → protects high level data. Problem: unclear who modified an object
- Controlled invocation: low level invoke high level → prevent corruption of high integrity data. Problem: hard to check that no secret is actually written in high integrity objects

2.7.4 Sanitization

- Sanitization: process of taking objects with low integrity and lifting them to high integrity
- Root cause of large classes of real-word security vulnerabilities: low input (user) can influence high data and code (service)
- Example: Web server (high) accepts input from web client (low) → SQL injection.

Fail-safe default

- Positively verify that low objects are within a valid set before elevating their integrity to high
- White list: check that all properties of good objects hold
- Do NOT blacklist: or at least, not only

2.8 Combining security properties

- Secure composition of mechanisms is hard!
- Composing confidentiality and integrity:
  - BLP: confidentiality, no integrity
  - BIBA: integrity, no confidentiality
  - Chinese Wall Model
- From multi-level to mutli-lateral security
  - Different entitites seek for different propertiesvu croix englais
  - Properties may have conflicting requirements
  - Who secures the TCB? 3rd party (pb: trust), trusted hardware (pb: trust manufacturer), advanced cryptography

2.8.1 Chinese Wall Model

- Inspiration: UK rules about handling conflicts of interest in the financial sector
- All objects associated with label denoting their origin (e.g. Coca-cola, Pepsi)
- Originators define conflict sets of labels (e.g. Pepsi and Coca-cola is a set)
- Subjects are associated with a history of their accesses to objects and their labels

Access rules

- Direct flow: a subject can read/write and object if the access does not allow an information flow between items in the same conflict set
- Indirect flow: access denied. Happens when A works in Pepsi, B in Coca-Cola, and A and B meet in IBM.
- Enable flexibility (which is restricted due to indirect flow): must sanitize: un-label some items as long as information cannot lead to conflicts of interest. Indirect flow

3 Applied cryptography

Example of an everyday declassification act: publishing code on Github. Many developpers are unknowingly posting their hardcoded credentials online. See the gitleaks repository which uses regexp and entropy (cool!).

3.1 Why Cryptography Matters?

- Frees you from physical security
- Reduces the TCB to the confidentiality and integrity of the keys
- We don't always a TCB (e.g. while data is in transit), so cryptography is useful in this case

3.2 Terminology

- Confidentiality: unauthorized parties cannot access information
- Plaintext: non-encrypted message, readable
- Cryptographic algorithm: set of mathematical instructions to encrypt and decrypt data
- Ciphertext: encrypted message obtained by using cryptographic algorithm and plaintext
- Unlike encoding, encryption and decryption require a key
- Cryptographic primitives: universal, exchangeable cryptographic building blocks. Either can't break it down any further or no security argument for its individual parts

3.3 History - Quest for confidentiality

- Ceasar's cipher (50 BC) and Kamasutra cipher (400 AD) can be very easily broken with frequency analysis

3.4 One Time Pad

- Key = string of random bits as long as the message
- Prevents frequency analysis (repeated characters are encrypted to different values)
- Must never reuse OTP key: even if cannot recover full message, can recover information for frequency analysis
- Problems: key as long as message, key must be random, key cannot be reused, no integrity

3.5 Symetric encryption

- Same key for encryption and decryption
- Two types of cipher: block and stream ciphers
- Integrity mechanism: MAC

3.5.1 Stream cipher

- Key: small
- Initialization vector: not secret, must not be reused. Purpose: same messages encrypted with the same key are different.
- Stream ciphers output stream of bits that is pseudo-random
- Security argument: pseudo-randomness of stream
- Remaining downsides (w.r.t. OTP): key must be random, no integrity
- Strengths: fast and low error propagation
- Weaknesses: Low diffusion (all information of a plaintext symbol contained in one ciphertext symbol) and susceptibility to insertions/modifications (difficult to detect)

3.5.2 Block ciphers

- Key = short random string of fixed size
- Encrypts blocks of short key-size
- Messages often ¿ 1 block, need to chain blocks together with **modes of operation**:
  - Electronic code book (ECB): independent blocks, don't use, $m_1 = m_2 \Rightarrow c_1 = c_2$
  - Cipher block chaining: encryption depends on previous blocks, need an IV each time. Problem: if IV is wrong (but known key), only the first block is corrupted
  - Counter mode (CTR): turns the block cipher into stream cipher, need a nonce, encryption same as decryption
- Stengths: high diffusion, easy tampering detection
- Weaknesses: slow (must accumulate an entire block before encryption/decryption), error propagation (depends on mode of operation)

3.6 Integrity (symmetric)

3.6.1 Message Authentication Code (MAC)

- MAC(K, m): two inputs
- Key property: very difficult to produce (m, MAC(K, m)) without knowing the key K
- Provides mutual authentication but not by third parties
- Problem: repudiation: the two parties having the key can say that the other produced the message, no third party can check

