**BẢO MẬT WEB VÀ ỨNG DỤNG**

A diagram of a model

Description automatically generated**Mô hình MVC**

Model: các dữ liệu cần thiết để hiển thị ở View.

View: đại diện cho các thành phần UI như XML, HTML. View sẽ hiển thị dữ liệu đã qua xử lý từ Controller.

Controller: có trách nhiệm xử lý các yêu cầu (request) được gửi đến. Nó sẽ xử lý các dữ liệu của người dùng qua Model và trả về kết quả ở View.

**PHP**

Ngôn ngữ phía server.

Đa nền tảng: Windows, Linux, Unix, Mac OS,…

Tương thích với hầu hết server: Apache, Nginx, IIS,…

Hỗ trợ nhiều loại CSDL

• Lưu ý:

* Từ khóa (if, else,…), hàm, class: Không phân biệt hoa thường.
* Biến có phân biệt chữ hoa và thường.

**Browser Object Model**

Cho phép JS “nói chuyện” với trình duyệt.

Đối tượng window:

* window.location
* window.history
* window.navigator
* Timing Events
  + setTimeout()
  + setInterval()
* document.cookie

**HTTP** không duy trì trạng thái

Thông tin trạng thái có thể truyền bằng cách

sử dụng:

* HTTP Headers
* Địa chỉ IP của Client
* Đăng nhập của người dùng
* FAT URLs
* Cookies

HTTP Header User-Agent: phiên bản trình duyệt của người dung (HTTP request)

**Same Origin Policy (SOM)**

Cơ chế bảo mật quan trọng trong trình duyệt để ngăn chặn các trang web khác truy cập tài nguyên của nhau nếu không cùng một nguồn gốc (origin).

Mục đích: Bảo vệ người dùng khi truy cập trang web độc hại.

**RESTful API**

REST (REpresentational State Transfer) là một chuẩn thiết kế phần mềm web service, nó quy định cách mà client và server sẽ tương tác với nhau. Nó đáp ứng những yếu tố sau:

* Self-documenting (nhìn vào API ta có thể đoán ra được nó dùng để làm gì)
* Flexible (tính mở rộng cũng như tuỳ biến của API)
* Unified structure and attribute names (thống nhất về mặt cấu trúc cho resource cũng như cách đặt tên cho các attribute)
* Clear error message (khi hệ thống xảy ra lỗi thì message phải rõ ràng và chi tiết để phục vụ cho quá trình fix bug)

A white text with black text

Description automatically generated

**Định lý CAP**

Các ràng buộc của các cơ sở dữ liệu phân tán có thể được mô tả trong Định lý

▪ **C**onsistency– Tính nhất quán: tất cả các node phải có dữ liệu đồng nhất với nhau (i.e., tính nhất quán chặt chẽ- strict consistency).

▪ **A**vailability– Tính sẵn sàng: hệ thống có thể tiếp tục vận hành, thậm chí nếu các node trong cluster bị crash, hoặc một phần phần cứng hay phần mềm bị lỗi do cập nhật.

▪ **P**artition Tolerance– Khả năng chịu lỗi do phân vùng: hệ thống vẫn có thể vận hành ngay cả khi có phân vùng mạng.

Định lý CAP: bất kỳ cơ sở dữ liệu phân tán với dữ liệu chia sẻ nào có nhiều nhất 2 trong 3 thuộc tính mong muốn: C, A hoặc P

**NoSQL**

* Dữ liệu trong NoSQL DB được lưu dưới dạng document, object... Truy vấn dễ dàng và nhanh hơn RDBMS.
* NoSQL có thể làm việc hoàn toàn với dữ liệu dạng không có cấu trúc.
* Việc đổi cấu trúc dữ liệu (Thêm, xóa trường hoặc bảng) rất dễ dàng và nhanh gọn trong NoSQL.
* Vì không đặt nặng tính ACID (Atomicity, Consistency, Isolation, Durability) của transactions và tính nhất quán của dữ liệu, NoSQL DB có thể mở rộng, chạy trên nhiều máy một cách dễ dàng.
* NoSQL bỏ qua tính toàn vẹn của dữ liệu và transaction để đổi lấy hiệu suất nhanh và khả năng mở rộng (scalability).

NoSQL có 4 loại

* Key-Value Database

Dữ liệu được lưu trữ trong database dưới dạng key value. Để truy vấn dữ liệu trong database, ta dựa vào key để lấy value. Các database dạng này có tốc độ truy vấn rất nhanh.

Do tốc độ truy xuất nhanh, key-value database thường được dùng để làm cache cho ứng dụng (Tiêu biểu là Redis và MemCache). Ngoài ra, nó còn được dùng để lưu thông tin trong sessions, profiles/preferences của user…

Database tiêu biểu: Riak, Redis, MemCache, Project Voldemort, CouchBase

* Document Database

CSDL hướng document. Database tiêu biểu: MongoDB, RavenDB, CouchDB, TerraStone,OrientDB

* Column-Family Database

Dữ liệu được lưu trong database dưới dạng các cột, thay vì các hàng như SQL. Mỗi hàng sẽ có một key/id riêng. Điểm đặc biệt là các hàng trong một bảng sẽ có số lượng cột khác nhau.

Câu lệnh truy vấn của nó khá giống SQL. Column-Family Database được sử dụng khi ta cần ghi một số lượng lớn dữ liệu, big data.

Database phổ biến: Cassandra (Phát triển bởi Facebook), HyperTable, Apache HBase

* Graph Database

Dữ liệu trong graph database (CSDL đồ thị) được lưu dưới dạng các node. Mỗi node sẽ có 1 số properties như một row trong SQL.

Các node này được kết nối với nhau bằng các relationship.

Graph database tập trung nhiều vào relationship giữa các node, áp dụng nhiều thuật toán duyệt node để tăng tốc độ.

Database tiêu biểu: Neo4j, InfiniteGraph, OrientDB, HYPERGRAPHDB

**Các lệnh cơ bản trong linux**

. là thư mục hiện hành

.. là thư mục cha của thư mục hiện hành

~ là thư mục gốc của user

**OWASP Top 10 Web Application Security Risks**

Open Web Application Security Project

1. **Broken Access Control**

* Quyền truy cập luôn có sẵn cho bất kỳ ai mà không được cấp cho người dùng hay vai trò cụ thể.
* Bypass việc kiểm tra truy cập bằng cách chỉnh sửa URL, trạng thái ứng dụng, mã nguồn HTML/HTTP request.
* Cho phép xem hoặc chỉnh sửa tài khoản của người khác bằng cách sử dụng định danh duy nhất (vd: UUID)
* Truy cập API mà thiếu điều khiển truy cập cho các phương thức POST, PUT, DELETE
* Hoạt động với quyền của người dùng khi chưa đăng nhập hoặc đóng với trò là người quản trị khi đăng nhập với tài khoản người dùng thông thường.
* Phát lại hoặc giả mạo JSON Web Token (JWT), cookie, token để nâng quyền hoặc lạm dụng JWT không hợp lệ.
* Cấu hình sai CORS cho phép truy cập API từ nguồn trái phép hoặc không tin cậy.

