**TÀI LIỆU LẬP TRÌNH AN TOÀN**

Nội dung

[1. Nội dung sổ tay lập trình 5](#_Toc447206311)

[2. Thế nào là lập trình an toàn? 5](#_Toc447206312)

[3. Các khái niệm và thuật ngữ liên quan 5](#_Toc447206313)

[4. Các lổ hổng, rủi ro bảo mật phổ biến 6](#_Toc447206314)

[4.1. SQL Injection 6](#_Toc447206315)

[4.1.1 Khái niệm 6](#_Toc447206316)

[4.1.2 Cách phòng chống 6](#_Toc447206317)

[4.2. Broken Authentication and Session Management 7](#_Toc447206318)

[4.2.1 Khái niệm 7](#_Toc447206319)

[4.2.2 Cách phòng chống 7](#_Toc447206320)

[4.3. Cross-Site Scripting (XSS) 7](#_Toc447206321)

[4.3.1 Khái niệm 7](#_Toc447206322)

[4.3.2 Cách phòng chống 8](#_Toc447206323)

[4.4. Insecure Direct Object References 8](#_Toc447206324)

[4.4.1 Khái niệm 8](#_Toc447206325)

[4.4.2 Cách phòng chống 9](#_Toc447206326)

[4.5. Sensitive Data Exposure 9](#_Toc447206327)

[4.5.1 Khái niệm 9](#_Toc447206328)

[4.5.2 Cách phòng chống 9](#_Toc447206329)

[4.6. Missing Function Level Access Control 10](#_Toc447206330)

[4.6.1 Khái niệm 10](#_Toc447206331)

[4.62 Cách phòng chống 10](#_Toc447206332)

[4.7. Cross-Site Request Forgery (CSRF) 10](#_Toc447206333)

[4.7.1 Khái niệm 10](#_Toc447206334)

[4.7.2 Cách phòng chống 11](#_Toc447206335)

[4.8. Using Known Vulnerable Components 11](#_Toc447206336)

[4.8.1 Khái niệm 11](#_Toc447206337)

[4.8.2 Cách phòng chống 11](#_Toc447206338)

[4.9. Unvalidated Redirects and Forwards 11](#_Toc447206339)

[4.9.1 Khái niệm 11](#_Toc447206340)

[4.9.2 Cách phòng chống 12](#_Toc447206341)

[5. Các lổ hổng bảo mật trên mobile 13](#_Toc447206342)

[5.1. Insecure Data Storage 13](#_Toc447206343)

[5.1.1 Khái niệm 13](#_Toc447206344)

[5.1.2 Cách phòng chống 13](#_Toc447206345)

[5.2. Insufficient Transport Layer 14](#_Toc447206346)

[5.2.1 Khái niệm 14](#_Toc447206347)

[5.2.2 Cách phòng chống 14](#_Toc447206348)

[5.3. Unintended Data Leakage 15](#_Toc447206349)

[5.3.1 Khái niệm 15](#_Toc447206350)

[5.3.2 Cách phòng chống 15](#_Toc447206351)

[5.4. Broken Cryptography 16](#_Toc447206352)

[5.4.1 Khái niệm 16](#_Toc447206353)

[5.4.2 Cách phòng chống. 16](#_Toc447206354)

[5.5. Client Side Injection 16](#_Toc447206355)

[5.5.1 Khái niệm 16](#_Toc447206356)

[5.5.2 Cách phòng chống 17](#_Toc447206357)

[5.6. Security Decisions Via Untrusted Inputs 18](#_Toc447206358)

[5.6.1 Khái niệm 18](#_Toc447206359)

[5.6.2 Cách phòng chống 18](#_Toc447206360)

[5.7. Improper Session Handling 18](#_Toc447206361)

[5.7.1 Khái niệm 18](#_Toc447206362)

[5.7.2 Cách phòng chống 19](#_Toc447206363)

[5.8. Lack of Binary Protections 19](#_Toc447206364)

[5.8.1 Khái niệm 19](#_Toc447206365)

[5.8.2 Cách phòng chống 19](#_Toc447206366)

[6. Tham khảo 21](#_Toc447206367)

# 1. Nội dung sổ tay lập trình

“Sổ tay lập trình an toàn” cung cấp thông tin cho đội ngũ phát triển ứng dụng về các khái niệm bảo mật cũng như những lổ hổng nguy hiểm, từ đó nâng cao ý thức bảo mật và tăng cường an ninh cho ứng dụng.

# 2. Thế nào là lập trình an toàn?

Ứng dụng được bảo mật là như thế nào?