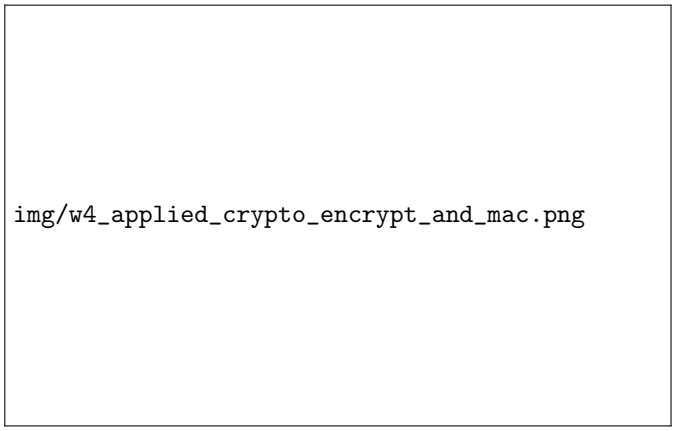
CBC-MAC

- Turning a block cipher into a MAC
- Fixed IV
- The MAC is the output of the last encryption block
- Limitation: only secure if message length is known: otherwise, length extension attack

3.6.2 Combine confidentiality and integrity

Encrypt-and-MAC

- ✓ Integrity of plaintext can be verified
- × No integrity of ciphertext, need to decrypt to know if valid (expensive)
- × May reveal info about plaintext: the IV is fixed (message sent twice → same MAC)



MAC-then-encrypt

- × No integrity of ciphertext, need to decrypt (expensive)
- ✓ Integrity of plaintext
- ✓ No information about plaintext, MAC is encrypted

Encrypt-then-MAC

- ✓ Integrity of ciphertext
- ✓ Integrity of plaintext
- ✓ No information leak, MAC encrypted

3.6.3 Authenticated Encryption with Associated Data (AEAD)

- Combinations are easy to confuse
- AEAD schemes provide authentication and confidentiality in one go
- Requires associated data, besides the key, message and nonce
- Output: ciphertext and tag

3.7 Asymmetric cryptography

- Problems of symmetric crypto: how to securely share keys? How many keys such that everyone can talk to everyone?
- **Public key infrastructure**: public keys can be uploaded to public servers
- Integrity is obtained using **digital signatures**: a piece of data produced with private key and verifyable with public key

3.7.1 Digital signatures

- Properties: integrity of message and authenticity of sender
- Non-repudiation (unlike MACs)
- Applications: Public key infrastructure and **certificates**
- An authority signs mapping between names and public keys, or names and verification keys
- Encryption key pair != signature key pair

3.7.2 Limitations

- Computationally costly: compared to most symmetric key algorithms of equivalent security
- Slow
- Not suitable for large amounts of data
- Solution: **hybrid encryption**, sign hash of message and only encrypt small symmetric key

3.8 Hash functions

- Only 1 input, no key
- Security properties:
  - Pre-image resistance: given H(m), difficult to get m
  - Second pre-image resistance: given m, difficult to get an m'!=m such that H(m) = H(m'), m is fixed
  - Collision resistance: difficult to find any m!=m' such that H(m) = H(m')
- Usage: **support digital signatures**, HMACs, password storage, file integrity, blockchain, ...

3.8.1 HMAC

- H(K || m) is *not* a good HMAC, sensitive to length extension attacks (similar to those in CBC-MAC)

3.8.2 Digital signatures on hash functions

- Sign the hash of the message h(m) instead of the message itself
- Second pre-image resistance: given signature of hash, cannot find another message with same hash (and thus same signature)

- Collision resistance: cannot sign a message and then claim they signed another message m'

3.8.3 Hybrid encryption

- Asymmetric encryption is slow, symmetric is fast
- Use public keys: avoid sharing keys
- Use private key to encrypt actual data → **session key**
- Send/share private key with public-key encryption (small amount of data! fast)
- Still a problem! See forward secrecy

3.9 Diffie-Hellman key exchange

3.9.1 Forward secrecy

- If at tome point in time, one of the two parties has its private key stolen, any past conversation might be decrypted (and future ones, if the private key is still used)
- Desirable property: forward secrecy, i.e. the secrecy of messages in a session is kept even if long term keys are compromised
- Diffie-Hellman provides forward secrecy

3.9.2 Basic maths

- Arithmetic modulo a number
- Arithmetic modulo a large prime p (¿ 1024 bits):
  - Addition and multiplication (mod p) can be computed
  - Exponentiation $(a, x) \rightarrow a^x \mod p$ can be computed
  - Discrete logarithms are hard, $(a, a^x \mod p) \rightarrow x =?$