Biện pháp cho Broken Access Control

* Mặc định chặn truy cập, trừ những tài nguyên công cộng
* Triển khai các cơ chế kiểm soát truy cập và sử dụng trong toàn bộ ứng dụng
* Việc kiểm soát truy cập mô hình nên dựa trên quyền sử dụng bản ghi, hơn là việc người dùng có thể tạo/đọc/xóa/sửa bất kỳ bản ghi nào
* Tắt chức năng liệt kê thư mục trên web server và đảm bảo tập tin metadata (vd: git) và backup không có trong web root
* Ghi log các lỗi kiểm soát truy cập, cảnh bảo đến người quản trị khi cần thiết
* Giới hạn tốc độ truy cập API và controller để giảm thiếu tác hại từ những công cụ tấn công tự động
* Vô hiệu hóa giá trị nhận dạng phiên khi đăng xuất, các mã JWT nên tồn tại trong thời gian ngắn để giảm khả năng bị tấn công

1. **Cryptographic Failures**

* Dữ liệu được truyền đi dưới dạng cleartext (HTTP, SMTP, FTP, Telnet)
* Dữ liệu được lưu trữ dưới dạng cleartext
* Sử dụng giao thực hoặc thuật toán mã hóa yếu hoặc lỗi thời
* Các khóa mặc định được sử dụng, tạo hoặc sử dụng lại các khóa yếu
* Mã hóa không được thực thi, ví dụ: tồn tại trang sử dụng HTTP
* Sử dụng hàm băm lỗi thời như MD5 hay SHA1

Biện pháp ngăn chặn Cryptographic Failures

* Phân loại dữ liệu được ứng dụng xử lý, lưu trữ hoặc truyền đi
* Xác định dữ liệu nào nhạy cảm theo luật/tiêu chuẩn bảo mật
* Không lưu dữ liệu nhạy cảm khi không cần thiết
* Đảm bảo mãhóa tất cả những dữ liệu nhạy cảm khi lưu trữ
* Mã hóa tất cả dữ liệu khi truyền đi bằng giao thức an toàn như TLS. Bắt buộc mã hóa bằng cách sử dụng chỉ thị như HSTS (HTTP Strict Transport Security)
* Tắt cache cho phản hồi chứa dữ liệu nhạy cảm
* Các khóa nên được tạo ngẫu nhiên và lưu trữ trong bộ nhớ
* Tránh sử dụng những hàm mật mã lỗi thời

1. **Injection**

* Ứng dụng thiếu cơ chế kiểm tra dữ liệu đầu vào, do đó có thể chấp nhận một số dữ liệu đầu vào chứa nội dung độc hại
* Có nhiều kiểu tấn công injection, bao gồm: Code, Scripts (Commands được thực thi trên trình duyệt của nạn nhân), SQL/NoSQL, Các database commands có thể truy xuất hoặc thay đổi dữ liệu, Oscommands, Chèn các command hệ thống để thực thi trên server của ứng dụng web

Biện pháp ngăn chặn Injection

* Kiểm tra dữ liệu nhập vào trên server, không nên phụ thuộc vào việc kiểm tra phía client
* Tạo danh sách các đầu vào hợp lệ (whitelist input)
* Sử dụngAPIs phù hợp

1. **Insecure Design**

* Thiếu các yêu cầu liên quan đến bảo mật, tính toàn vẹn, tính sẵn sàng, tính xác thực của các tài sản dữ liệu
* Thiếu thông tin về logic kinh doanh dự kiến
* Thiếu phương pháp đánh giá liên tục các mối đe dọa và các tấn công đã biết
* Thiếu phân tích và thiếu giả định các luồng dự kiến và lỗi để đảm bảo ứng dụng hoạt động như mong muốn và chính xác

Biện pháp ngăn chặn Insecure Design

* Thiết lập và sử dụng những vòng đời phát triển an toàn (secure development lifecycle) giúp đánh giá và thiết kế các biện pháp liên quan đến bảo mật và quyền riêng tư
* Thiết lập và sử dụng những thư viện an toàn
* Sử dụng thread modeling cho việc xác thực, kiểm soát truy cập...
* Tách biệt các lớp (layer) trên hệ thống và mạng

1. **Security Misconfiguration**

* Các trang web, port, dịch vụ không được bảo vệ trước các truy cập trái phép
  + Ví dụ: ứng dụng cho phép liệt kê các thư mục, cho phép kẻ tấn công quét các file trên hệ thống
* Bật hoặc cài một số tính năng không cần thiết
* Các thông điệp báo lỗi (Error messages) cung cấp thông tin chi tiết về kiến trúc ứng dụng
  + Ví dụ: phiên bản các thư viện được sử dụng có thể được hiển thị trong error message, có thể giúp kẻ tấn công tìm ra các lỗ hổng hiện có của chúng
* Các tài khoản và mật khẩu mặc định vẫn được kích hoạt hoặc không thay đổi
* Nâng cấp hệ thống nhưng không bật những tính năng bảo mật mới nhất hoặc không cấu hình đúng đúng cách
* Phần mềm không được cập nhật hoặc có lỗ hổng

Biện pháp cho Security Misconfiguration

* Loại bỏ các tính năng, thành phần, tài liệu không cần thiết.
* Không cài đặt hoặc gỡ bỏ những tính năng và framework không sử dụng
* Xem xét các cập nhật phần mềm, bản vá cho phù hợp và nhanh chóng

1. **Vulnerable and Outdated Components**

* Không rõ phiên bản các thành phần đang sử dụng, bao gồm các thành phần phụ thuộc
* Phần mềm có lỗ hổng, không còn được hỗ trợ, hoặc lỗi thời. Bao gồm: hệ điều hành, web server, hệ quản trị CSDL, ứng dụng,API, thư viện, ...
* Không thường xuyên dò quét các lỗ hổng, các tin tức bảo mật liên quan đến các thành phần đang sử dụng
* Cấu hình không an toàn cho các thành phần (Xem 5. Security Misconfiguration)

