Security concepts and principles

Security goals

- **Confidentiality**: keep something secret, data in communication or at rest (cryptography, authentication…)
- **Integrity:** no corruption or control hijacking, who can write (message, data hashing)
- **Availability:** system uptime, response time, free storage…
- **Privacy:** right to be left alone, personal information
- **Accountability:** login and audit trails (secure timestamping, integrity in logs…)
- **Non-repudiation:** two parties can't deny they have interacted (trust a 3rd party and generate evidence)

Security guiding principles

- Secure the weakest link (attacker needs only one flaw)
- Practice defense in depth (use layers of defense)
- Fail securely
- Compartmentalize (separate code into parts)
- Be reluctant to trust
- Principle of least privilege (minimum access and time necessary)
- Keep it simple (to reduce attack surface, tradeoff with usability)
- Promote privacy
- Hiding secrets is hard (hiding details is not enough, attacker can have ways of finding them)
- Use community resources

Examples of threats and attacks

- Web defacement: replace legitimate pages with illegitimate ones
- Infiltration, control hijacking
- Phishing: spoofed site that looks real
- Data theft/ Data loss
- Denial of service (DoS): flood server with packets, cause server to drop legitimate packets or make service unavailable
- Ransomware: malicious software than encrypt a victim's files or locks their system and demands a ransom payment for returning system access

OWASP Top 10

Information Gathering

An attacker wants to find an easy way of attacking, and as a developer you want to decide test scope, coverage and prioritize

What to gather?

- **Application structure** - identify pages, external links, and trust zones (areas with varying access permissions)
- **Data flow** within the application (observe how data moves between client and server, focusing on GET/POST requests, responses, and parameters)
- **Infrastructure or platform** (web server, applications, entry points, execution path, framework...)

Tools: web debugging proxy (example: Burp Suite): this can be used to capture and examine requests and responses, manipulate payloads, attacks... Automated scanners can't capture everything, so manual testing is recommended.

Injection Attacks

SQL injection

An attacker manipulates user input to inject malicious SQL into a query. This allows unauthorized access to data, data corruption, privilege escalation, or even destruction of the database. Also possible in XML with Xpath injection.

- Blind SQL injection: guess database schema with binary search after checking if its vulnerable to sql injection.

- Countermeasures
  - **Blacklisting**: filter quotes, semicolons, whitespace.. It can have problems with functional requirements and it is easy to bypass
  - **Whitelisting**: only allow well defined safe inputs. The problem is that RegExp (regular expressions) are hard to define for all safe values
  - **Escape input**: treating everything as strings and not as logic, like blacklisting, it could miss dangerous characters
  - **Prepared statements & bind variables**: decouple query statement and data input with a template, most robust.
    - ORM:

```
def show_orders(request):
    month = request.GET.get('month')
    orders = Order.objects.filter(user=request.user, order_month=month)
```

- Manual: `cursor.execute("SELECT * FROM orders WHERE user_id=%s AND order_month=%s", [user_id, month])`

- **Mitigate impact**
  - Avoid information leakage (don't display errors and stack traces)
  - Limiting privileges
  - Encrypt sensitive data
  - Key management precautions (don't store encryption key in DB)
  - Hash password

Session Management Attacks

HTTP is stateless, so unless you reauthenticate for every request , you can't know for sure if the requests are from the same clients. With session management, you can assign session tokens and then validate them for every request. You can store session tokens by:

- **Embedding in url**: https://site.com/checkout?sessionToken= 1234. Exposes the token everywhere, basically)
- **In hidden form field**: <input type= "hidden" name = "sessionToken" value = "1234"> Must be included in every form manually, relies on client-side data.
- **In browser cookie**: setcookie: sessionToken = 1234 (these are set in headers of HTTP requests and responses). Safer, but server only sees cookie, not domain who sent the cookie.

Attacks

- **Session token theft**
  - Methods
    - **Network sniffing:** an attacker can see someone's cookie with the session ID when they make requests to sites with HTTP (not HTTPS) and use it (since the server doesn't check the domain from where it is being sent)

Find in settings.py SECURE_SSL_REDIRECT = True (to force HTTPS and cookie settings) SESSION_COOKIE_SECURE = True SESSION_COOKIE_HTTPONLY = True CSRF_COOKIE_SECURE = True 'django.middleware.security.SecurityMiddleware' (in MIDDLEWARE)

    - **Logout problem:** logging out should destroy the token in both client and server. Usually browsers delete the cookie (session expired or logout), but the server keep the session valid, so attackers who stole the token can still use it.

```python
def logout_view(request):
    del request.session['user_id']
    return redirect('login')
```

This is wrong, in Django use logout(request) + short session lifetimes in session.py

    - **Cross-site scripting***: attacker crafts malicious URL with script inside that returns the session cookie from the client's browser (where the script is executed when client clicks into the malicious URL)

views.py

```python
from django.shortcuts import render

def search(request):
    name = request.GET.get("name", "")
    return render(request, "result.html", {"name": name})
```

result.html (TEMPLATE)

```html
<!-- BAD: directly inserts user input unescaped -->
<h2>Your query: {{ name|safe }}</h2>
```

Django has built-in auto-escaping for XSS, so you should be fine. It could be unsecure if it has the "| safe" in html, which should only be used if 100% the data has been sanitized previously

- Solutions
  - Once user logged in (token around), communication through HTTPS
  - Remember to log out
  - Time-out session ID and delete expired session ID
  - Bind session token to client's IP or computer

- **Session token predication attack**: some tokens can be predicted, seeing one or more tokens shouldn't be able to predict others. Solution: use token generators from frameworks (in Django - check code for manually set session keys. Correct: request.session['key'] = value.
- **Session fixation attack:** server sends anonymous token to browser (not logged in yet), which will elevate privileges when the client logs in. If attacker overwrites the token before logging in, he will have an elevated token after logging in.
  - Methods
    - **Tampering through network**: client visits server through HTTP, attacker can intercept and alter the HTTP traffic, and he injects into response an overwritten cookie (session token).
    - **Cross-site scripting** *: attacker crafts malicious URL with script inside that establishes the anonymous session cookie before logging in, and then when the client clicks it and logs in (elevates the session), the attacker has a logged in session.
  - Solution/Mitigation: always issue a new session token, when elevating from anonymous token to logged-in token.

```python
from django.contrib.auth import login

def my_login_view(request):
    ...
    login(request, user)  # this safely regenerates the session ID
```

Does request.session.flush(). Correct.
Look for manually setting session variables without regenerating sessionId (wrong)

*Cross-site scriptiong (XSS) – attacker can insert javascript scripts that will be run in the victim's browser.

- **Reflected XSS**: javascript injected into request and reflected immediately in response

- Stored XSS: script injected into request, stored somewhere in server and reflected repeatedly

Mitigation: sanitize input data, escape (transform "<" into "<" so it is displayed as text and not executed)

Broken access control

Cross site request forgery (CSRF)

Browser has the sessionID cookie stored for bank.com, and when user accesses evil.com, it can make browser execute a request to bank.com (like a money transfer), and it will be sent with the right cookie. Browser automatically sends the cookies on requests, and can't be sure if the user intentionally made the request or not.

- **Identify vulnerability**: create HTML page separate to the testing web, include something like <img src="http://mysite.com/account/del" width="0" height="0">, log into account in the web and with the session active open the HTML page and check if account has been deleted.

1.Never change data in GET requests, only POST + csrf_token
2.Check for csrf middleware

- **Mitigation**
  - Extra authentication
  - DSRF tokens
  - SameSite cookies: browser setting that prevents cross-site cookie usage (not let evil.com access cookies for bank.com)
  - Referer-based validation: verifies the request originates from own domain
  - Validation via CSRF token: add action token as hidden field in the body of the request to genuine forms, token that will not be sent by the browser accidentally and the attacker can't access through evil.com

Server Side Request Forgery (SSRF)

Attacker writes URL with and embedded URL in it. The second URL would usually not be accessible for the attacker, but from the server it is possible (inside the trust zone).

- **Countermeasures**: no universal fix, highly depends on requirements
  - Whitelist and DNS resolution
  - Response handling
  - Disable unused URL schemas
  - Authentication on internal services
  - Network segregation

Security misconfiguration

XML External Entities (XXE)

<!ENTITY name SYSTEM "URI"> is a way of defining an external resource to be reusable in the document, but can be vulnerable to attackers reading local files, internal services, doing SSRF or DoS with inputs like this:

```
<!DOCTYPE test [
  <!ENTITY xxe SYSTEM "file:///etc/passwd">
]>
<test>&xxe;</test>
```

```
<!ENTITY a "lol">
<!ENTITY b "&a;&a;&a;&a;&a;&a;&a;&a;&a;&a;">
```

- **Countermeasures**:
  - Disable external entity resolution
  - Use safe XML parsers – defusedxml: fromstring(request.body).

Check for views that do uploads or similar (request.FILES[“upload”]) and parse it as XML without filtering or using defusedxml.

Software and data integrity failure

Insecure deserialization

Serialization: turning an object into a data format that can be sent over the network (so python class, for example, to JSON or XML)

Deserialization: converting serialized string back into an object. The problem comes when the server blindly deserializes input from the user and then trusts the resulting object. Attacker modifies the serialized data and then the deserialized object will be executed, leading to SQL injection, Remote Code Execution, Authentication bypass, Data tampering, Privilege escalation.

- **Countermeasures**:
  - Don’t deserialize untrusted input
  - Use data-only formats like JSON
  - Avoid pickle, yaml.load() on input (python libraries to serialize and deserialize data)
  - Validate input manually after deserialization
  - Use digital signatures (to check if content has been tampered with)
  - Run in low-privilege environments

Search for pickle and yaml loads or raw use of deserialized data in code

Identification and authentication failure

Broken authentication

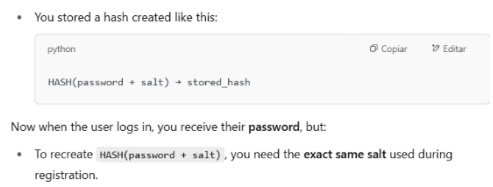
Authentication: process of verifying who you are by

- **Something you know**: passwords & security questions. Easy to implement and understand, but easy to crack and forget
- **Something you have**: bankID device, phone (SMS). Hard to crack, but can be broken, stolen and forged (cloned) and the strength only depends on how difficult it is to forge.
- **Something you are**: biometrics. Hard to crack and to steal, but not always accurate, privacy issues and hard to replace

Crack a password

- Vulnerable password storage: basic but vulnerable, all users compromised if hacker obtains password file.
  - Countermeasure: Hashing – compare hash of password when sent

- Dictionary attack: use words from dictionary and compute the hashes (online or offline)
    - Countermeasure:
        - **Salting** – include additional info in hash, hash password concatenated with salt (a random number) & store salt in password file. Good for online dictionary attack, but ineffective against online.



