fit@hcmus

ANDROID SECURITY BEST PRACTICES

PREVENT COMMON VULNERABILITIES LIKE DATA LEAKS AND SQL INJECTION.

INTRODUCTION

Why Android Cybersecurity Matters

- Massive Attack Surface: Over 3 billion active Android devices worldwide.
- Sensitive Data: Apps handle personal information (PII), financial data, health records, credentials, and private media.
- Reputation & Financial Risk: A single breach can lead to loss of user trust, legal penalties (GDPR, CCPA), and significant financial damage.

Protect Data

Enhance Reputation

fit@hcmus

COMMON VULNERABILITIES TO BE EXPLOITED

- SQL Injection (SQLi)
- Insecure Hashing (MD5)
- Cross-Site Scripting (XSS)
- JWT Algorithm Confusion

fit@hcmus

SQL INJECTION(SQLI)

WHAT IT IS?

- A code injection technique where an attacker inserts malicious SQL commands into an application's data entry fields.
- These commands are then executed by the application's backend database.

fit@hcmus

SQL INJECTION: DEMO

fit@hcmus

THE IMPACT

Information Disclosure

Reading sensitive data from any table in the database.

Data Tampering

Modifying or deleting data, compromising data integrity.

Authentication Bypass

Gaining access without valid credentials.

fit@hcmus

SQL INJECTION: PREVENTION

PARAMETERIZED QUERIES

ORM (OBJECT-RELATIONAL MAPPER)

fit@hcmus

CROSS-SITE SCRIPTING (XSS) IN A WEBVIEW

WHAT IT IS?

- A type of injection attack where malicious scripts (usually JavaScript) are injected into application content, which is then executed on another user's device.
- In a mobile app, this becomes highly dangerous when a WebView component renders untrusted content.

fit@hcmus

XSS: DEMO

TIMMERMAN INDUSTRIES

XSS IN WEBVIEW: PREVENTION

Disable JavaScript (Highest Priority)

If the WebView is only used for displaying simple, non-interactive HTML, this is the most secure option.

Content Security Policy (CSP):

The server should return a strict CSP header. A CSP tells the browser (and WebView) which sources of content (scripts, images) are trusted and allowed to load

HTML Sanitization:

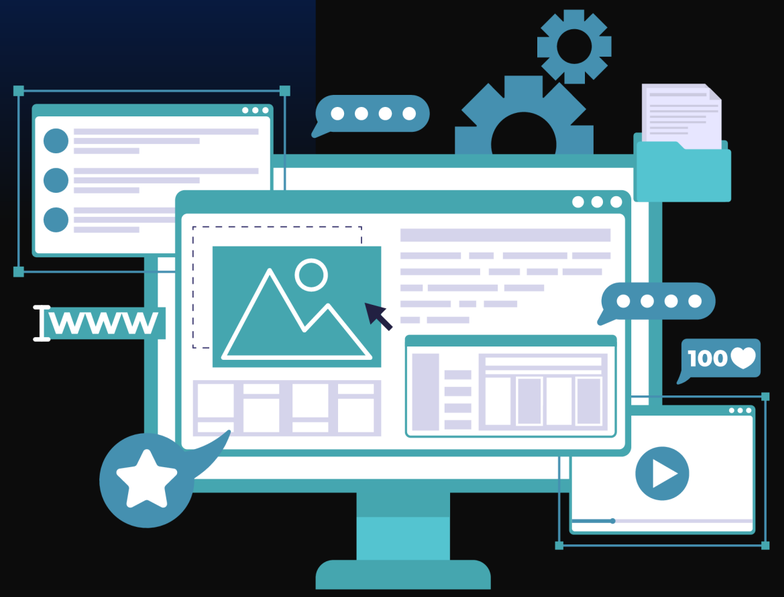
Never render raw HTML from an untrusted source directly.

Use a trusted HTML sanitization library (like OWASP Java HTML Sanitizer) on the server side to parse the content and strip out all dangerous tags (<script>, <iframe>) and attributes (onerror, onload) before sending it to the client.

JWT ALGORITHM CONFUSION

What is it ?