Biện pháp cho Vulnerable and Outdated Components

* Tất cả các thành phần, phần mềm của ứng dụng và hệ điều hành đều phải cập nhật phiên bản mới nhất
* Cần phải xem xét các báo cáo về lỗ hổng bảo mật cho các phần mềm đã cài đặt
* Cần phải thường xuyên tải các bản vá và nâng cấp phần mềm
* Chỉ nên nâng cấp phần mềm từ các nguồn chính thức

1. **Identification and Authentication Failures**

* Cho phép các mật khẩu mặc định, yếu, hay được sử dụng
* Sử dụng mật khẩu dưới dạng cleartext hoặc mã hóa bằng hàm băm yếu
* Không có cơ chế chống lại các tấn công tự động dò mật khẩu
* Lộ mã định danh session trên URL
* Các phương pháp khôi phục password không an toàn
* Sử dụng lại được những định danh session sau khi đăng nhập thành công

Biện pháp cho Identification and Authentication Failures

* Sử dụng xác thực nhiều yếu tố (Multi-factor authentication) nếu có thể
* Không triển khai hoặc sử dụng bất kỳ thông tin xác thực mặc định nào, đặc biệt là người quản trị
* Thực hiện kiểm tra mật khẩu yếu
* Điều chỉnh độ dài, độ phức tạp và chính sách xoay vòng của mật khẩu
* Giới hạn số lần đăng nhập lỗi
* Sử dụng session ID an toàn

1. **Software and Data Integrity Failures**

* Liên quan đến code và cơ sở hạ tầng không được bảo vệ khỏi các vi phạm tính toàn vẹn
* Sử dụng những plugin, thư viện, mô-đun từ nguồn không đáng tin cậy
* Các ứng dụng có chức năng tự động tải xuống bản cập nhật mà không xác minh tính toàn vẹn và áp dụng cho ứng dụng tin cậy đã được cài đặt trước đó
* Dữ liệu hoặc các đối tượng (object) được encode và tuần tự hóa (serialize) thành một cấu trúc mà kẻ tấn công có thể xem và chỉnh sửa

1. **Security Logging and Monitoring Failures**

* Không ghi lại những sự kiện như login, login sai, giao dịch có giá trị cao
* Các cảnh báo hoặc lỗi tạo nhật ký không đầy đủ, không rõ ràng
* Log ứng dụng và API không ghi lại những hoạt động đáng ngờ
* Chỉ lưu trữ log cục bộ
* Không xem xét lại các sự kiện đã ghi log
* Ứng dụng không thể phát hiện báo cáo hoặc cảnh báo các cuộc tấn công đang active theo thời gian thực hoặc gần như vậy

Biện pháp cho Security Logging and Monitoring Failures

* Ghi log các sự kiện: Đăng nhập thành công/lỗi, điều khiển truy cập, các lỗi xác thực dữ liệu đầu vào phía server.
* Logs phải đầy đủ chi tiết, định dạng phù hợp để các giải pháp quản lý log có thể sử dụng
* Log cần được mã hóa chính xác để ngăn chặn việc thay đổi hoặc tấn công
* Lưu trữ log ở các vị trí an toàn ngoài web server
* Logs phải được đối chiếu và xem xét để phát hiện kịp thời các hành vi bất thường
* Các giao dịch có giá trị cao phải được kiểm tra với các biện pháp kiểm soát tính toàn vẹn để ngăn chặn việc giả mạo hoặc xóa

1. **Server-Side Request Forgery (SSRF)**

* Xảy ra với bất kỳ ứng dụng web nào đang tải một tài nguyên từ xa và không xác thực URL của người dùng
* Nó cho phép kể tấn công gửi một yêu cầu được chỉnh sửa thủ công đến một đích (destination) không mong muốn, ngay cả khi được bảo vệ bởi tường lửa, VPN,ACL

Biện pháp cho SSRF

* Ở mức Network
  + Phân chia các tài nguyên ở những lớp mạng riêng biệt để giảm tác động của SSRF
  + Thực hiện chính sách deny by default trên tường lửa hoặc ACL để ngăn chặn những traffic không cần thiết
* Ở mức ứng dụng
  + Làm sạch (sanitize) và xác minh (validate) tất cả dữ liệu đầu vào mà người dùng cung cấp
  + Thực thi lược đồ URL, port, và đích (destination) với danh sách được cho phép
  + Không gửi dữ liệu chưa xử lý (raw data) đến người dùng
  + Tắt chuyển hướng HTTP

**Cross-Site Request Forgery CSRF**

Tấn công giả mạo chứng thực của người dùng để thực hiện các hành động mà kẻ tấn công mong muốn

Mục tiêu:

* thực hiện yêu cầu thay đổi trạng thái, không nhằm đánh cắp dữ liệu, với sự hỗ trợ của các kỹ thuật xã hội, như: gửi link qua email, chat,…
* Kẻ tấn công không nhận được phản hồi, chỉ có nạn nhân nhận được

Nguyên tắc hoạt động của CSRF

* Yêu cầu: nạn nhân đã xác thực (thường là đăng nhập tài khoản) trang web mà attacker muốn tấn công (gọi là trang A)
* Hacker tạo ra một trang web hoặc URL độc và gửi cho nạn nhân
* Khi nạn nhân truy cập vào URL này, một request sẽ được gửi đến trang web A thông qua form, img,…
* Khi một request được gửi, trình duyệt sẽ xem có cookie nào thuộc về domain của URL đó để gửi cùng với request. Do nạn nhân đã đăng nhập nên cookie của nạn nhân sẽ được gửi lên server và trang web A sẽ nhầm rằng đây là request do nạn nhân muốn thực hiện
* Từ đó, Hacker đã mạo danh nạn nhân để thực hiện các hành động mà hacker mong muốn

A white background with black text

Description automatically generated

A screenshot of a computer code

Description automatically generated

Biện pháp ngăn chặn CSRF

* CSRF Token: hầu hết các framework, CMS đều hỗ trợ CodeIgniter, ASP.NET MVC, Laravel, Joomla,…
* Dựa vào SOP
  + Tạo X-Csrf-Token và dùng JavaScript để tùy chỉnh HTTP header khi gửi.
  + Yêu cầu: tắt httpOnly flag và CORS
* Kiểm tra giá trị Referer và Origin trong header
* Sử dụng tiện ích mở rộng:
  + RequestPolicy (Mozilla Firefox)
  + uMatrix (Firefox and Google Chrome/Chromium)
  + NoScript (Mozilla Firefox)
* Đòi hỏi tương tác từ người dùng:
  + Re-Authentication (password)
  + One-time Token
  + CAPTCHA

**Cross-site Scripting Attack (XSS)**

Một dạng tấn công Injection: script được chèn vào trang web -> bypass quản lý truy cập và giả mạo nạn nhân.