*Hashing salt does not make sense, because then you can't reverse the salt and then can't verify the password properly

        - **Password Pepper**: like salt but all passwords have the same pepper value, which is stored in an encrypted form in another secure place. Defends better against dictionary attacks and also against offline attacks better.

Other password security techniques

- **Filtering**: guarantee strong password by setting a minimum length, requiring mixed characters and measuring the strength
- **Limiting logins**: allow 3-4 logins and lock account, inconvenient to forgetful user.
- **Aging password**: require to change passwords every once in a while (users will do workarounds and become more insecure)
- **Last login / Protective monitorin**g: notify users of suspicious login, educate users to pay attention and report possible attacks
- **One-time password**: send one-time password through SMS
- **Two-factor/two-channel authentication**: combine different ways of authentication
- **Password recovery** with URL tokens, PINs, Offline methods and security questions

CAPTCHA and reCAPTCHA: Completely Automated Public Turing Test to Tell Computers and Humans Apart. Commonly used to block bots. Machine learning is catching up.

Security logging and monitoring failures

Insufficient logging and monitoring

- Not logged auditable events like logins and transactions
- Unclear/inadecuate/no log messages from warnings and errors
- Logs not monitored for suspicious activity
- Logs only stored locally
- Not effective alerting thresholds
- Unable to detect, escalate or alert for attacks in real time

Insecure design

Clickjacking: attacker can place a layer with a URL that he wants the user to click that seems transparent by setting HTML iframe and opacity.

- Countermeasures
    - X-Frame-Options: deny – completely disables the loading of the page in a frame
    - X-Frame-Options: sameorigin only embed from the same server
    - X-Frame-Options: allow-from – embed only from whitelist

Cryptography

Secure communication: establish a shared secret key (through face-to-face, handshake algorithms…) and then transmit the data using the shared key. Cryptography is the basis for many security mechanisms, but not the solution to all security problems.

Kerckhoff's principle: the encryption algorithm is open, the only secret is the key, that should be chosen at random and kept secret

Step 1. Establish shared secret key

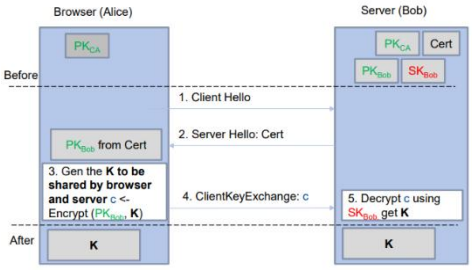
- **Public key encryption**: system where you don't use the same key to encrypt and to decrypt. To encrypt, you use the public key of the receptor, and then only the receptor can decrypt it with his secret key. Mathematically speaking, this happens with a trapdoor one-way function, that is easy to go forward, but not to invert. Y = F(PK,x) easy; F^-1 (SK,y) = x only possible with SK
- **Digital signature**: bind document to author, in digital world it will depend on the content of the document.
  - Bob first hashes* document m
  - Bob signs the Hash(m) using his secret key SK_{Bob} and F^{-1} (here, F is the trapdoor function)

$F^{-1}(SK_{Bob}, Hash(m)) \Rightarrow$ Signature

Alice receives document m from Bob
Alice receives the Signature
Alice receives Bob's public key PK_{Bob}
Alice wants to know if the document m is the one signed by Bob

Check if $F(PK_{Bob}$, Signature) = Hash (m) ?
- Yes: the document is signed by Bob
- No: the document is not signed by Bob

- **Certificate Authority**: third party that ensures that when Bob sends his public key to Alice, it is a verified and real key
- **SSL/TLS handshake**



TLS 1.2 is insecure, use TLS 1.3

RSA – prime factorization, standardized, simple, effective, widely used

ECDSA – limited support (y^2 = x^3 +ax + b), higher complexity, faster, shorter keys and limited support. Uses points on elliptic curves

Step 2. Transmit data using a shared secret key

- **Confidentiality**
  - **Substitution cipher:** letter matrix, each letter defined by letter row and letter column

S	Y	D	W	Z
R	I	P	U	L
H	C	A	X	F
T	N	O	G	E
B	K	M	Q	V

SS TY TD SW = ?

  - **Shift cipher**: associate letter with number, choose one and shift every letter by that position with wraparound. Only 26 possible keys, insecure.

a b c d e f g h i j k l m n o p q r s t u v w x y z
encrypt
d e f g h i j k l m n o p q r s t u v w x y z a b c
decrypt
a b c d e f g h i j k l m n o p q r s t u v w x y z

- To encrypt: **C = (M + K) mod 26**

C: Ciphertext
M: Plaintext
K: Secret key (a single letter that reflects the number of positions to shift)

```
plaintext    helloworld
key          cccccccccc
ciphertext   jgnnqyqtnf
```

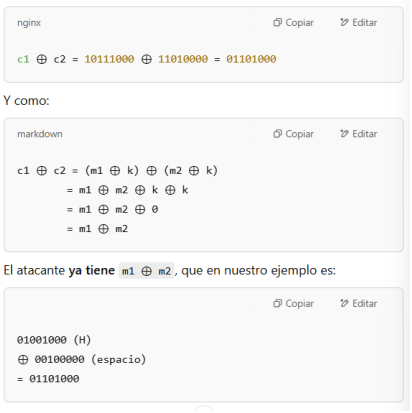
- To decrypt: **M = (C - K) mod 26**

- **The Vigenere cipher**: key is a string, shift each character by the amaunt dictated by the character of the key. Much more keyspace 26^n (n length of key), but also insecure, since you can crack it this way: Guess the length of the key (it is correct if the letters obtained from the same character of the key follow the normal frequency of letters in English) and then guess each character of the key by substituting the most used characters in the text with the most used characters in english.

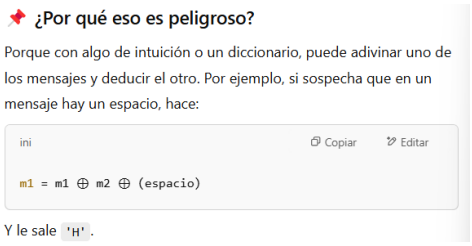
```
plaintext    tellhimaboutme ...
key          cafecafecafeca ...
ciphertext   veqpjiredozxoe ...
```

(encrypt HOLA with key LUNA : (H)7+(L)11 = 18 = S; 14(O)+20 (U) = 34 mod 26 = 8 = I ...

- **One time pad**: use random key of the same size of the plaintext used only once, with the same logic as Vigenère. Perfectly secure if used correctly, but key must be as long as plaintext and be used once. It has no integrity protection, if something is change in the cipher, it's changed in the plain too.

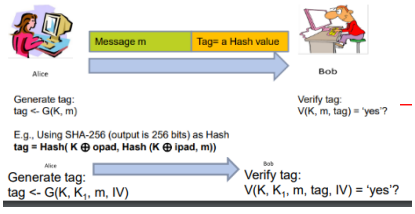


Basically, with the XOR, the attacker can know the differences between the two original texts, and by applying some mechanisms on the similar parts they can identify both messages



- **Stream cipher**: use short key as seed to generate a pseudo random key as long as the message to make OTP encryption
- **Block cipher**: cut text into blocks and encrypt each using short keys. In DES (Data Encryption Standard) you have each block 64 bits, k=56 bits and 16 rounds, and its reversible round function. Applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output. Then, rightmost 32 bits are swapped with leftmost 32 bits
  - ECB (Electronic code book) is wrong, it can disclose plaintext information (same blocks represented same way)
  - CBC (Cipher Block Chaining) with random initialization vector, each block XORs with the previous before ciphering.

- **Integrity**
  - **MAC** (Message Authentication Code): with message and shared secret key a MAC function can be applied to generate a tag sent with the message. The receiver then can recalculate the MAC with the same key and compare the result (if it is the same, its integrity hasn't been modified)
  - **HMAC** - uses hash functions like SHA-256 or SHA-3
  - **ECBC** – Enhanced CBC mode that adds an integrity verification system.



- **Combination**
  - MAC then Encrypt (TLS)
  - Encrypt and MAC (SSH)
  - Encrypt then MAC (IPsec)

Authorization, Authentication, Multi-Level, SSO…

Authorization and MultiLevel Security

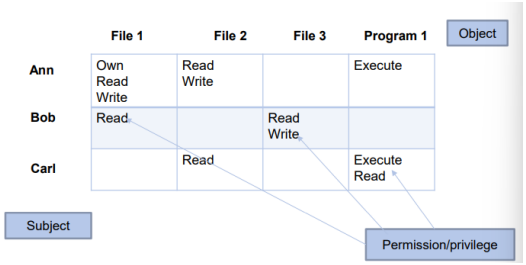
Access control: determines which entities (users, programs, machines) can access specific resources (files, memory, services). Initially physical in nature, access control evolved to handle multiple programs and users interacting with a shared system.

Levels

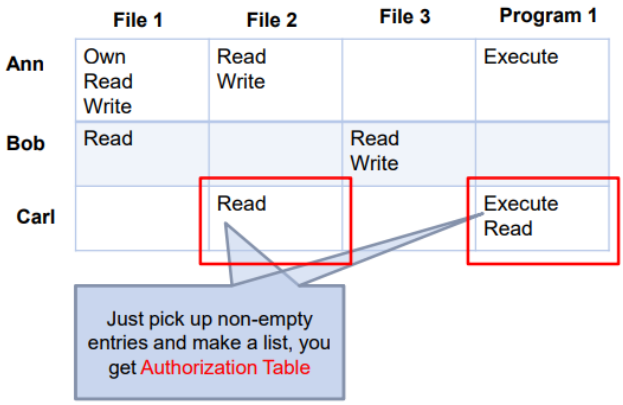
- Application: complex and context-sensitive rules
- Middleware: enforce consistency (databases ensuring transaction integrity)
- Operating system: manage file and resource access
- Hardware: uses CPU and memory management units to restrict memory access

Models

- **Discretionary access control (DAC)**: owner of a resource decides how it can be shared (choose to give read, write, or other access to users. Does not distinguish between user and process, so an attacker can make a trojan horse (process executing malicious programs) that can exploit the authorization of the user (weak control over information flow)
    - Access control matrix

