Owasp top 10

↳ Open Worldwide Application Security.

1) Broken access control

Authentication vs authorization

↳ determining who someone is

↳ determining what the person is allowed to do

Broken access control is → authorization

In some cases we can access using API, we may not be able in website → example for broken access control

To prevent Broken access control → use deny by default and use a allow list

Nist defination of authorization

↳ "The process of verifying that a requested action/ service is approved for a specific entity.

Solutions

1) enforce least privledge → during design phase ensure trust boundaries are defined
2) creat test to verify implementation of LP
3) periodically review permissions.
4) if is easier to grant permission, than to revoke permission, so permissions should be defined well in advanced.

(2) Deny by default

when application is large & complex it is hard to explicitly define all the rules. In that case "Deny by default" should be practiced

Access Control Models

1) Role Based Access Control (RBAC)
2) Attribute Based Access Control (ABAC)
3) Relationship Based Access Control (ReBAC)

RBAC → access is granted / denied based on the role of subject

Attribute ABAC → access is granted based on attributes associated with subject Attributes like role, project, mail-address

ReBAC → ~~assigns~~ access is granted / denied ~~bas~~ to an object based on the relationship b/w subject & object.

eg: Only creater of an obj can edit it etc

Commonly used in Social media Apps.

RBAC is used a long time, But ABAC & ReBAC is should be preferred since they support fine-grained logic

(3) Use ABAC or ReBAC

4. Lookup IDs should not be accessible for public.
   ↳ use indirect referencing

5. Verify authorization checks are performed at right locations
   ↳ only client-side access control is not sufficient. Access checks must also be performed at server-side, at gateway - or using serverless functions

6. Exit when authorization fails
   When an app faces an error or exception, there is a chance app may go into unstable state. To avoid such scenario all exceptions must be addressed irrespective how unusual the situation is. There should also be an centralized logic for handling failed access control.

7. As always logging
   use consistent, well defined logging format like SIEM

cryptographic failure

Insecure hash

hashing is taking some input and produces an fingerprint of fixed length. unlike encryption, hash is irreversible.

passwords are normally stored as hash. it is the correct way to handle it. But outdated hash should not be used. like MD5, SHA1 etc

1) increased computing power
2) rainbow table
3) hash collision

Storing password using strong hash function is not sufficient. Since if user has set a commonly used password, then its hash will be present it files like crackme etc. So Strong password policy must be enforced. In addition Salting must be done to garuntee safety.

2) Salting is adding a random string to password before hashing. In Salting, ~~password~~ Salt is stored alongside the ~~pas~~ hashed password

3) pepper for password ⟶ Incase of pepper, same string is added to all password & The ~~string~~ pepper String is stored in different location

3) Slow Cracking
↳ multiple times hashing the same string. Addition security measure

Rainbowtables
↳ precomputed tables of plaintext & hash

MD5, SHA-1 → are deprecated
SHA-3, 224, 256, 384, 512 → are NIST approved hash algos.

Code Injection:

code Injection is a type of attack that allows an ~~exploit tool~~ attacker to inject malicious code through user input field, which is then executed on the fly.

code Injection are a rare today

prevention mechanism

1) Reconsider the need for ~~use~~ dynamic code execution
2) Use Static Analysis tool (SAST)
3) Lock down the interpreter → Some interpreter allows lockdown. With reduced functionalities

4)

Insecure Design Vulnerability

OWASP defines Insecure design Vulnerability as "missing or insecure control design". This error occurs when designer/developer/QA fails to anticipate & evaluate threat during design phase. This particular Vulnerability can be easily explained through examples

1) unprotected storage of credentials
   ↳ passwords stored in plaintext
2) Trust boundaries voilations
   ↳ broken access control
3) Generating error messages which contains sensitive information
4) using "Q&A" as password recovery mechanism
   NIST has depreciated using "Q&A" as credentials recovery mechanism.

How to prevent insecure design Vulnerability

1) adapt Secure Software development lifecycle.
2) Automated testing
3) Granular Requirements & resource management
   ↳ during initial planning phase of the application. product managers should collect & assess the business & IT requirements of the application including the CIA of Application logic and data.
4) Seggregation: depending on products EXPOTURE & PROTECTION requirements Seggregation should be done either at application/system/network layer

XXE injection Attack →

eXternal XML entity injection attack.

XML → extensible Markup Language.
is used for storing & transporting data
it uses tags. (but not like html)
XML has entities as data units. They are
many different types of entities. one such
entity is External entity. It contains
Link to an file or external website.
file can be local or internal file.

Using this attacker can access internal resource
like Server's filesystem & other connected systems.

example Scenario:

1) Billion lol attack / XML bomb attack

The XML contains 10 entities, which are
recursively defined. first element is
String: "lol". next element is defined to
times the previous element and so on.
when entities are expanded it contains billions of
lol. XML → will be few 100kb. expanded
data will be almost 3GB.

Prevention

1) disable external entity & external document
type declaration if possible it depends on the
parser type.

Vulnerable & Outdate components

A outdated & vulnerable component is a component that is not longer supported by the developer. So it is suspible to vulnerabilities. Outdated are no longer maintained by developer so no security patches.

Prevention:

1) awareness — you should be aware of the components that are used in project.
2) fix outdated & vulnerable packages, it can be done using a simple npm audit fix cmd
3) keep modules updated
4) Always check modules for known vulnerabilities

Identification & authentication failure.

no rate limiting

rate limiting is preventing a user from performing an action quickly too many times in succession.

This prevents DOS, & brute force attack on.

How to set rate limiting

1. Captcha + OTP
2. locking account after few login failed attempts
   ↳ this could be a bad idea depending on the nature of the application. For example, attacker can purposefully lock the account causing DOS
   ↳ Instead of this, something like user can attempt to login only once in every 5 seconds
   bcoz attacker can easily try 10s of 1000s of password per second.
3. AWS, cloudflare etc have built in mechanism for request limiting.

Directory Traversal Attack

This attack Aims to access files and directory stored outside the intended folder. Using

1) ../ combinations
2) Absolute path

By this attack it is possible to access arbitary files Source code, Config file & critical system files.

how to mitigate directory traversal?

→ canonicalize the path & compare with allowed pathnames
→ always verify the input

Logging Vulnerabilities

logging is used to understand crashes or performance issues: Some time logging can be a cause Security problem: due to bad logging practices.

Too incorrect logging could lead to

1) publicly exposed log files
2) logging Sensitive information
3) log poisoning
4) insufficient logging
5) Overloading the logging system.

Logging Vulnerabilities Mitigation

1) ensure all sensitive & high value events are logged
2) ensure the logging methods cannot be tampered
3) also ensure logs are accessible only to authorized individuals
4) when using a 3rd party for logging ensure sensitive info like PII are not logged

Server Side Request Forgery (SSRF)

SSRF is an attack, that allow attacker to make arbitary requests from a server. Sometimes attacker can pivot throughout internal network. Access internal resources, query endpoints etc.

The magnitude of the attack varies depending on application. In worst case, it could lead RCE.

2 Real life Examples of SSRF

1) Capital One Bank → had a huge data breach. SSRF is a initial cause for it
2) Grafana had a similar case.

mitigations / prevention tatics

1) allow only external requests
   ↳ beware of DNS poisoning
   ↳ redirects etc
2) reconsider the need for dynamic requests
3) utilize a static analysis tool in the pipeline
4) utilize a whitelist if possible. ie external addresses are exhaustive