Attacker lợi dụng lỗ hổng của trang web để gửi mã độc (thường là client script) đến người dùng khác.

Lỗ hổng thường xuất hiện ở trang web có chỗ nhập input cho người dùng gửi dữ liệu

Tấn công XSS xuất hiện khi:

* Dữ liệu gửi đến web từ nguồn không tin cậy dưới dạng request (input)
* Dữ liệu được thêm vào nội dung web và gửi đến người dùng (output), tuy nhiên dữ liệu không thực hiện hoặc thực hiện thiếu: Validation (kiểm tra), Encoding (mã hóa), Sanitize (làm sạch).

-> Trình duyệt không thể phát hiện payload độc hại.

Có thể chia thành 3 dạng tấn công:

* Stored (còn gọi là persistant – bền vững)
* Reflected (còn gọi là non-persistent – không bền vững)
* DOM-based (ít phổ biến)
* <script>aler(1)</script>

**Stored XSS Attack (Type I)**

* Script độc khi gửi đi được lưu trong server mục tiêu dưới dạng từ: tin nhắn forum, comment,…
* Nạn nhân truy cập dữ liệu đã được lưu script độc được gửi về.
* Stored XSS không cần attacker dựng một URL độc để thực hiện khai thác
* Tầm ảnh hưởng: bất cứ user nào có yêu cầu trang web đã được nhúng mã script đều bị ảnh hưởng.

**Reflected XSS Attack (Type II)**

Có thể xảy ra ở bất cứ trang web nào mà ở đó:

* Nhận input từ người dùng thông qua các tham số trong request GET hoặc POST.
* Thêm các tham số đó vào response trả về để hiển thị cho user (không có hoặc thiếu cơ chế kiểm tra, mã hóa dữ liệu trước khi hiển thị)

Script độc được thêm vào như một phần của URL hoặc tham số HTTP

Script độc sau đó chèn vào response được server gửi lại cho nạn nhân dưới dạng: thông báo lỗi, kết quả tìm kiếm, bất kỳ dạng response nào chứa input đã được gửi.

Tấn công được thực hiện bằng cách gửi cho nạn nhân một URL (thường là URL dạng GET request) qua email, mxh,...

Người dùng click vào URL và server gửi mã độc về trình duyệt người dùng để thực thi.

Tầm ảnh hưởng: chỉ những user click vào URL chứa mã độc.

**DOM-based Attack (Type 0)**

Nội dung trang web được server trả về không đổi, không chứa script độc, tuy nhiên code bên client thực thi khác thường do bị thay đổi cấu trúc DOM.

Script độc được chèn để chỉnh sửa cấu trúc DOM của trình duyệt client: thay đổi form, chuyển hướng submit, chặn không cho gửi giá trị về server,...

Tác động của XSS:

* Chiếm quyền điều khiển: cookie, session token
* Truy cập và gửi thông tin nhạy cảm
* Chỉnh sửa nội dung HTTP – Deface
* Keylog
* Yêu cầu nạn nhân tải virus
* XSS worm
* Tạo ra botnet để thực hiện DDOS

**SQL Injection**

Kỹ thuật cho phép lợi dụng lỗ hổng:

* Kiểm tra dữ liệu đầu vào trong các ứng dụng web
* Các thông báo lỗi trả về từ hệ quản trị cơ sở dữ liệu

Chèn và thực hiện các câu lệnh SQL bất hợp pháp

\* Đây là lỗ hổng ứng dụng web, không phải lỗi của CSDL hay web server.

Phân loại

* In-band SQL Injection

Attacker sử dụng cùng một kênh giao tiếp để thực hiện tấn công và nhận về kết quả

Một số dạng tấn công:

* + Dựa trên điều kiện luôn đúng (Tautology): 1 = 1
  + Thực hiện nhiều câu truy vấn (PiggyBacked Query)

Dựa trên Batched SQL Statements: một hoặc nhiều câu truy vấn được tách biệt nhau bằng dấu ;

A screen shot of a computer

Description automatically generated

* + Dựa vào thông báo lỗi kết hợp sử dụng Union (UNION SQL Injection)

Lệnh UNION kết hợp kết quả của 2 truy vấn SELECT có cùng số cột → Lợi dụng để lấy thêm dữ liệu từ CSDL trong kết quả của 1 truy vấn

- Lỗi MySQL: mã lỗi (thường 4 số) – mô tả

VD: 1064 - You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1.

- Lỗi SQL Server: Msg 105, Level 15, State 1, Line 1

Unclosed quotation mark after the character string ''.

* Blind/Inferential SQL Injection
  + Không thông điệp lỗi

Web application có thể bị tấn công SQL Injection nhưng kết quả không hiển thị cho attacker.

* + Trang web chung

Trong Blind SQL Injection, khi hệ thống có lỗi trong khi attacker tấn công luôn hiển thị một trang web chung, nên không thấy được các thông điệp báo lỗi hữu ích

* + Tốn thời gian

A screenshot of a computer

Description automatically generatedBiện pháp ngăn chặn

- Giới hạn quyền kết nối đến CSDL

- Sử dụng Prepared statement

- Kiểm tra dữ liệu được truyền vào có phải là kiểu dữ liệu mong đợi

- Escape dữ liệu truyền vào nếu không hỗ trợ Prepared Statement

- Không cho in ra màn hình bất kì thông tin cụ thể nào về CSDL, đặc biệt là lỗi

- Log lại các câu query → hữu dụng khi điều tra

* Out of band SQL Injection

**File Inclusion**

Có thể xuất hiện khi dùng tham số từ request của người dùng để chọn:

▪ Page sẽ load

▪ File code để server thực thi

→Nếu không có cơ chế kiểm tra tham số có thể khiến hệ thống hiển thị nội dung file hoặc thực thi code nhất định

Ví dụ: trong PHP với các lệnh: include(); require();

A screenshot of a computer program

Description automatically generated

Phân loại:

* Local File Inclusion: file nằm trên web server
* Remote File Inclusion: file nằm bên ngoài web server
* Có thể chứa server code → thực thi lệnh từ xa → làm hại toàn hệ thống
* Lưu ý:
  + Để khai thác RFI, server phải được cấu hình allow\_url\_fopen và allow\_url\_include
  + Có thể dùng lỗ hổng LFI để hiện file hệ thống:

../../../../../../etc/passwd

A screenshot of a computer

Description automatically generated

Giải pháp:

* Sử dụng Switch/Case
* Giới hạn độ dài: strlen(“$language”) < 4
* Giới hạn giá trị:

$available\_languages = array('eng', 'nor', 'ger');

in\_array($language, $available\_languages)