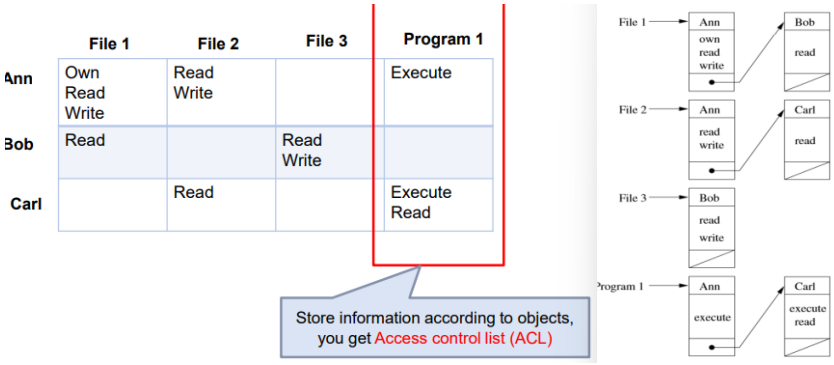


    - Authorization table

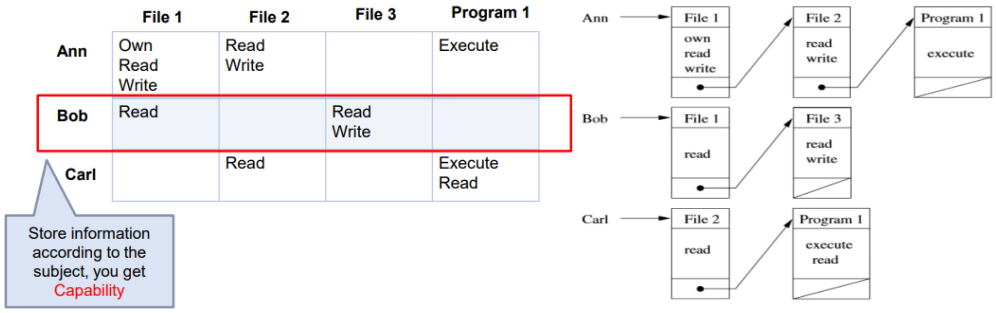


USER	ACCESS MODE	OBJECT
Ann	own	File 1
Ann	read	File 1
Ann	write	File 1
Ann	read	File 2
Ann	write	File 2
Ann	execute	Program 1
Bob	read	File 1
Bob	read	File 3
Bob	write	File 3
Carl	read	File 2
Carl	execute	Program 1
Carl	read	Program 1

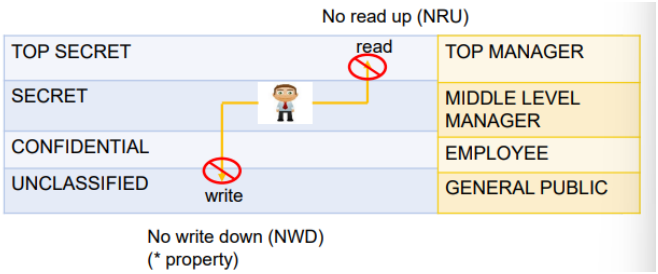
    - Access control list (ACL) – used in modern OS



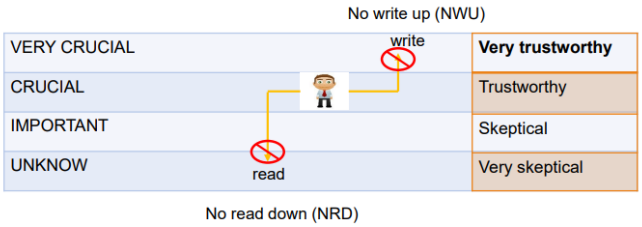
- Capabilities



- **Mandatory access control (MAC)**: system enforces a security policy independent of the user's action (in DAC users could take their own permissions), basis of regulations by central authority. The access class is assigned to each object and subject. Cumbersome administration.
  - **Bell-LaPadula model** – no read up and no write down. This prevents the Trojan horse attempts to copy secret data into an unclassified file (confidentiality)



  - **Biba model**: no write up and no read down. No improper modification of high-integrity objects and not contaminated due to reading and using unreliable data.



  - **Combination**: objects and subjects have to be assigned to two access classes (one for confidentiality and one for integrity).
- **Role-based access control (RBAC)**: method to manage user permissions in complex systems based on roles instead of individual entities. Easy authorization management and it maps to real-world role hierarchy. Coarse-grain access control
- **Attribute-based access control (ABAC)**: access decisions based on attributes like department, owner of assets, time, location, device type… and a policy/rule engine evaluates whether access is allowed based on these attributes. Fine-grain access control

Authentication and Single sign-on

Without SSO: user must login separately to each domain and it stores a separate cookie in the browser, and due to same-origin policy, cookies from one domain can't be read or reused by another domain.

With SSO: user logs into a service provider that redirects to a central identity provider, user enters credentials once and the central IdP sends back confirmation. This allows user to access multiple services without having to log in again

SSO trends:

- Moving from SOAP/XML to HTTP/JSON
- Adoption of social login (Google, Facebook)
- Standard protocols
    - OpenID Connect for authentication: ID token, logging in (SSO), Who are you?
    - OAuth 2.0 for authorization: access token, accessing APIs/data, What can you do?

Scenario	Token Type	Purpose
Traditional web app (PHP, Django)	Session token	Identify user across page loads
SPA (React, Angular) + API	Access token	Authenticate API calls
SSO system with OAuth	Access + ID tokens	Login once, access multiple services

Key Steps:

1. **App asks for permission** → redirects you to the real login page (e.g., Facebook).
2. **You log in** and approve access.
3. **Facebook sends back a short-lived code** to the app.
4. **App exchanges the code for an access token** (securely, from the backend).
5. **App uses the access token** to access your data (e.g., friends list).

Why It's Secure:

- App never sees your **password**.
- Tokens are handled **server-side**.
- **Authorization and access are separate**

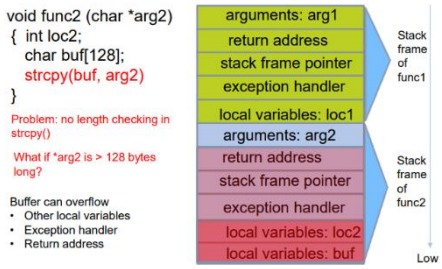
Control hijacking attacks

The attacker's goal is to take over the target machine and execute arbitrary code by hijacking application control flow. Compromise confidentiality, integrity and availability. Targets mainly C/C++ code.

Attackers use buffer overflows to hijack control, the attacker fills a buffer and overwrites nearby memory, changing the program's behavior.

These attacks can go undetected if done early since the bug is unknown to the vendor at the beginning.

How does buffer overflow work: in the memory layout you have the Stack (function return addresses and variables) and the heap (objects and function pounters), which are working at runtime, so they are the attacker's focus



An attacker could overwrite a variable that determines access control (not even a need to execute code)

An attacker could overwrite the return address, pointing to malicious code they injected that then will be executed

An attacker could read past a buffer and get sensitive information (heartbleed bug)

- Countermeasures:
    - Always use safe functions: UNSAFE FUNCS (strcpy, strcat, gets, strncpy...)
    - Leverage defenses in compilers
    - Check length when read/write buffer
    - Use tools to audit source code
    - Rewrite software in type-safe language
    - Type-safe languages help, since the programmer is no longer in charge of defining the size of the variable.

Threat modeling and STRIDE

Threat modeling: a way of imagining the vast vulnerability landscape of a system and ways to attack it. Look at a system from an adversary's perspective to anticipate attack goals.

Why? To understand and document a system's threat environment, discover possible weaknesses early, how to best spend the security budget…

When? During the design phase after establishing security requirements and attack surface analysis, and before implementation. Continuous refinement, early and frequent analysis and aligned to the organization's development practices.

- **McGraw**'s model emphasizes that security is not one step, but many interwoven actions throughout development, and threat modeling aligns with requirements (abuse cases) and risk analysis .
- In **Agile**, threat modeling should be integrated into each agile cycle. Project inception (high-level threats), requirements planning (threats with higher impact), sprint planning (where are the threats), sprint execution (develop, update and complete), final release planning (complete models)

Who? Security experts, developers and engineers, stakeholders and team members, and attackers.

- Anti-Pattern: Hero Threat modeler - don't rely on "that one smart person" to do the threat modeling, everyone should contribute.
- Pattern: Varied viewpoints - diverse team with appropriate subject matter experts and cross-functional collaboration.

How? There is no single best or correct way of performing threat modelling, it is a question of trade-offs and what we want to achieve by doing it.

- Anti-Pattern: Perfect representation – better to create multiple TM representations, there is no single ideal view.

Threat agents

By identifying likely attackers, we can prioritize the most realistic threats and apply appropriate countermeasures

- The spooks: government intelligence services. High skills and salary.
- Cyber Warriors: military/government offensive units. High skills and salary.
- The crooks: cybercriminals. Profit-driven, efficient. Jobless, with a lot of motivation.
- The geeks: curious hackers, tinkerers. May exploit for fun or curiosity
- The terrorists: politically motivated, destructive. Unpredictable, potentially targeting infrastructure. No skills but highly motivated.
- CEO criminals: sophisticated fraud rings. Deeply strategic, targeted attacks, insiders
- The swamp: trolls with some skill and some motivation to make some fun
- The insiders: people who want to expose something, not a lot of skills or money. Mostly people who are fired.