3.9.3 Basic Diffie-Hellman key exchange

- Shared public parameters p, g
- Both parties have random secret keys x, y
- Public keys $P_{Bob} = g^x \mod p$, $P_{Alice} = g^y \mod p$
- Shared session key (private!): $K = g^{xy} \mod p = (P_{Bob})^x = (P_{Alice})^y$
- After session is ended, delete secrets x and y. The session key can never be recovered, forward secrecy is achieved

4 Authentication

- Process of verifying a claimed identity
- NOT message authentication, in which the message comes from the designated sender and wasn't modified
- We saw authorization, i.e. whether a principal can access an asset, here we need to bind the user to a principal before authorization
- User or machine authentication

4.1 Ways to Prove Who You Are

- Traditional: what you know, what you are, what you have
- Modern: where you are, how you act, who you know, ...

4.2 What you Know: Passwords

- Password = shared secret between user and system
- Must consider many aspects!
- Secure transfer: eavesdropping & impersonation, replay attacks, must secure the channel and use challenge-response protocols
- Secure storage: password database compromises, use pre-image resistant hash function, dictionnary attacks, parllel computing, rainbow tables → salt
- Secure checking: letter-by-letter checks induce timing leaks, consider typo flexibility?
- Secure passwords: put constraints, easy-to-remember pwds tend to be broken easily, often reused across different systems
- Use authentication libraries!!!

4.2.1 Challenge-Response Protocol - Password transfer

Used to avoid replay attacks (for passwords and in general). For A that wants to login to B:

1. A: I want to log in
2. B: send a challenge: random number (from large space)
3. A: encrypt password together with challenge
4. Once B authenticated A, B deletes the challenge

4.2.2 Salted Hash - Password Storage

- The only secure way to store passwords (no way do store in clear, hash alone not sufficient)
- Compute hash $h = H(pwd||salt)$ and store h alongside with the salt
- Less sensitive to dictionary attack: more computationally costly for the attacker
- Important: retrieving all passwords in a database is more difficult, but a targeted attack is equally difficult!
- Use slow hash functions

4.3 What you are - Biometrics

- Biometrics = measurement and statistical analysis of people's unique physical characteristics
- Examples: fingerprint, face recognition, retina, voice, handwritten signature, DNA
- Pros: nothing to remember, passive, difficult to delegate
- Two phases: enrolment and verification

4.3.1 Enrolment

- First phase of biometric authentication
- Introduction of the user biometric in the system and association with their login
- Capture of the biometric with the sensor → biometric template

4.3.2 Verification

- Second phase of biometric authentication
- Capture and process biometric as before and compare biometric template with the stored one
- Local storage vs remote storage: local more privacy-preserving but harder to secure and update, remote is less-privacy-friendly but easy to secure and update
- Comparison is not exact: trade-off false positives and false negatives, depends on the context

4.3.3 Problems with Biometrics

- Hard to keep secret: liveness detection
- Revocation is difficult/impossible
- Identifiable and unique: linking across systems
- May reveal private information: iris and disease, face and identity
- Not always immutable or universal: iris changes with lenses, fingerprint disappear

4.4 What you have: Tokens

- Examples: smart card (signs a challenge sent by ATM to prove knowledge of a key), or device which output a number based on a secret and time (time syncronized with a server, number must match with server number)
- Step 1: offline initialization, common seed (random number) and clock syncronization
- Step 2: upon authentication request, both compute a number that depends on time, only a encrypted number is transferred by the token
- Note: the cryptographic function cannot be a hash since hashes do not require keys (everyone can compute it)

Two Factor Authentication (2FA)

- Combine two out of three factors: what you know, what you have, what you have
- Example: card = what you have + what you know (PIN)

4.5 Machine Authentication

- Machines use public key cryptography to produce digital signatures
- Example: HTTPS/TLS to authenticate the server
- Defend from man in the middle → signatures
- Defend from replay attacks → challenges / nonces

5 Adversarial thinking and attacks

5.1 Why Study Attacks?

- Deeper understanding of defense: very good attackers make very good defenders, mediocre attackers make extremely poor defenders (asymmetry adversary-defender)
- Employability: penetration testing is a major industry
- Need both security and privacy
- Fail safe principle and sanitization: attack space cannot be completely explored (it's huge!), not finding an attack doesn't guarantee security

5.2 How are Attacks Deployed?

- Attacks typically do not happen by chance
- Usually discovered by studying systems in systematic ways

5.2.1 The Security Engineering Process

1. Define security policy and threat model: what to protect from whom
2. Define security mechanisms that support the policy given the threat model: how to protect it
3. Build implementation that supports the mechanisms: implement protections

5.2.2 The Attack Engineering Process

Inverse approach

1. Exploit flaws in security policy and threat model: misidentified principals, assets, properties, capabilities beyond what considered in threat model (access, computational) = underestimation of the adversary
2. Exploit flaws in security mechanisms: design weaknesses