**Web Application Firewall (WAF)**

A diagram of a process

Description automatically generated

Ba mô hình bảo mật:

* Mô hình Positive Security
  + Từ chỗi tất cả ngoại trừ dữ liệu tốt
  + Có thể ngăn chặn các lỗ hổng Zero-day
  + An toàn hơn dạng blacklist
  + Cần có hiểu biết toàn diện về ứng dụng
  + Quá trình tạo chính sách tốn nhiều thời gian
* Mô hình Negative Security
  + Cho phép tất cả ngoại trừ dữ liệu xấu
  + Có thể áp dụng nhanh chóng
  + Cần ít hiểu biết hơn về ứng dụng
  + Có thể bảo vệ nhiều ứng dụng
  + Tiêu tốn tài nguyên
* Mô hình Hybrid Security: kết hợp cả 2 mô hình

Fingerprinting WAF: dựa vào cookie, dựa vào phản hồi HTTP (response)

**Android**

Kiến trúc ứng dụng di động bao gồm nhiều tầng, bao gồm 3 tầng cơ bản sau:

* Tầng Hiển thị: giao diện người dùng UI
* Tầng Xử lý: luồng hoạt động, các thành phần, thực thể xử lý logic dữ liệu
* Tầng dữ liệu: dữ liệu, truy cập dữ liệu

Android là một software stack cho các thiết bị di động, bao gồm hệ điều hành, middleware và các ứng dụng cốt lõi.

Android và các ứng dụng của nó được phát triển bằng ngôn ngữ JAVA/Kotlin.

Dalvik VM, được tùy chỉnh và tối ưu để chạy trên các thiết bị di động.

Android SDK cung cấp các công cụ cho phát triển ứng dụng Android và nhiều API hữu ích.

A screenshot of a web application

Description automatically generated

**Linux Kernel**

Lưu ý rằng: Android dựa trên kernel Linux không phải là một hệ điều hành Linux.

Cung cấp cơ chế Security, quản lý bộ nhớ (Memory management), quản lý tiến trình (Process management), Network stack và Driver model

Hoạt động như một lớp trừu tượng giữa phần cứng và các phần khác của software stack.

**Android Runtime**: cốt lõi của Android platform

Chức năng của các thư viện Java cốt lõi tùy vào Dalvik VM và Linux kernel bên dưới.

Có thể có nhiều Dalvik VMs chạy cùng lúc.

Mỗi ứng dụng Android chạy trong tiến trình riêng của nó, với một instance máy ảo Dalvik riêng.

Công cụ "dx" trong Android SDK: chuyển class JAVA đã biên dịch sang định dạng .dex

**DVM vs. JVM**

Dalvik Virtual Machine (DVM)

* Phát triển bởi Google
* Chạy các file thực thi Dalvik (.dex)
* Chỉ hỗ trợ một phần các thư viện Java chuẩn

JVM

* Phát triển bởi Sun
* Chạy Java bytecode.

**Applications**: tập các ứng dụng cốt lõi được cung cấp với Android platform, lập trình bằng Java.

Email client, SMS, calendar, maps, browser, contacts…

**Application Framework**

Đơn giản hóa việc sử dụng lại các components

* Activity Manager: quản lý vòng đời của các ứng dụng và cung cấp các điều hướng
* Notification Manager: cho phép các ứng dụng hiển thị các thông báo trên khung trạng thái
* Resource Manager: cung cấp khả năng truy cập vào các tài nguyên không phải mã nguồn như chuỗi, hình ảnh, các file layout…
* Content Providers: truy cập đến dữ liệu của các ứng dụng khác hoặc chia sẻ dữ liệu của chính ứng dụng
* Views: dùng để xây dựng ứng dụng, gồm danh sách, các khung nhập liệu, nút, hoặc thậm chí trình duyệt nhúng

Thành phần cốt lõi

* Activity
  + Về cơ bản, một Activity hiển thị một giao diện người dùng
  + Một ứng dụng có thể có 1 hoặc nhiều Activity, mỗi Activity kế thừa từ android.app.Activity và cần được định nghĩa trong AndroidManifest.xml
  + Activity được kích hoạt bằng các message bất đồng bộ được gọi là intents. Một Intent object chứa nội dung của message.
    - Explicit intent: intent nêu rõ tên thành phần (ví dụ activity) sẽ chạy → Android chạy thành phần này ngay lập tức
    - Implicit intent: intent chỉ định nghĩa hành động (action) cần chạy, Android sẽ tìm thành phần thỏa mãn trong file Manifest bằng intent-filters. Nếu nhiều hơn 1 kết quả, hệ thống sẽ hỏi người dùng.
* Service
  + Một service không có giao diện người dùng, chạy ngầm trong một khoảng thời gian nào đó
* Broadcast receivers
  + Thành phần nhận và phản ứng với những thông báo được broadcast
* Content providers
  + Một content provider cho phép cung cấp một số dữ liệu của ứng dụng cho các ứng dụng khác.
  + Dữ liệu có thể được lưu trữ trong file system, trong SQLite database, hoặc bất kỳ cách nào khác

**ProGuard**

Công cụ tích hợp sẵn trongAndroid Studio

Tính năng:

* Thu gọn mã nguồn ứng → để dễ phân phối
* Làm rối → để chống dịch ngược, vì tên các hàm, biến ... bị đổi tên khó đọc
* Tối ưu để ứng dụng chạy nhanh hơn.
* ProGuard là một công cụ rút gọn (shrink), tối ưu hoá (optimize) và làm mờ (obfuscate) code.
* Proguard-rules.pro là file cấu hình thêm về cách thức hoạt động cho ProGuard.

**Application Sandboxing**

Android gán một UID (thường được gọi là app ID), tại thời điểm cài đặt ứng dụng.

Ứng dụng Android được cô lập (isolated) ở cả 2 mức: tiến trình (process) và tập tin (file).

* Mỗi tiến trình của ứng dụng Android được thực thi như một user độc lập (với UID riêng biệt)
* Mỗi ứng dụng có 1 thư mục mà chỉ có ứng dụng đó được quyền đọc và ghi (phân quyền dựa trên UID)

Android không có file /etc/passwd, thông tin các UID được định nghĩa trong tập tin android\_filesystem\_config.h

Các daemon hệ thống chạy ngầm và ứng dụng được gắn với các UID, một vài daemons chạy dưới quyền root user (UID 0).

* UID của các dịch vụ hệ thống được bắt đầu 1000 (AID\_SYSTEM)
* UID của mỗi ứng dụng có giá trị từ 10000 trở lên (AID\_APP)

Dữ liệu của ứng dụng được chứa trong thư mục /data/data trên single-user device.