Software centric threat models

Models that focus on the software being built or a system being deployed. Pattern: Systematic approach – achieve thoroughness and reproducibility by applying security and privacy knowledge in a structured manner

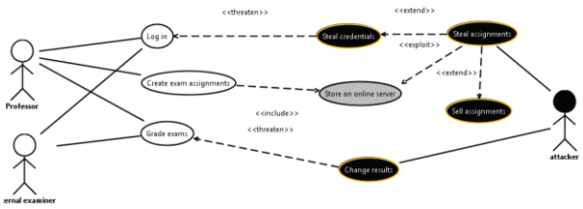
1. Identify critical assets
2. Decompose the system to be assessed
3. Identify possible points of attack
4. Identify threats
5. Categorise and prioritise the threats
6. Mitigate

STRIDE

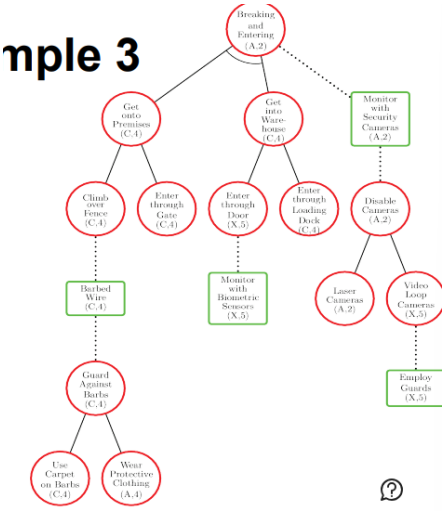
- Spoofing: pretending to be something/someone you're not (fake webs, emails, CSRF…)
- Tampering: modifying something you're not supposed to modify (forms, URLs, files, databases…)
- Repudiation: claiming you didn't do something regardless whether you did it or not (have not received, attacking logs, use someone else's account…)
- Information disclosure: exposing info to people who are not authorized to seeit (steal files/database contents, eavesdrop network data, system/api information…)
- Denial of Service: attacks designed to prevent a system from providing service (network flooding, crashing software, making systems slow, filling storage…)
- Elevation of Privilege: program or user technically able to do things that they're not supposed to do (XSS, buffer overflow, injection attacks, modify access control, social engineering…)

Notations

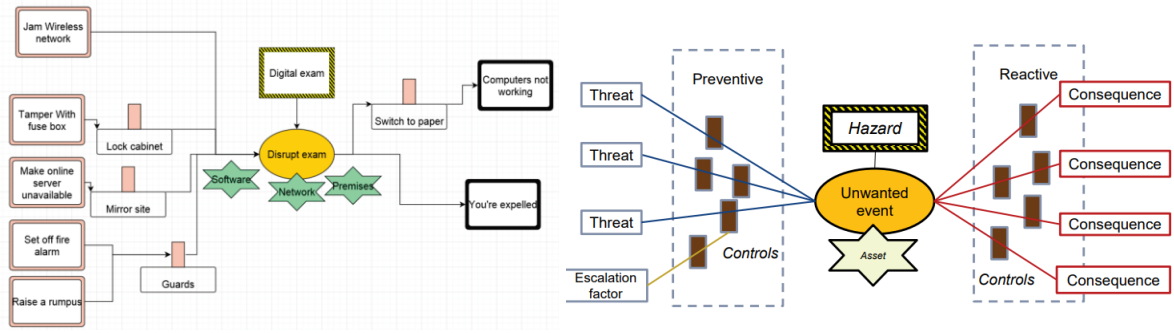
- Misuse cases: extends UML use cases, high-level negative scenarios, easy to grasp by stakeholders



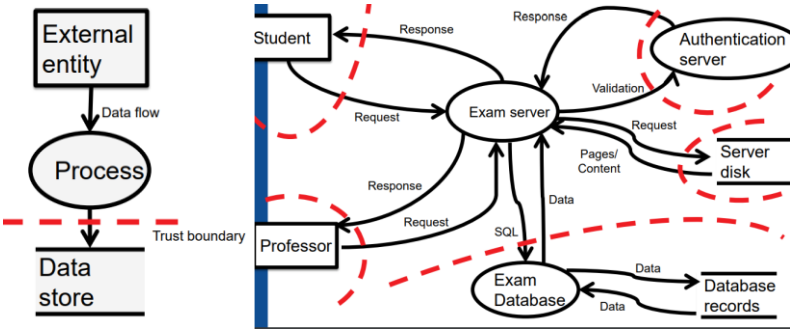
- Attack trees: possible ways of achieving an attack goal, tree with AND/OR nodes, more technical than misuse cases. Attributes: cost, detectability, difficulty, impact, penalty, profit, probability, special skill, time



- Bow-tie diagram: model a single unwanted event at a time, different causes/threats to unwanted events, different consequences once the event has happened, preventive/reactive controls, tradition from safety



- Data Flow Diagram: understand the system, dataflow between subsystems, find attack surface and critical components, define trust/privilege boundaries.



Risk Management during development

Risk Management Framework

Continuous and iterative process to identify, rank, track and understand software security risk as it changes over time

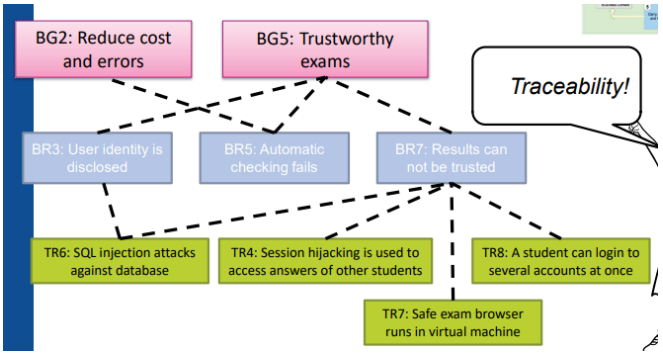
1. Understanding the business context: business goals (circumstances to care about and risk scales), business assets (what are you trying to protect) and stakeholders (who cares – users, regulators, attackers…)
    a. Risk dimensions and scales: likelihood and impact/consequence

Low	Medium	High	Extreme
Once every 10 years or less	Once per year or less	Once per month or less	Every week

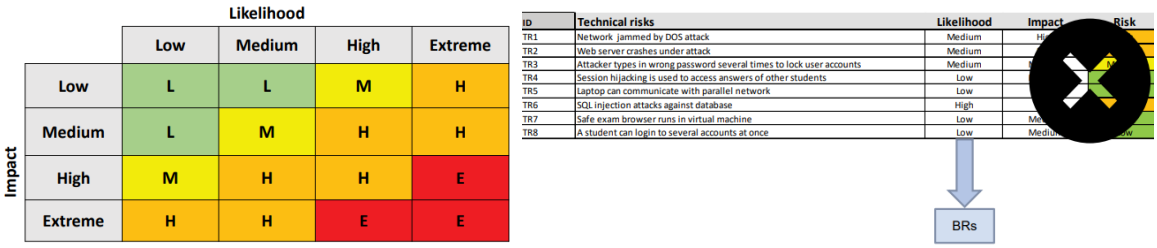
Dimension	Low	Medium	High	Extreme
Confidentiality	No or minimal exposure of internal information or individual personal data.	Exposure of internal information or individual personal data.	Exposure of confidential information or sensitive or personal data of many.	Exposure of secret information or all personal data.
Availability	Tasks can be performed with delays or poorer quality.	Unsatisfactory quality or severe delays.	Limited ability to perform tasks.	Not possible to perform critical tasks.
Financial	Lesser economic loss that can be restored.	Significant economic loss that can be restored.	Irreperable economic loss	Significant and irreperable economic loss
Reputation	No loss of reputation and little influence on trust.	Reputation and trust can be damanged.	Damage to repuatation, serious loss of trust.	Serious damage to reputation and trust.

2. Identify business risks: threaten directly one or more of a customer's business goals. Usually related to data, time, money, reputation and legal. Example (too difficult to use, unavailable, failure at checks, too expensive…)
3. Identify technical risks and link them with business risks: technical risks are threats and attacks that may bring negative impacts on business (how it happens). Inputs to identify: documents, user feedback, testing and threat intelligence. Tools: misuse cases, attack trees,

DFDs… (Here is where threat modeling comes into scene, that is how you identify technical risks) Examples (web crashes, network jammed by DOS attack, SQL injection…



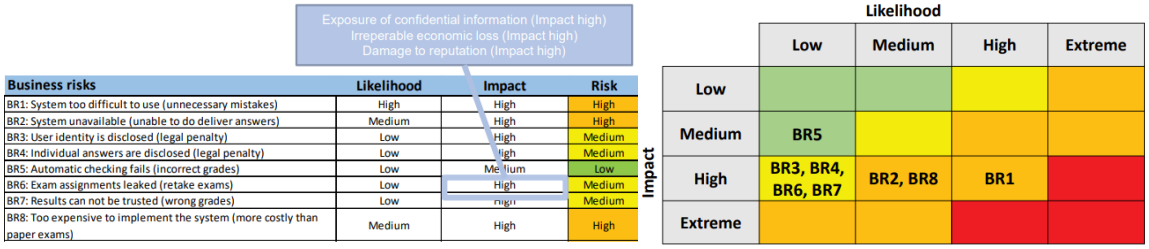
4. Synthesize and prioritize risks:



Impact \ Likelihood	Low	Medium	High	Extreme
Low	L	L	M	H
Medium	L	M	H	H
High	M	H	H	E
Extreme	H	H	E	E

ID	Technical risks	Likelihood	Impact	Risk
TR1	Network jammed by DOS attack	Medium		
TR2	Web server crashes under attack	Medium		
TR3	Attacker types in wrong password several times to lock user accounts	Medium		
TR4	Session hijacking is used to access answers of other students	Low		
TR5	Laptop can communicate with parallel network	Low		
TR6	SQL injection attacks against database	High		
TR7	Safe exam browser runs in virtual machine	Low		
TR8	A student can login to several accounts at once	Low		

BRs



Business risks	Likelihood	Impact	Risk
BR1: System too difficult to use (unnecessary mistakes)	High	High	High
BR2: System unavailable (unable to do deliver answers)	Medium	High	High
BR3: User identity is disclosed (legal penalty)	Low	High	Medium
BR4: Individual answers are disclosed (legal penalty)	Low	High	Medium
BR5: Automatic checking fails (incorrect grades)	Low	Medium	Low
BR6: Exam assignments leaked (retake exams)	Low	High	Medium
BR7: Results can not be trusted (wrong grades)	Low	High	Medium
BR8: Too expensive to implement the system (more costly than paper exams)	Medium	High	High

Impact \ Likelihood	Low	Medium	High	Extreme
Low				
Medium	BR5			
High	BR3, BR4, BR6, BR7	BR2, BR8	BR1	
Extreme				

5. Define the risk mitigation strategy: reducing the likelihood, the severity of impacts and derive security requirements
    a. Security requirements: statement of needed security functionality that ensures one of many security properties of software being satisfied. What you require, not how to achieve it (understandability, clarity, cohesion, testability.

Examples of security requirements

Technical risks	Security requirements
TR3: Attacker types in wrong password several times to lock user accounts	Two-factor authentication should be required. Logs should contain source and results of login attempts.
TR6: SQL injection attacks against database	User inputs should always be validated and sanitized.

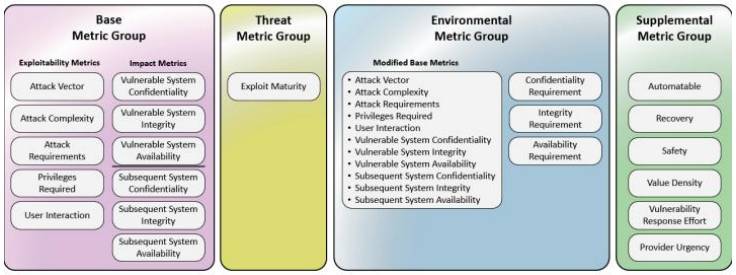
6. Carry out fixes and validate: implementation of mitigation strategies and risk-based testing (focused on security requirements)

Risk quantification

In security we assume a hostile opponent who can cause failure at the least convenient time and in the most damaging way possible. Breaches are inevitable because defenders have to be right 100% of the time, whereas attackers only have to be right once.