3. Exploit flaws in implementation: software is very complex, many small errors can enable adversary to infiltrate the TCB

5.3 Exploit flaws in security policy / threat model

Exploiting misidentified assets in security policy

- Example: Extract keys from Hardware Secure Modules (HSMs)
- HSM = CPU secured physically, hold crypto keys which can't be extracted by observing device (power consumption, computation timing, ...)
- Economy of mechanism: strict API, can access HSM through small set of functions
- HSMs implement PKCS#11 standard for interoperability
- API to create a new key (internally!) from the secret key: given bit_len and offset, uses bit_len of the secret key from position offset
- Problem: PKCS#11 considers the full key as an asset to protect, not the bytes of the key
- Procedure:
  1. Ask HSM to derive a new key of length 1 at offset 0
  2. Use new key to do an operation (e.g. HMAC) on known input
  3. Brute-force the key (input and output known, only 1 key byte)
  4. Repeat with keys at different offsets

Exploiting unforeseen access capabilities

- Example: from cable to the air. Engine Control Units (ECU) control the vehicle. ECU connected to GSM/Wifi give a remote adversary access to the CAN bus and all the (safety) functions of the vehicle. Due to gradual evolution of technology (initially ECU not connected), this was lacking relevant security
- Example: IoT devices are usually very poorly protected. Researchers at Princeton University showed that accessing those devices and changing their power consumption could crate arbitrary electricity demands (could bring down the grid). Much easier than getting in an electrical central!
- Example: Unilateral user authentication in GSM. When GSM was designed antennas were difficult and expensive to build. So operators did not implement authentication. But now, one can easily impersonate an antenna and make people connect to yours.

Exploiting unforeseen computational / algorithmic capabilities

- Example (capabilities): NSA, brute-force against RSA keys
- Example (algorithmic side): machine learning revolution. Machine learning eases attacks: substitutes complex modeling tasks by data collection.
- Note: machine learning also helps: improved malware detection, predicting zero days, identifying vulnerable devices, automated log analysis

5.4 Exploit flaws in security mechanisms

Exploiting security mechanisms design weaknesses

- Weak cryptographic primitives: Tesla (Key Fob algo) and A5/1 A5/2 for GSM
- Both examples used secret algorithms, security by obscurity is a bad idea, open design principle

5.5 Exploit flaws in implementation

Exploiting bad operation decisions to subvert security mechanisms Example: WEP and bad use of RC4. RC4 is a secure stream cipher. WEP used very small IV, so often repeated, and very bad

Exploiting implementation flaws to subvert security mechanisms Example: programmers make mistakes, besides bad parametrization of algorithms in implementations. Example of bug that allowed users to gain root access in Linux: wrong check in the implementation of the sudo command.

5.6 From specific to generic attacks

- Ultimate goal: elevation of privilege or execution of arbitrary code in the TCB
- Specific attacks: problem in a particular security policy, threat model, mechanism. Adversary may violate a specific security property.
- Generic attacks: get access to the TCB of the system, the adversary can violte all security properties

5.7 Reasoning about attacks

5.7.1 STRIDE

- Helps security engineers to reason about threats to a system in a systematic way: what can go wrong?
- **S**poofing: authenticity
- **T**ampering: integrity
- **R**epudiation: non-repudiability
- **I**nformation disclosure: confidentiality
- **D**enial of service: availability
- **E**levation of privilege: authorization

5.7.2 Common Weaknesses enumeration (CWE)

- Idea: a database software errors leading to vulnerabilities to help security engineers avoid common pitfalls: what not to do.

- MITRE has list of most dangerous software errors with consequences

Insecure Interaction Between Components

- One subsystem feeds another subsystem data that is **not sanitized**
- Class of errors: programmers do not check information that is sent between different components (non-sanitized)
- Example: OS Command Injection. Input controlled by adversary → command to the OS (POST HTTP method)
- Example: Cross-site Scripting (XSS), improper neutralization of input during web page generation. Uses input to dynamically generate web content (GET HTTP method).
- Avoid injection: sanitization! BIBA: never bring information from low (unknown) into high (OS, server). But very hard task: need to define the universe of good things
- Example: cross-site request forgery. Confused deputy (= browser) problem, enabled by ambient authority (= cookie-based authentication)
- Avoid cross-site request forgery: don't accept cookies from unknown websites, use a challenge to avoid "reypling" cookies, don't use cookies

Risky Resource Management

- Software does not properly manage creation, usage, transfer, destruction of important system resources
- The system acts on non-sanitized input
- Example: family of buffer overflow bugs, misestimation of the space reserved in memory
- TCB under the control of the adversary

Porous defenses

- Defensive techinques that are often misused, abused or just plain ignored
- Violation of complete mediation
- Failures and bugs in authentication, authorization, encryption
-
-

o f