Các ứng dụng có thể được cài đặt bằng cách sử dụng cùng UID, được gọi là shared user ID.

→ chạy cùng tiến trình, chia sẻ dữ liệu với nhau

Để sử dụng cùng UID, các ứng dụng cần được ký bởi cùng code signing key.

**Quyền hạn (Permission)**

Permission là một chuỗi (string) qui định khả năng thực hiện một tác vụ nào đó.

Quyền hạn được gán cho từng ứng dụng tại thời điểm cài đặt ứng dụng bằng dịch vụ Package manager. Package manager được lưu trữ trong file /data/system/packages.xml

Một ứng dụng Android yêu cầu được cấp quyền bằng cách khai báo trong tập tin AndroidManifest.xml

Các ứng dụng có thể định nghĩa:

* Quyền hạn tùy chỉnh (Custom permission)
* Hạn chế truy cập (protection levels) → giới hạn truy cập tới các tài nguyên và dịch vụ của những ứng dụng được tạo bởi cùng một tác giả.

Protection level của permission đặc trưng cho những rủi ro có thể có bao hàm trong permission đó, và hệ thống nên làm gì để quyết định có cấp permission được yêu cầu hay không.

* normal: rủi ro thấp → permission sẽ tự động được cấp mà không đòi hỏi xác nhận từ người dùng.
* dangerous: các permission có thể cho phép truy cập dữ liệu người dùng hay các hình thức kiểm soát thiết bị → Trước khi cấp phép, Android hiển thị một thông báo cho người dùng chấp thuận.
* signature: chỉ được cấp các quyền hạn ở mức bảo vệ là signature cho các ứng dụng được ký bởi cùng một khóa bởi ứng dụng khai báo quyền đó. → Protection level mạnh nhất!
* signatureOrSystem: được cấp cho các ứng dụng hoặc một phần của system image, hoặc được ký với cùng một khóa với ứng dụng khai báo quyền hạn.

Giúp các vendor không cần chia sẻ khóa trên các ứng dụng cài đặt sẵn trên thiết bị.

Process Attribute Assignment: zygote và tiến trình của ứng dụng

* Tất cả các ứng dụng khởi chạy đều có tiến trình cha là zygote (PPID 181)
* Giúp khởi động nhanh tiến trình ứng dụng

**Credential Storage PBE**

Android hỗ trợ thuật toán mã hóa (Android password based encryption – PBE) sử dụng SHA-256 và AES, để tính khóa mã hóa đối xứng từ password (hashing, mã hóa, thực hiện nhiều lượt chu kỳ mã hóa)

Salt dùng để nối thêm vào mã hash của password → tăng độ khó của việc tìm ra khóa

**SSL/TLS trong Android**

Tạo kết nối HTTPS đến server công cộng

Sử dụng lớp HttpsURLConnection: hiện thực SSL/TLS trên class chuẩn HttpURLConnection

* getCipherSuite(): thuật toán mã hóa dùng trong kết nối
* getServerCertificates(): trả về các chứng chỉ mà máy chủ cung cấp.

**Inter-Process Communication - IPC**

Process Isolation: Các tiến trình Android độc lập với nhau.

Một số dịch vụ chia sẻ giữa các tiến trình đã có:

• File

• Socket

• Pipe

• Semaphore

• Shared memory

• Message queue,…

**IPC: Binder (New IPC)**

Binder: phiên bản IPC mới được phát triển cho Android, dựa trên OpenBinder.

Dựa trên kiến trúc thành phần phân tán (distributed component architecture)

Binder chỉ chạy trên một thiết bị, không hỗ trợ RPC (remote procedure call) thông qua mạng.

Các đối tượng Binder luôn được duy trì trong một định danh – identity duy nhất giữa các tiến trình.

Binder object có thể hoạt động như token bảo mật → hỗ trợ cơ chế capability-based security model của Android.

**IPC: Capability-based Security**

Các ứng dụng được cấp phép truy cập tới những tài nguyên nhất định bằng cách cấp cho chúng năng lực không thể giả mạo - vừa tham chiếu tới các đối tượng tài nguyên vừa đóng gói các quyền truy cập tới nó.

* Binder đảm bảo UID và PID của các tiến trình (caller) không thể bị giả mạo → nhiều dịch vụ hệ thống dùng binder để kiểm soát ứng dụng gọi tới API nhạy cảm thông qua IPC.
* Hệ thống sử dụng package database để xác định quyền hạn mà các thành phần được gọi (callee) yêu cầu; sau đó ánh xạ caller UID với package name và lấy danh sách các quyền hạn đã cấp cho caller.

Android Binder không sử dụng cơ chế an ninh ACL (access control list)

Đối tượng Binder hoạt động như một năng lực (capability), và được gọi là Binder token.

Việc sở hữu Binder token cho phép tiến trình có toàn quyền truy cập tới Binder → cho phép nó thực thi các giao dịch Binder trên đối tượng đích (target).

Truy cập đối tượng Binder:

* Binder sử dụng context manager (tham chiếu tới Binder) để cho phép xác định các dịch vụ (service discovery)
* Trong Android, context manager có tên là service manager – khởi chạy trong quá trình boot.
* Các dịch vụ được đăng ký với hệ thống bằng cách truyền vào service manager tên của dịch vụ và tham chiếu Binder.
* Sau khi đăng ký, các dịch vụ của hệ thống có thể được truy cập bằng tên thông qua tham chiếu Binder.
* Cơ chế kiểm tra quyền hạn qui định việc truy cập vào Binder chỉ giới hạn trên một số tiến trình hệ thống được đăng ký.

**Code Signing & Platform Keys**

Tập tin cài đặt APK là một extension của định dạng file JAR → việc ký lên ứng dụng APK cũng dựa trên quy trình ký lên tập tin JAR.

Android sử dụng chữ ký trên file APK để đảm bảo:

* Việc cập nhật ứng dụng luôn đến từ một nhà phát triển (gọi là same origin policy).
* Tạo mối quan hệ tin cậy giữa các ứng dụng
* Thực hiện so sánh giữa phiên bản hiện tại với phiên bản cập nhật của 1 ứng dụng.

Các ứng dụng hệ thống (System app) được ký bởi một số các platform key. Các thành phần hệ thống có thể chia sẻ tài nguyên và chạy trong cùng một tiến trình khi nó được ký bởi cùng 1 platform key.

**SEAndroid**

* Discretionary access control (DAC): user có quyền truy cập đến file có thể chuyển quyền cho user khác tuỳ ý
* Mandatory access control (MAC): dựa trên policy, chỉ có thể sửa bởi quản trị viên
* Security Enhanced Linux (SELinux) là một phiên bản của MAC cho Linux kernel.