CVSS (Common vulnerability scoring system): standardized way of measuring the technical severity of a vulnerability. NOT a direct risk value by itself.

- Base: constant over time and across user environments
- Threat: characteristics of a vulnerability that change over time
- Environmental: unique to a user's environment
- Supplemental: do not modify the final score, gives additional insight.



High-level overview

1. Understanding business context
   - ¿Qué queremos proteger? ¿Quién se ve afectado?

2. Identify business risks
   - Riesgos que impactan en los objetivos del negocio (datos, dinero, reputación

3. Identify technical risks
   - Aquí entra el Threat Modeling:
     - Paso 1: Identificar activos críticos
     - Paso 2: Descomponer el sistema (DFD)
     - Paso 3: Identificar puntos de ataque
     - Paso 4: Identificar amenazas (STRIDE)

4. Link technical risks with business risks
   - ¿Qué impacto tendría cada amenaza técnica en el negocio?

5. Synthesize and prioritize risks
   - Evaluar impacto × probabilidad
   - Coincide con el paso 5 del modelo software-centric

6. Carry out fixes and validate
   - Aplicar mitigaciones, definir requisitos de seguridad, validar
   - Coincide con el paso 6 del modelo software-centric

Static Analysis and Tools for Security

- Weaknesses and vulnerabilities
  - **Weaknesses**: errors in software implementation, code, design or architecture that if left unaddressed could result in vulnerabilities
  - **Vulnerabilities**: mistake/weakness in software that can be directly used by an attacker to gain access to a system or network
  - **Exploit**: piece of software containing attack vectors that could be directly used to take advantage of a vulnerability
- What and why static code analysis
  - **What is static analysis**: passive scanning of application code without executing it. A source code security analyzer examines source code to detect and report weaknesses that can lead to security vulnerabilities (results in a report form, integrated in IDEs and CI/CD)
  - **Why**: because it can catch security defects early in the software development life cycle (already in the implementation, significant for code review)
- **Static Application Security Tools**
  - How does the program analyzer work
    - Source code: Source code parsed into Abstract syntax tree transformed into Control Flow Graph and then data flow analysis over it
    - Byte code
    - Binarial
  - Techniques for static code analysis
    - Model construction for source code analysis (lexical analysis, semantic analysis, control flow analysis)
    - Security knowledge
    - **Pattern matching** (report security issues if patterns are found in the code)
    - **Control flow analysis** (construct graph for if, while, for, switch, exceptions and cover all through analysis)
    - **Data flow analysis** (analyse data based on control flow graph)
      - Taint analysis (source, exit and sanitization points). If tainted data (untrusted) reaches a dangerous sink, a security issue may occur

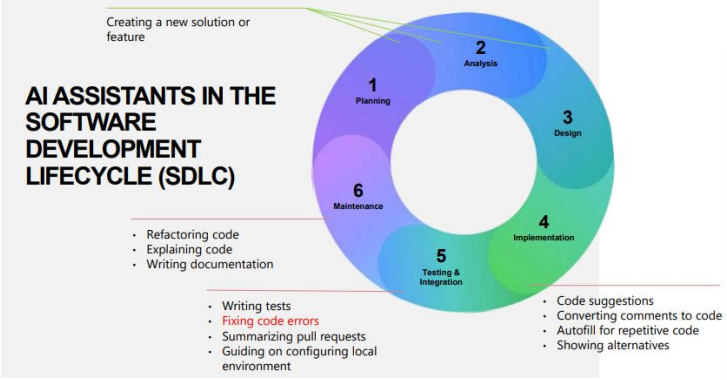
  - **Performance metrics**: to measure how good a SAST tool perform
    - **Soundness**: sound if never overapproximates the set of bugs in a given program
    - **Completeness**: complete if never underapproximates the set of bugs in a given program

Penetration testing

- Steps for a successful web application test
  - Understand the application you are testing
  - Quickly check for low hanging fruit (automated scanners or tools for the clearer things)
  - Dive deep into what makes sense

- Don’t get stuck in the mindset of only checking for vulnerabilities, or relying on tools. They can provide a false sense of security if not fully understood. Easy to make stuff without knowing how it works.
- Pen testing: type of security assessment where someone simulates an attack on a system before real attackers do.
    - Don’t hack things without permission, don’t go out of scope, do “ethical” testing
    - (1) Reconnaissance: check public information, try to decompile the code, look at network traffic. Learn a lot just by observing how the application work and then start to dive into the potential weaknesses
    - Try to step on the good side, there are plenty of regulation stuff that changes
    - (2) Download code and attack against local installation, passively listening to traffic, broadcast beacons without interfering with others and report findings to the organizations
        - BAD: active burp scans against servers, test wormhole attacks, broadcast beacons that interfere and sell exploits on the dark web

Secure coding with LLMs



- Coding assistants can:
    - Automate repetitive tasks
    - Reduce cognitive load
    - Accelerate navigation of unfamiliar languages/frameworks
    - Maintain “flow state” during development
    - Democratize coding expertise
- Potential risk categories
    - Sensitive data leaks
    - Suggesting vulnerable code
    - Overlooking security
    - Training data from sources with vulnerable code
    - Missing security context in unfamiliar data domains and has limited awareness of environmental security requirements
    - Models themselves being vulnerable to attack and manipulation
    - Developer overreliance on AI (trust too much)
- Key developer practices
    - Choose AI assistant wisely
    - Apply secure prompt engineering
        - Explicit security requirements in prompts, framework-specific security guidance, context setting and example-driven prompting
    - Add real time vulnerability detection tools
    - Embed security in development workflow
    - Human-in-the-loop validation
- Key organizational practices
    - Implement AI-aware security toolchains
    - Develop clear security standards for AI-generated code
    - Provide security training for AI tool users
    - Establish accountability frameworks for AI contributions
    - Monitor and iterate on security processes

Privacy by design

Data privacy: right to privacy and right to decide when someone can use your own personal data, interdisciplinary. This is regulated by the General Data Protection Regulation (GDPR) + other relevant laws depending on domains. Law does not distinguish between what is processed by IT/ manually. The point is not to avoid using personal data, but to use it correctly in the correct context.

Challenge: to secure high quality privacy assessments, you need relevant domain expertise, else the lack of understanding will result in false safety and lack of compliance with the Personal Data Act.

Important definitions:

- Personal Data: any information related to a natural person, one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person.
- Processing: operation or set of operations performed on personal data, whether or not by automated means (collection, storage, alteration, use, destruction…)
- Special categories of personal data: ethnic origin, political opinions, religious beliefs, genetic data, health, sex life or orientation shall be prohibited to process, except under very specific situations
- Roles:
    - Controller: natural or legal person, authority or body which determines the purposes of the processing of personal data and the means to be used. (A hospital decides to collect and manage patient health records. The hospital decides what information is needed and how it will be used)
    - Processor: natural or legal person, authority or body which processes personal data on behalf of the controller (the hospital hires an IT company to host and maintain the patient database. That company stores and secures the data but does not decide what data is collected or how it is used)
    - Recipient: natural or legal person, authority or body to which the personal data are disclosed, whether a third party or not (the hospital sends patient data to a national health authority as part of a legal obligation).

Data Protection Impact Assessment

Risk assessment tool required under GDPR to help organisations identify and minimize the privacy risks of their data processing activities. Formal document that many business avoid or delay because they think it's complex, but it does not need to be long or complicated. It is mandatory when data processing is likely to result in a high risk.

Challenge: GDPR is cryptic for those who develop digital products, since it is hard to translate GDPR requirements into process and solution.

Processing of personal data (iterative process):

- **Describe the processing of personal data**
    - Identify the purpose: (subprocess) or service that uses personal data that should have business focus and should be split into smaller and more specific during the development cycle. Needs domain expertise to be properly defined

Purposes - Some examples (2/2)

- **Student loans – example how to granulate purposes**
  - First proposal (Coarse-grained purpose): **Handle grants and student loans including applications**
    - Only one coarse-grained purpose (including most of the business of Lånekassen) is in this context probably a too broad approach. Increased insight will probably show that several more fine-grained purposes is a better idea.
  - Refined proposal: **Apply for grants and student loans and get a decision**
    - This proposal is one of several more fine-grained purposes related to the business of Lånekassen
    - This proposal also have a more end-user focus
- **Graduation high school / exams – another example of how to granulate purposes**
  - Planning and implementation of exams for students in senior high school
    - As a purpose, this is too broad because the individual municipality (school owner) is responsible (and thus responsible for processing) for oral exams and exams in vocational subjects, while the Directorate of Education (UDI) is responsible for written subjects in study specialization
  - This high level propose needs to be split info at least two fine-grained purposes because of two different controllers even if the same IT-system supports all exams mentioned
    - Planning and implementation of exams when UDI is responsible
    - Planning and implementation of local exams for each the school owner

- Identify the scope of the processing
- Define sources, recipients and responsibilities
- Should be understandable for all parties and gradually elaborated over several iterations.
- Do we need a complete DPI? Validate High-risk:

Check list – "high risk" or not

A Complete formal DPIA is needed if 2 or more criteria's are met

Does the processing of Personal data include

1. Evaluation or scoring
2. Automated-decision making with legal or similar significant effect
3. Systematic monitoring of the data subjects
4. Sensitive data or data of a highly personal nature
5. Data processed on a large scale
6. Matching or combining datasets
7. Data concerning vulnerable data subjects
8. Innovative use or applying new technological or organizational solutions
9. Prevent data subjects from exercising a right or using a service or a contract