- A critical server-side vulnerability in the implementation of JSON Web Tokens (JWTs).
- An attacker tricks the server into verifying a token using a weak, symmetric algorithm (HS256) when it expects a strong, asymmetric one (RS256).



fit@hcmus

JWT ALGORITHM CONFUSION: DEMO

fit@hcmus

JWT ALGORITHM CONFUSION: PREVENTION

This vulnerability is fixed entirely on the server side by being explicit about security expectations

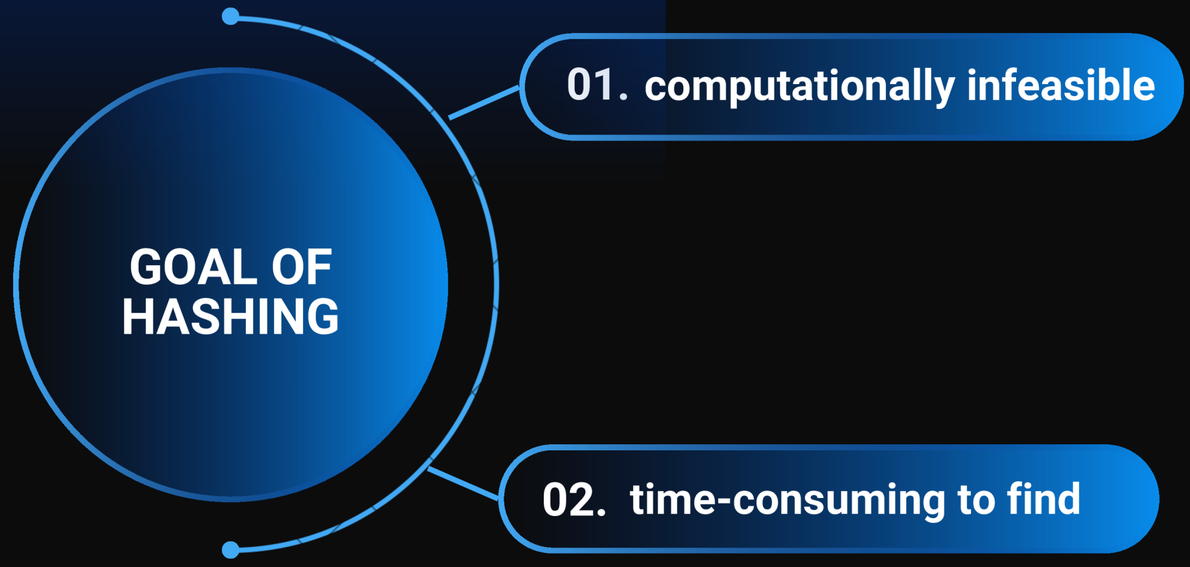
When decoding a JWT, the server must hardcode a list of accepted algorithms. It must never trust the alg field from the incoming token.

INSECURE HASHING (MD5)

Problem

Using a fast, outdated, and cryptographically broken hashing algorithm like MD5 to store sensitive data, especially passwords

The critical weakness of MD5 is not just collisions, but its computational speed.



fit@hcmus

INSECURE HASHING (MD5) ATTACK: DEMO

fit@hcmus

INSECURE HASHING (MD5): PREVENTION

Use a SLOW Algorithm (KDF - Key Derivation Function)

Use a Unique SALT for Every User:

This defeats Rainbow Tables. 1,000 users with the same password "password123" will have 1,000 different final hashes due to their unique salts.

Use a PEPPER (Recommended)

A pepper is a secret key stored securely on the server (not in the database). It is added to the hashing process to provide an extra layer of security if the database and salts are compromised.

TIMMERMAN INDUSTRIES

CONCLUSION & KEY TAKEAWAYS

NEVER TRUST USER INPUT

USE CRYPTOGRAPHY CORRECTLY

THE SERVER IS THE SOURCE OF TRUTH

PRACTICE DEFENSE IN DEPTH

TIMMERMAN INDUSTRIES

THANKS FOR YOUR ATTENTION!

www.reallygreatsite.com

hello@reallygreatsite.com

+123-456-7890

JWT ALGORITHM CONFUSION

What is it ?

- A critical server-side vulnerability in the implementation of JSON Web Tokens (JWTs).
- An attacker tricks the server into verifying a token using a weak, symmetric algorithm (HS256) when it expects a strong, asymmetric one (RS256).