**Verify Boot**

Phiên bản Android 4.4 hỗ trợ xác thực boot bằng verity (của Linux’s Device Mapper).

Kiểm tra tính toàn vẹn của các block trên thiết bị dựa vào cây mã hóa:

* Nút lá: giá trị băm của một khối dữ liệu (data block)
* Nút trung gian: chứa giá trị băm của các nút con của nó
* Nút gốc: chứa giá trị băm của tất cả các nút trực tiếp thuộc về nó.

→chỉ cần dựa vào nút gốc

Quá trình xác thực được thực hiện bởi một khóa công cộng RSA trong mỗi phân vùng boot.

Từng block của thiết bị sẽ được kiểm tra lúc runtime bằng cách tính giá trị băm của block với giá trị được ghi trong cây băm.

Bởi vì tất cả các kiểm tra được thực hiện bởi kernel, do đó qui trình boot cần xác thực tính toàn vẹn của kernel nhằm cho các verified boot hoạt động.

A screenshot of a computer

Description automatically generated

**M1: Improper credential usage**

* Liên quan đến khía cạnh quản lý và bảo vệ thông tin xác thực như: API key, khoá bí mật, khoá công khai,...
* Thông tin xác thực được mã hoá cứng (hardcoded) và việc sử dụng thời gian để tạo khoá dễ bị khai thác.
* Thông tin khoá được lưu trữ hoặc sử dụng không đúng cách.
* Thông tin xác thực được gửi đi không được mã hoá thích hợp dẫn đến bị đánh cắp.
* Không áp dụng các quy trình xác thực nhiều bước (MFA).
* Không triển khai hoặc quản lý phiên truy cập (session) lỏng lẻo.

**M2: Inadequate Supply Chain Security**

* Kẻ tấn công tận dụng các thư viện, package bên thứ ba sử dụng trong ứng dụng để khai thác mà không cần tấn công vào ứng dụng (Tấn công chuỗi cung ứng).
* Tấn công chuỗi cung ứng sẽ lâu dài nên đòi hỏi phải kiểm tra thường xuyên các đoạn mã (review code) trong ứng dụng và kiểm soát các bản cập nhật của ứng dụng.

**M3: Insecure authentication/Authorization**

* Lỗ hổng này liên quan đến khâu xác thực người dùng và quản lý phiên đăng nhập
* M3 bao gồm các yếu tố sau:
  + Không xác thực, định danh được người dùng
  + Không thể duy trì định danh của người dùng khi có yêu cầu
  + Gặp điểm yếu trong quy trình quản lý phiên
  + Phân quyền có sai sót hoặc không phân quyền
  + Có khả năng bị khai thác leo thang đặc quyền

**M4: Insufficient Input/Output Validation**

* Các lỗ hổng phổ biến trong API do dữ liệu đầu vào bị sửa đổi hoặc không đúng định dạng dẫn tới việc chèn các câu lệnh khai thác, mã thực thi.
* Tấn công phổ biến: SQL Injection, Command Injection, XSS,...

**M5: Insecure communication**

* Kết nối dữ liệu không được mã hóa và xác thực.
* poor handshaking/weak negotiation (f. ex. lack of certificate pinning)
* SSL version không thích hợp
* Giao tiếp với dữ liệu dạng cleartext của các thông tin nhạy cảm
* Sử dụng HTTP thay vì HTTPS

VD: Lấy thông tin GPS theo thời gian thực trên đồng hồ trẻ em (smartwatch)

**M6: Inadequate Privacy Controls**

* Thông tin và dữ liệu riêng tư của người dùng có khả năng không được kiểm soát đầy đủ. Bao gồm các dữ liệu các nhân và việc tuân thủ quyền riêng tư như GDPR, CCPA.
* Dữ liệu có thể bị rò rỉ qua nhiều cách thức khác nhau, có thể do ứng dụng lưu trữ nhiều dữ liệu dư thừa của người dùng mà không dùng đến hoặc việc bị kẻ tấn công lợi dụng để lấy thông tin người dùng.

**M7: Insufficient Binary Protections**

* Buffer overflow
* Format string
* Binary patching
* Local resource modification
* Method hooking and swizzling
* Dynamic memory modification

Ví dụ: Ứng dụng Pokemon GO bị dịch ngược, thêm vào các vị trí giả mạo nhằm tìm kiếm các Pokemon hiếm và làm cho trứng nở nhanh hơn.

**M8: Security Misconfiguration**

* Cấu hình sai về bảo mật dễ xảy ra do việc kiểm soát tính năng, phân quyền ở một số phần mềm là quá phức tạp, dẫn tới việc bỏ sót hoặc cấp quyền không cần thiết.
* Áp dụng nguyên tắc “đặc quyền tối thiểu” trong phân quyền dữ liệu.

**M9: Insecure data storage**

* Lưu trữ dữ liệu không an toàn
* Rò rỉ dữ liệu vô ý

Bao gồm:

* wrong keychain accessibility option
* insufficient file data protection
* access to privacy resources when using this data incorrectly.

**M10: Insufficient cryptography**

* Sử dụng một thuật toán mã hóa đã lỗi thời/dễ bẻ khóa; hoặc tự viết một thuật toán mã hóa có lỗ hổng
* VD: Lỗ hổng trên ứng dụng Ola (dịch vụ thuê xe như Grab) tại Ấn Độ do Appknox công bố → ảnh hưởng đến ví tiền trong ứng dụng.
  + Sử dụng thuật toán mã hoá không mạnh
  + Appknox phát hiện ra Key dùng để mã hóa là: PRODKEYPRODKEY12

**Fuzzy testing**

Hàm reverse có chức năng nhận giá trị đầu vào là một danh sách → cho kết quả trả về là một danh sách bị đảo ngược.

Fuzzy testing (tự phát sinh dữ liệu test)

Fuzz Testing là một loại kiểm thử trong đó các kỹ thuật kiểm tra tự động hoặc bán tự động được sử dụng để phát hiện lỗi mã hóa và lỗ hổng bảo mật trong phần mềm, hệ điều hành hoặc mạng bằng cách nhập dữ liệu không hợp lệ hoặc ngẫu nhiên (gọi là FUZZ) vào hệ thống.

Fuzz testing là một trong những kỹ thuật thử nghiệm hộp đen.