Description processing: Example – Apply for student loan

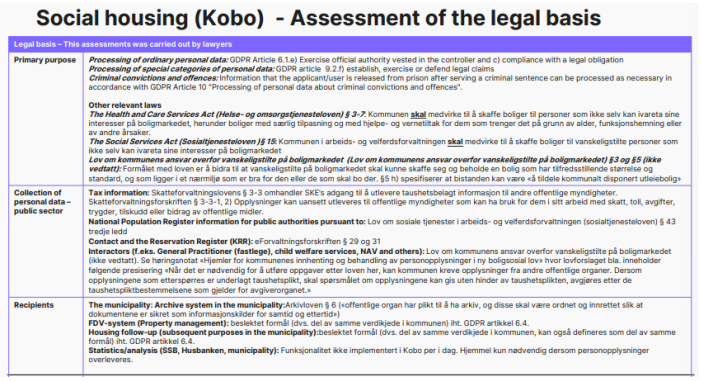
Purpose	The nature and context of processing (High level functional description)	Categories of Personal Data per category of Data ubjects
Apply for student loans and grants from Lånekassen	After logging in to Lånekassen's service, the applicant fills in an electronic application for grants and student loans, and in addition to central contact information, the place of study and course of study, etc. Lånekassen collects/verifies information from the university listed, the National Population Registe (Folkeregistert) and other relevant sources. Based on a defined set of rules for the allocation of grants and student loans, Lånekassen's loan/application solution automatically processes most applications (>80%) and executes a decision (executed automatically). In the remaining cases where an automatically decision cannot be made, the case is assigned to a case officer who does an assessment and decides The applicant is immediately informed of the decision by e-mail/text message. The decision itself is available to the applicant when logging in to Lånekassen's solution. The automatic processing is carried out in the public cloud, the information that is collected/stored is encrypted, but during the processing itself, the information is in plain text.	**Lånesøker/-kunde:** Navn og -kontaktopplysninger, kundenr, studieopplysninger, låneopplysninger, saks- og vedtaksopplysninger **Saksbehandler:** Rolleopplysninger, saker under behandling, utførte saker

The scope of the processing	Responsibilities	Sources for the collection of personal data
200.000 søknader behandles hvert år, to ganger i året (vår og høst hvorav vår utgjør 80%) Utenlandske statsborgere utgjør 10% av søknadene. 15% er norske statsborgere med studiested i utlandet. 5% av søknadene innvilges ikke	Lånekassen er behandlingsansvarlig: Følgende er databehandlere: <liste> Databehandleravtale er inngått (inngås) med disse	Søknadsinformasjon innhentes fra søker Informasjon om studiested hentes fra studiested (integrasjon – online oppslag) Informasjon innhentetes også fra bla Folkeregisteret (integrasjon – online oppslag

Recipients	High level technical description	Test data
Andre (offentlige myndigheter): Beskriv årsak og hva som overføres og hvor ofte For grunnleggende driftsløsning basert på public cloud, så er leverandør (navgi) å definere som uten for EU/EØS i et land med et personvernlovverk som ikke er likeverdig GDPR	Egenutviklet fagapplikasjon, .net. Opprinnelig dedikert infrastruktur Kjører nå i public cloud etter «Lift-and-Shift» <Si noe om gjeldende sikkerhetsmodell>, Tilgangsstyring, Teknisk logg og forretningslogg	Syntetiske testdata basert på syntetiske f.nr (fra Folkeregisteret)

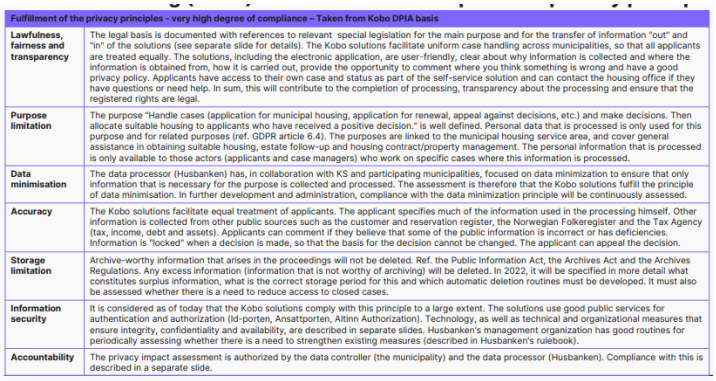
- **Assess compliance with the privacy principles**: inherent part of the digital product, identify and implement the necessary technical measures to comply with each of the principles.
  - **Lawfulness, fairness and transparency**: compliance with privacy principles as an inherent part of the digital product. Ensure real co-determination, real transparency and predictable treatment.
    - Consent, performance of contract, compliance with legal obligation, vital interests, public interest…
  - **Purpose limitation**
    - Identify the most appropriate purposes that focus on end-user-oriented services
    - Verify that personal data processed is not used in any context other than what is necessary and lawful

        - Role-based access control is a good basic measure to ensure purpose limitation
        - Verify that all processing included are necessary and lawful
    - **Data minimization**: verify that only information that is necessary for the purpose is collected
    - **Accuracy**: personal data processed is correct and the result of processing is correct (also in a year or two)
    - **Storage limitation**: consider storage periods, storage time, exceptions, ask for analysis and statistics…
    - **Information security**: consider measures to ensure confidentiality, integrity and availability

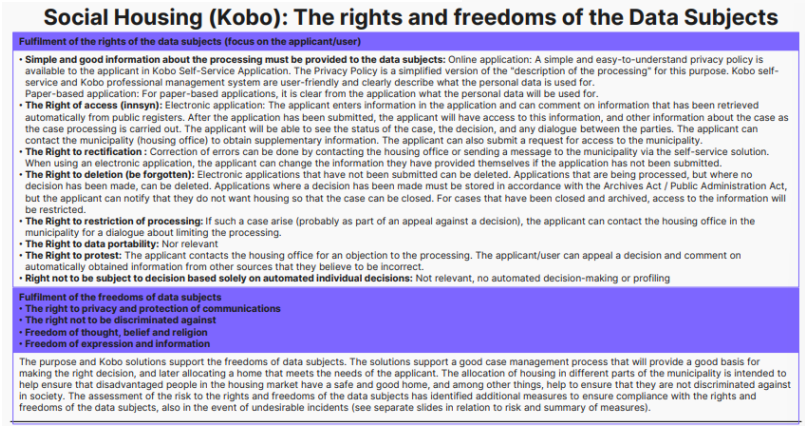


Social housing (Kobo) - Assessment of the legal basis

Legal basis – This assessments was carried out by lawyers	
Primary purpose	***Processing of ordinary personal data:*** GDPR Article 6.1.e) Exercise official authority vested in the controller and c) compliance with a legal obligation ***Processing of special categories of personal data:*** GDPR article 9.2.f) establish, exercise or defend legal claims ***Criminal convictions and offences:*** Information that the applicant/user is released from prison after serving a criminal sentence can be processed as necessary in accordance with GDPR Article 10 "Processing of personal data about criminal convictions and offences". **Other relevant laws** ***The Health and Care Services Act (Helse- og omsorgstjenesteloven) § 3-7.*** Kommunen **skal** medvirke til å skaffe boliger til personer som ikke selv kan ivareta sine interesser på boligmarkedet, herunder boliger med særlig tilpasning og med hjelpe- og vernetiltak for dem som trenger det på grunn av alder, funksjonshemming eller av andre årsaker. ***The Social Services Act (Sosialtjenesteloven) §§ 15*** Kommunen i arbeids- og velferdsforvaltningen **skal** medvirke til å skaffe boliger til vanskeligstilte personer som ikke selv kan ivareta sine interesser på boligmarkedet ***Lov om kommunens ansvar overfor vanskeligstilte på boligmarkedet (Lov om kommunens ansvar overfor vanskeligstilte på boligmarkedet) §3 og §5 (ikke vedtatt):*** Formålet med loven er å bidra til at vanskeligstilte på boligmarkedet skal kunne skaffe seg og beholde en bolig som har tilfredsstillende størrelse og standard, og som ligger i et nærmiljø som er bra for den eller de som skal bo der. §5 hj spesifiserer at bistanden kan være «å tildele kommunalt disponert utleiebolig»
Collection of personal data – public sector	**Tax information:** Skatteforvaltningslovens § 3-3 omhandler SKE's adgang til å utlevere taushetsbelagt informasjon til andre offentlige myndigheter. Skatteforvaltningsforskriften § 3-3-1, 2) Opplysninger kan uansett utleveres til offentlige myndigheter som kan ha bruk for dem i sitt arbeid med skatt, toll, avgifter, trygder, tilskudd eller bidrag av offentlige midler. **National Population Register information for public authorities pursuant to:** Lov om sosiale tjenester i arbeids- og velferdsforvaltningen (sosialtjenesteloven) § 43 tredje ledd **Contact and the Reservation Register (KRR):** eForvaltningsforskriften § 29 og 31 **Interactors (f eks. General Practitioner (fastlege), child welfare services, NAV and others):** Lov om kommunens ansvar overfor vanskeligstilte på boligmarkedet (ikke vedtatt). Se høringsnotat «Hjemler for kommunenes innhenting og behandling av personopplysninger i ny boligsosial lov» hvor lovforslaget bla. inneholder følgende presisering «Når det er nødvendig for å utføre oppgaver etter loven her, kan kommunen kreve opplysninger fra andre offentlige organer. Dersom opplysningene som etterspørres er underlagt taushetsplikt, skal spørsmålet om opplysningene kan gis uten hinder av taushetsplikten, avgjøres etter de taushetspliktbestemmelsene som gjelder for avgiverorganet.»
Recipients	**The municipality: Archive system in the municipality:** Arkivloven § 6 («offentlige organ har plikt til å ha arkiv, og disse skal være ordnet og innrettet slik at dokumentene er sikret som informasjonskilder for samtid og ettertid») **FDV-system (Property management):** beslektet formål (dvs. del av samme verdikjede i kommunen) iht. GDPR artikkel 6.4. **Housing follow-up (subsequent purposes in the municipality)** beslektet formål (dvs. del av samme verdikjede i kommunen, kan også defineres som del av samme formål) iht. GDPR artikkel 6.4. **Statistics/analysis (SSB, Husbanken, municipality):** Funksjonalitet ikke implementert i Kobo per i dag. Hjemmel kun nødvendig dersom personopplysninger overleveres.