Chiến lược kiểm tra:

* Bước 1: Xác định hệ thống đích
* Bước 2: Xác định đầu vào
* Bước 3: Tạo dữ liệu mờ
* Bước 4: Thực hiện kiểm tra bằng cách sử dụng dữ liệu mờ
* Bước 5: Theo dõi hành vi hệ thống
* Bước 6: Tổng hợp lỗi

Ưu điểm

* Cải thiện kiểm thử bảo mật cho phần mềm/hệ thống
* Có thể tìm thấy những lỗ hổng nghiêm trọng như crash, rò rỉ bộ nhớ, không xử lý ngoại lệ,…
* Tìm thấy các bug mà ít nhận được sự chú ý của nhân viên kiểm thử
* Những lỗi không được tìm thấy khi kiểm thử bị hạn chế về thời gian và nguồn lực thì cũng được kiểm thử fuzzing tìm ra

Nhược điểm

* Không thể đưa ra báo cáo tổng thể về bảo mật nếu chỉ áp dụng mỗi phương pháp Fuzz testing
* Ít hiệu quả trong các trường hợp xử lý các mối đe dọa bảo mật không gây ra sự cố chương trình (như worm, trojan, virus,…)
* Đòi hỏi nhiều thời gian
* Thiết lập điều kiện biến đầu vào ngẫu nhiên là một việc khó, đòi hỏi nhiều công sức/ thuật toán

**Report**

Report format

* Cover page

Cung cấp tên báo cáo, phiên bản, ngày, tác giả, tên nhà cung cấp dịch vụ và bên nhận báo cáo.

Các mục bổ sung:

• Phân loại bảo mật tài liệu

• Nêu bật kết quả từ các phần khác

* Confidentiality statement

Bảo vệ thông tin thu thập được trong quá trình tham gia

Cần giải thích:

• mức độ bảo mật liên quan đến tài liệu

• ai được phép xem tài liệu

• những gì được phép và không được phép sao chép, quyền phân phối

• ngôn ngữ pháp lý khác

* Document Control

Liệt kê phiên bản và các chỉnh sửa được thực hiện đối với đề xuất giao hàng

Giúp người đọc tận dụng phiên bản mới nhất.

* Timeline

Cung cấp ước tính số giờ cho từng giai đoạn của dự án:

• tên giai đoạn

• các nhiệm vụ cần hoàn thành

• thời gian dự kiến

* Executive summary

Cung cấp một cái nhìn tổng quan cao cấp về lý do tại sao các dịch vụ được thực hiện

Bao gồm những gì đã dẫn đến vấn đề được giải quyết, tình huống có vấn đề và giải pháp đề xuất với kết quả mong đợi.

* Methodology

Cung cấp tổng quan về cách bạn cung cấp dịch vụ:

• quy trình cho từng giai đoạn

• các công cụ được sử dụng

• cách bạn xử lý các mối đe dọa đã xác định

▪ Phát triển các sơ đồ minh họa luồng quy trình và cấu trúc báo cáo tài nguyên

**Broken Access Control**

Các mô hình kiểm soát truy cập:

* Discretionary access control (DAC)

Chủ sở hữu xác định quyền hạn và cấp quyền hạn cho một nhóm. Chỉ phù hợp cho mô hình nhỏ, dễ quản lý.

* Mandatory access control (MAC)

Phân chia nhiều quy tắc, chính sách chặt chẽ theo cấp bậc và mức độ quan trọng. Cấp cao hơn sẽ bao gồm quyền hạn của cấp thấp hơn.

* Role-based access control (RBAC)

Phân chia cụ thể vai trò, các vị trí, quyền hạn và tài nguyên với từng user. Đảm bảo quyền hạn phù hợp.

Các dạng kiểm soát truy cập:

* Vertical access controls – kiểm soát truy cập chiều dọc
* Horizontal access controls – kiểm soát truy cập chiều ngang
* Context-dependent access controls – kiểm soát truy cập phụ thuộc ngữ cảnh

Lỗ hổng kiểm soát truy cập:

* Lỗ hổng kiểm soát truy cập theo chiều dọc: thông qua thay đổi giá trị các tham số khiến người dùng bình thường có thể truy cập vào trang quản trị.
* Lỗ hổng kiểm soát truy cập theo chiều ngang: lỗ hổng trong dạng kiểm soát truy cập theo chiều ngang qua các tham số xác định duy nhất người dùng.
* Lỗ hổng kiểm soát truy cập nhiều bước: trong mỗi bước nếu việc kiểm soát quyền truy cập từ người dùng không được chặt chẽ, kẻ tấn công có thể lợi dụng tính năng này bỏ qua một số bước để thực hiện mục đích xấu.
* Lỗ hổng kiểm soát truy cập thông qua header referer.

Khuyến nghị ngăn chặn

* Không dựa vào duy nhất một yếu tố để kiểm soát truy cập.
* Từ chối trực tiếp quyền truy cập theo mặc định trừ khi tài nguyên được dành cho truy cập công khai.
* Tại bất cứ bước nào trong hệ thống đều cần xác thực lại quyền kiểm soát truy cập của người dùng.
* Quy định các quyền truy cập chặt chẽ đối với từng người dùng.
* Thường xuyên kiểm tra, xem xét kỹ lưỡng, cập nhật, nâng cấp các biện pháp kiểm soát truy cập để đảm bảo chúng hoạt động bình thường.
* Xác thực danh tính người dùng: cần thực hiện xác thực danh tính người dùng qua nhiều khía cạnh.
* Ủy quyền: Quy định chặt chẽ các quyền được phép truy cập và quyền truy cập trái phép của người dùng.
* Kiểm tra quyền kiểm soát truy cập của người dùng trong mọi giai đoạn thuộc bất kì hành vi nào do người dùng thực hiện hoặc yêu cầu.
* Hạn chế CORS: Giao thức CORS (Chia sẻ tài nguyên đa nguồn gốc) cung cấp một cách thức có kiểm soát để chia sẻ tài nguyên có nguồn gốc chéo.
* Việc triển khai CORS dựa trên tiêu đề giao thức HTTP được sử dụng trong giao tiếp giữa máy khách và ứng dụng đích. Khi giao thức CORS bị định cấu hình sai, một bên độc hại có thể kiểm soát miền để gửi yêu cầu đến miền của bạn.

Trắc nghiệm

* TOP 10 OWASP 2021
* TOP 10 OWASP Mobile 2024
* CSRF/SSRF
* APK file
* SQL injection / NoSQL
* LFI / RFI
* Các thành phần có trong android: activity, intents, …
* Cơ chế IPC
* Php đang xài version bao nhiêu và version nào đã lỗi thời

Tự luận

1. Đưa đoạn code php 1 hàm có lỗ hổng gì đó ????

* Xác định điểm yếu, nguy cơ, trình bày cách khai thác (nếu có)
* Phương pháp bảo vệ (mitigation)

1. Cơ chế bảo mật đường truyền trong android app: mô tả, triển khai chức năng
2. CSP
3. Đăng nhập trong android