Fulfillment of the privacy principles - very high degree of compliance – Taken from Kobo DPIA basis	
Lawfulness, fairness and transparency	The legal basis is documented with references to relevant special legislation for the main purpose and for the transfer of information "out" and "in" of the solutions (see separate slide for details). The Kobo solutions facilitate uniform case handling across municipalities, so that all applicants are treated equally. The solutions, including the electronic application, are user-friendly, clear about why information is collected and where the information is obtained from, how it is carried out, provide the opportunity to comment where you think something is wrong and have a good privacy policy. Applicants have access to their own case and status as part of the self-service solution and can contact the housing office if they have questions or need help. In sum, this will contribute to the completion of processing, transparency about the processing and ensure that the registered rights are legal.
Purpose limitation	The purpose "Handle cases (application for municipal housing, application for renewal, appeal against decisions, etc.) and make decisions. Then allocate suitable housing to applicants who have received a positive decision." is well defined. Personal data that is processed is only used for this purpose and for related purposes (ref. GDPR article 6.4). The purposes are linked to the municipal housing service area, and cover general assistance in obtaining suitable housing, estate follow-up and housing contract/property management. The personal information that is processed is only available to those actors (applicants and case managers) who work on specific cases where this information is processed.
Data minimisation	The data processor (Husbanken) has, in collaboration with KS and participating municipalities, focused on data minimization to ensure that only information that is necessary for the purpose is collected and processed. The assessment is therefore that the Kobo solutions fulfill the principle of data minimisation. In further development and administration, compliance with the data minimization principle will be continuously assessed.
Accuracy	The Kobo solutions facilitate equal treatment of applicants. The applicant specifies much of the information used in the processing himself. Other information is collected from other public sources such as the customer and reservation register, the Norwegian Folkeregister and the Tax Agency (tax, income, debt and assets). Applicants can comment if they believe that some of the public information is incorrect or has deficiencies. Information is "locked" when a decision is made, so that the basis for the decision cannot be changed. The applicant can appeal the decision.
Storage limitation	Archive-worthy information that arises in the proceedings will not be deleted. Ref. the Public Information Act, the Archives Act and the Archives Regulations. Any excess information (information that is not worthy of archiving) will be deleted. In 2022, it will be specified in more detail what constitutes surplus information, what is the correct storage period for this and which automatic deletion routines must be developed. It must also be assessed whether there is a need to reduce access to closed cases.
Information security	It is considered as of today that the Kobo solutions comply with this principle to a large extent. The solutions use good public services for authentication and authorization (Id-porten, Ansattporten, Altinn Authorization). Technology, as well as technical and organizational measures that ensure integrity, confidentiality and availability, are described in separate slides. Husbanken's management organization has good routines for periodically assessing whether there is a need to strengthen existing measures (described in Husbanken's rulebook).
Accountability	The privacy impact assessment is authorized by the data controller (the municipality) and the data processor (Husbanken). Compliance with this is described in a separate slide.



Social Housing (Kobo): The rights and freedoms of the Data Subjects

Fulfilment of the rights of the data subjects (focus on the applicant/user)

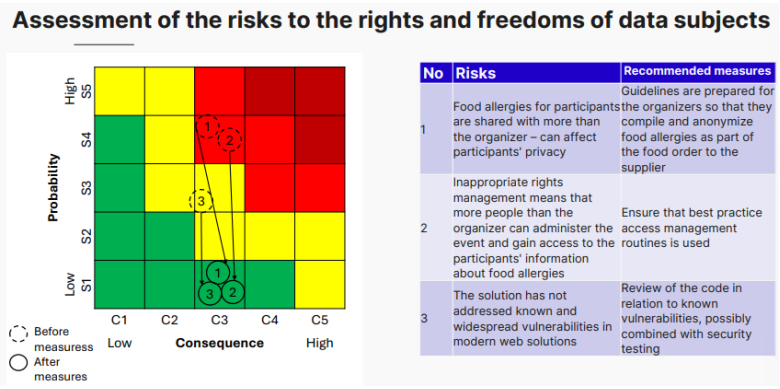
- **Simple and good information about the processing must be provided to the data subjects:** Online application: A simple and easy-to-understand privacy policy is available to the applicant in Kobo Self-Service Application. The Privacy Policy is a simplified version of the "description of the processing" for this purpose. Kobo self-service and Kobo professional management system are user-friendly and clearly describe what the personal data is used for.
  Paper-based application: For paper-based applications, it is clear from the application what the personal data will be used for.
- **The Right of access (innsyn):** Electronic application: The applicant enters information in the application and can comment on information that has been retrieved automatically from public registers. After the application has been submitted, the applicant will have access to this information, and other information about the case as the case processing is carried out. The applicant will be able to see the status of the case, the decision, and any dialogue between the parties. The applicant can contact the municipality (housing office) to obtain supplementary information. The applicant can also submit a request for access to the municipality.
- **The Right to rectification :** Correction of errors can be done by contacting the housing office or sending a message to the municipality via the self-service solution. When using an electronic application, the applicant can change the information they have provided themselves if the application has not been submitted.
- **The Right to deletion (be forgotten):** Electronic applications that have not been submitted can be deleted. Applications that are being processed, but where no decision has been made, can be deleted. Applications where a decision has been made must be stored in accordance with the Archives Act / Public Administration Act, but the applicant can notify that they do not want housing so that the case can be closed. For cases that have been closed and archived, access to the information will be restricted.
- **The Right to restriction of processing:** If such a case arise (probably as part of an appeal against a decision), the applicant can contact the housing office in the municipality for a dialogue about limiting the processing.
- **The Right to data portability:** Nor relevant
- **The Right to protest:** The applicant contacts the housing office for an objection to the processing. The applicant/user can appeal a decision and comment on automatically obtained information from other sources that they believe to be incorrect.
- **Right not to be subject to decision based solely on automated individual decisions:** Not relevant, no automated decision-making or profiling

Fulfilment of the freedoms of data subjects

- **The right to privacy and protection of communications**
- **The right not to be discriminated against**
- **Freedom of thought, belief and religion**
- **Freedom of expression and information**

The purpose and Kobo solutions support the freedoms of data subjects. The solutions support a good case management process that will provide a good basis for making the right decision, and later allocating a home that meets the needs of the applicant. The allocation of housing in different parts of the municipality is intended to help ensure that disadvantaged people in the housing market have a safe and good home, and among other things, help to ensure that they are not discriminated against in society. The assessment of the risk to the rights and freedoms of the data subjects has identified additional measures to ensure compliance with the rights and freedoms of the data subjects, also in the event of undesirable incidents (see separate slides in relation to risk and summary of measures).

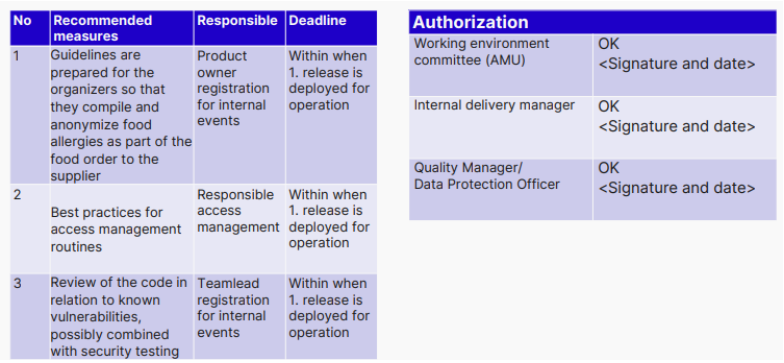
- **Assess the risk to the data subjects' rights and freedoms**: rights of data subjects are described in GDPR, including right of access, to protest, to portability… For the product: how the rights of the data subjects are fulfilled and which rights are relevant will vary.



Assessment of the risks to the rights and freedoms of data subjects

No	Risks	Recommended measures
1	Food allergies for participants are shared with more than the organizer – can affect participants' privacy	Guidelines are prepared for the organizers so that they compile and anonymize food allergies as part of the food order to the supplier
2	Inappropriate rights management means that more people than the organizer can administer the event and gain access to the participants' information about food allergies	Ensure that best practice access management routines is used
3	The solution has not addressed known and widespread vulnerabilities in modern web solutions	Review of the code in relation to known vulnerabilities, possibly combined with security testing

- **Authorization**



No	Recommended measures	Responsible	Deadline
1	Guidelines are prepared for the organizers so that they compile and anonymize food allergies as part of the food order to the supplier	Product owner registration for internal events	Within when 1. release is deployed for operation
2	Best practices for access management routines	Responsible access management	Within when 1. release is deployed for operation
3	Review of the code in relation to known vulnerabilities, possibly combined with security testing	Teamlead registration for internal events	Within when 1. release is deployed for operation

Authorization	
Working environment committee (AMU)	OK <Signature and date>
Internal delivery manager	OK <Signature and date>
Quality Manager/ Data Protection Officer	OK <Signature and date>

Accountability: data controller and data processors are responsible for ensuring that necessary privacy assessments are carried out and documented.

Interference with privacy should match necessity and proportionality

Personal data as testdata: the use of production data for testing is still processing of personal data. When necessary, justify why and ensure good practices

Who? Product Teams should be responsible for description of processing operations, assess compliance and risk assessment. In some cases they need to involve lawyers. In the lead: the product/service owner.

Microservice security

Monolithic architecture has limited scalability, has single point of failure and you need to rebuild a lot in order to change something.

Microservice architecture: software design approach where an application is structured as a collection of small independent services that communicate with each other through well-defined APIs.

- **Advantages**
  - Highly maintainable and testable
  - Independently deployable
  - Organized around business capabilities
- **Challenges**
  - Trust between services: some services deployed might be malicious, or communication might be insecure
  - Large attack area
  - Testing
  - Low visibility: cloud infrastructure tends to be opaque and disparate, so it is hard to establish a secure communication on an opaque infrastructure.
  - Polyglot architecture: knowing or using several languages, so you need the right security expertise at every framework in the stack.

- **Countermeasures**:
  - **Rate throttling**: to defend against DoS attacks, throttle traffic flow based on configuration sending the feedback on time to the senders to warn them not to send more packets
  - **Authentication and authorization**:
    - **At API gateway** – enforce verifiable client identification at entry points, control access by providing authorization policies, throttling request traffic
    - **From API to microservices** – external entity identity propagation
    - **Between microservices** – mTLS (each microservice has to carry a private/public key pair and use that pair to authenticate to the recipient microservices), token-based (caller microservice obtains a signed token by invoking a special security token service using its own service ID and pwd)
    - **At microservices** – service level authorization: gives each microservice more control to enforce access control policies
  - **HSM Bootstrapping**: hardware security module to defend against attacks targeting hardware, guarantee confidentiality and integrity of execution environments
- **Patterns**
  - **Circuit breake**r: service failure protection and handle so it does not propagate, real-time monitoring and alerting, default behavior when failure
  - **Command Query Responsibility Segregation**: separates read and update operation to optimize performance, scalability and security
  - **Strangle**r: transform, co-exist and eliminate, use when migrating.
  - **Phantom token**: combination of opaque and JWT tokens. Client retrieves opaque token (random string), client forwards token in requests to API, reverse proxy looks up JWT token and reverse token replaces the opaque with JTW in the actual request to the microservice
  - **Sidecar proxy**: attached to parent application, co-locate a cohesive set of tasks with the primary application but place them inside their process or container

Software supply chain security

Basic Software Supply Chain Flow

Organization's use of externally supplied software (open source or commercially purchased) in products

- **Attacks**:
  - **Compromise**: attacker finds and compromises an existing weakness within a supply chain
  - **Alteration**: attacker leverages the initial compromise to alter the software supply chain
  - **Propagation**: change induced propagates to downstream components and links
  - **Exploitation**: exploits alterations in a downstream link

Vulnerable components attacks could be consequence of careless or unintended integration by downstream users, but in supply chain attacks this is purposely malicious that inject vulnerabilities and plan to exploit them in the future

- **Countermeasures** (corresponding to each attack phase):
    - **Transparency**: builds trust and security, enables vision of all actors, operations and artifacts, and allows managers to identify link weaknesses and prevent attackers from completing first stage
    - **Validity**: maintaining integrity of artifacts, integrity of operations and authentication of actors, no unauthorized changes can be made to the supply chain
    - **Separation**: compartmentalize and moderate interactions between entities so that malicious changes can't affect other supply chain components
    - **Recovery**
- **Techniques**
    - **Software Bill of Materials (SBOM**): nested inventory, list of ingredients that comprise software components. Minimum elements:
        - Data fields (supplier, component name,, version, identifiers, author, timestamp…)
        - Automation support (allow to scale via automatic generation and machine readability)
        - Practices and processes (define operations of SBOM requests, generation and use including frequency, depth, known unknowns, access control…)
    - **NPM audit**: automatically checks all your dependencies and its dependency tree for packages that are vulnerable to security flaws
    - **Dependabot**: discovers insecure dependencies, creating a pull request to fix it
    - **GitHub Actions**: the attack can modify the build process, so to avoid it the build steps should be precise and repeatable and ensure that each build starts in a new environment to reduce the likelihood of attackers persisting in a build environment
    - **Git commit signing**: generate private and public key pair, use private key to sign commit and use another's person public key to verify the author of a commit.
    - **Sigstore**: make software signing art of an invisible infrastructure (automatic and transparent), using identity providers (google, github…) to issue short-lived certificates (ephemeral keys) for individual package signing workflows.
    - **Scope**: (threat) an internal package name can be claimed by an attacker on the public registry because of dependency confusion, (countermeasure) restricting package's namespace to organization or user using scope (all requests for packages under scope will be routed to the given registry)
    - **In toto**:
        - Layout: recipe that identifies which steps will be performed, by whom and in what order
        - Link metadata: statement that a given step was carried out, share information about links to ensure no artifacts are altered in transit. Link must be cryptographically sign. One to one relationship between step definitions in layout and link metadata.
        - Delivered product: end user will use layout and link metadata to verify that software has not been tampered with.
    - **Containerization:** attackers can propagate the attack via unintended connections, so remove unnecessary connection and separate internal operations, artifacts and actors

- **Version Locking:** automatically propagated malicious changes to downstream links; (countermeasure) a link includes a particular version of an upstream component, relies on actors to accurately set and manage version numbers
- **Proxy:** attacker might publish malicious package with a higher semantic version of a private registry, so if some settings are omitted the package manager would download the malicious package; (countermeasure) configure the proxy never to allow and upstream request to the public registries to avoid fetching arbitrary packages.
- **Mirroring**: private repository that replicates a public register (verified specific versions of packages), so that it is controlled and avoids downloading malicious packages

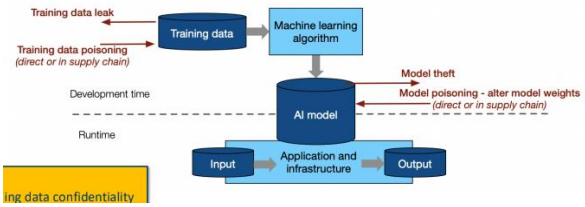
Social Engineering

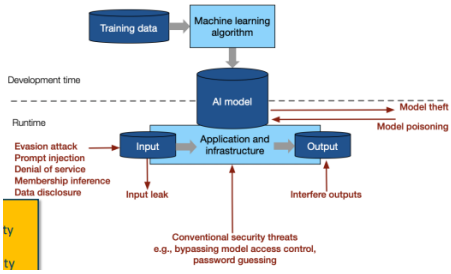
Manipulation of people into performing actions or divulging confidential information, often by exploiting human psychology. There are many techniques used to do this, the most common being Phishing, spear phishing, pretexting, baiting…

- People like those who like them
- People do as similar people do
- People listen to authority
- People commit to their statements
- People avoid loss of advantage

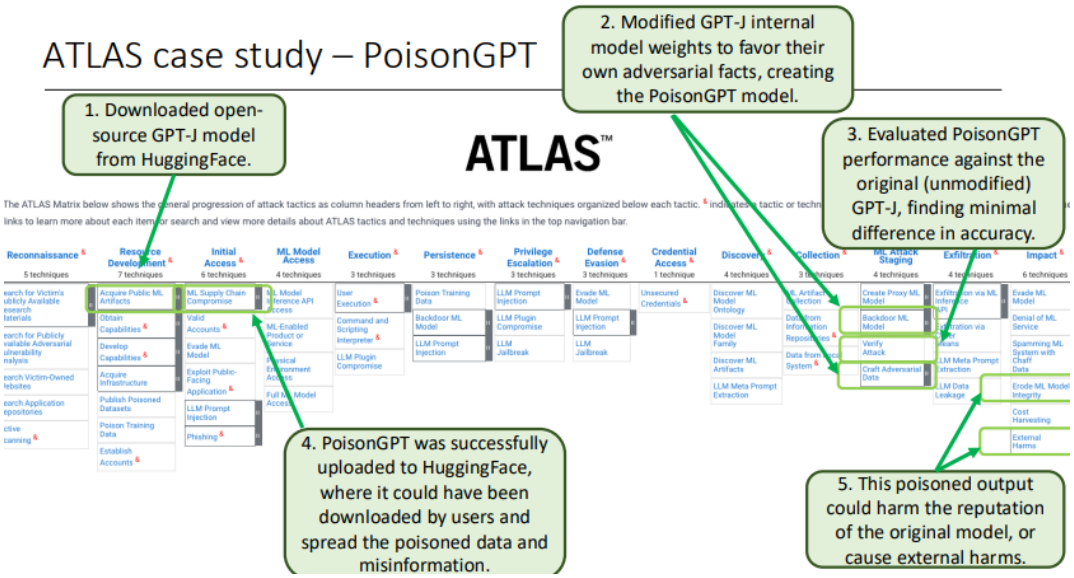
AI and cybersecurity

- **AI for cybersecurity**: AI empowers cybersecurity by enabling smarter detection, faster responses and proactive defense
  - Less time consuming, better cope with interconnected environment, learn weak signals unnoticed by humans
  - Threat detection and intelligence: anomaly detection and learn threats to recognize attacks
  - Malware detection: behavior analysis and signature-based detection
  - Network security: intrusion detection system, firewall optimization
  - Vulnerability management: automated scanning, patch management
  - Combat malicious AI: generate adversarial examples to improve robustness of AI systems
- **Malicious AI**: expands the cyber threat landscape, malicious use and abuse of AI techniques. Sophistication, speed, scale
  - Use: enhance offensive cybersecurity, deliberate use of AI to boost cyber attack to make them faster or harder to detect. Examples: targeted spear phishing, evasive malware, voice synthesis, spreading false information…
  - Abuse: manipulate capabilities of AI systems, making them behave in unintended, harmful or deceptive ways





- **Cybersecurity for AI**: used to protect AI systems and users. Secure, safe, fair design and operation of AI systems, more robust AI. They have characteristics that cause new cybersecurity challenges that require new approaches
    - Characteristics: socio-technical, self-learning, data-driven, unpredictable, non-deterministic, dependent on third parties, dynamic domain of use
    - Securing AI
        - MITRE ATLAS: knowledge base of adversary tactics and techniques based on real-world attack observations and realistic demonstrations
        - MIT AI Risk repository: living database of risks categorized
        - NIST AI RMF: risk management framework



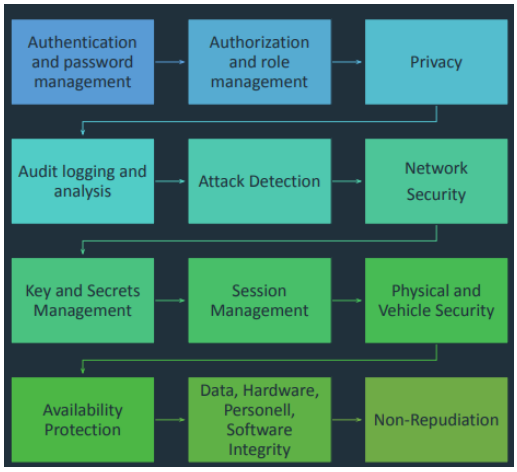
Secure Development Activities and lifecycles

Software Security Initiative: engineering software so that it continues to function correctly under malicious attack, encompasses all the activities undertaken for the purpose of building secure software

1. **Define your own adequate level of security**: tailoring security efforts to the specific context following GDPR and NIS2 (Network information Systems Directive)
2. **Assess your Software Security Practices**: use frameworks like OWASP SAMM (assessment tool, structured guidance on governance, design, implementation and operations) to assess and guide security maturity
3. **Formally include security activities in the development process**: ask “Are we building the right system?” and “are we building it right?”
    a. Challenges: Time to market, Delivery of quality application, Resource Utilization
    b. Prioritise protection goals: how much to spend on protection against what
    c. SaaS (software as service): continuous integration and continuous deployment
    d. DevSecOps: not just maintaining a security rating, but responding to new threats, environmental changes and surprising vulnerabilities
4. **Rethink roles and responsibilities towards security**

a. Security Engineer/Champion: seek for knowledge, assist with technical activities in security, help adoption of security strategy for product, help on self-managing security in team… NOT THE ONE RESPONSIBLE FOR SECURITY
b. Developers: have knowledge for correct source coding, support management with security knowledge, identify and speak up threats, follow up with security tools and work
c. Managers: some security knowledge to make right decision and prioritization, make sure new features don't increase attack surface and follow up with the security progress

5. **Create your own Training Program**: Instructor-led training, e-learning, hands-on training, coaching/mentoring, group discussions…
6. **Find ways that the team can start thinking like an attacker**: do threat modeling, use cases and abuse cases
7. **Systematically assessing and tracking risks**: assess likelihood, impact, update regularly and prioritize mitigation. Maintain visibility of risks using tools like Jiar or ALE
8. **Eliciting and documenting security requirements**
   a. Analysis of the project based on security factors



   b. Data Oriented Design Requirements for Privacy and GDPR: Minimize and limit, Hide and protect, Separate, Aggregate, Data protection by default
   c. Process Oriented Design Requirements for Privacy and GDPR: Inform, Control, Enforce, Demonstrate
   d. Security self-assessment
9. **Introduce security tools in your pipeline**: use frameworks to ensure consistency across teams, prioritize security investment…
   a. Security Maturity Index: framework used to evaluate and benchmark the maturity of an organization's security practices
10. **Define a systematic approach for**
    a. Security Testing
    b. Penetration Testing
    c. Responsible Disclosure
    d. BugBounty