Một ứng dụng được bảo mật cần đảm bảo ba tính chất bảo mật (*Confidentiality*), toàn vẹn (*Integrity*) và sẵn sàng (*Availability*).

Khái niệm CIA (Confidentiality, Integrity, Availability)?

Tính bảo mật (*Confidentiality*), tính toàn vẹn (*Integrity*), và tính sẵn sàng (*Availability*) là ba yếu tố đảm bảo khả năng an ninh thông tin của một hệ thống. Đây là ba yếu tố quan trọng nhất của hệ thống.

* Tính bảo mật (*Confidentiality*): Đảm bảo thông tin chỉ được truy cập bởi đối tượng được cấp phép, ngăn chặn việc tiết lộ thông tin một cách trái phép.
* Tính toàn vẹn (*Integrity*): Đảm bảo tính toàn vẹn của thông tin, thông tin chỉ được phép xóa hoặc sửa bởi những đối tượng được phép và đảm bảo thông tin vẫn nguyên vẹn khi được lưu trữ hay truyền đi.
* Tính sẵn sàng (*Availability*): Đảm bảo độ sẵn sàng của thông tin, thông tin có thể được truy cập bởi những đối tượng được phép vào bất cứ khi nào.

Lập trình an toàn là gì?

Lập trình an toàn hạn chế thấp nhất các lổ hổng, rủi ro có thể ảnh hưởng đến an ninh của ứng dụng. Để có thể làm được điều này, người phát triển ứng dụng cần có kiến thức về những lổ hổng, rủi ro có thể xảy ra cho ứng dụng cũng như đảm bảo các nguyên tắc an toàn khi phát triển ứng dụng.

# 3. Các khái niệm và thuật ngữ liên quan

OWASP (Open Web Application Security Project): Một dự án mở về bảo mật ứng dụng web, cung cấp các thông tin, công cụ và tiêu chuẩn nhằm đáp ứng nhu cầu nâng cao bảo mật thông tin cho mọi đối tượng.

# 4. Các lổ hổng, rủi ro bảo mật phổ biến

## 4.1. SQL Injection

### 4.1.1 Khái niệm

Injection là cách lợi dụng những lỗ hổng trong quá trình lập trình ứng dụng về phần truy xuất cơ sở dữ liệu.

Khi attacker gửi những dữ liệu (thông qua các form), ứng dụng sẽ thực hiện và trả về cho kết quả câu truy vấn hay những thông báo lỗi có liên quan đến cơ sở dữ liệu. Và nhờ những thông tin này mà attacker biết được nội dung cơ sở dữ liệu và từ đó có thể điều khiển toàn bộ hệ thống ứng dụng.

* Vi dụ:

Ứng dung được xây dựng việc truy vấn SQL login như sau:

String query = “ Select \* From accounts WHERE custID=”’ + request.getParameter(“id”) + “”;

Attacker có thể thay đổi tham số ‘id’ trong trình duyệt để gửi đến: ’or’1’=’1. Việc thay đổi ý nghĩa của câu truy vấn và trả ra giá trị của tất cả các tài khoản trong cơ sớ dữ liệu thay vì chỉ của 1 nhân viên mà thôi.

[http://vabank.vn/app/accountView?id=’or’1’=’1](http://vabank.vn/app/accountView?id='or'1'='1).

### 4.1.2 Cách phòng chống

- Kiểm tra dữ liệu đầu vào hợp lệ bằng các từ điển tiêu chuẩn phù hợp hay sử dụng OWASP’s ESAPI để kiểm tra dữ liệu đầu vào.

- Kiểm tra kĩ giá trị nhập vào của người dùng, thay thế những kí tự như ‘ ; v.v…

- Loại bỏ các kí tự meta như “',",/,\,;“ và các kí tự extend như NULL, CR, LF, ... trong các string nhận được từ:

* Dữ liệu nhập do người dùng nhập vào.
* Các tham số từ URL.
* Các giá trị từ Cookie.

- Đối với các giá trị Numeric, chuyển sang Integer trước khi thực hiện câu truy vấn SQL, hoặc dùng ISNUMERIC để chắc chắn đó là một số Integer.

- Dùng thuật toán để mã hoá dữ liệu.

## 

## 4.2. Broken Authentication and Session Management

### 4.2.1 Khái niệm

Broken Authentication là phương pháp attacker hoặc người sử dụng sở hữu tài khoản muốn đánh cắp thông tin từ 1 tài khoản khác.

Session Management gồm 2 loại:

**Session Fixation:** là kĩ thuật tấn công cho phép attacker mạo danh người dùng hợp lệ bằng cách gửi một session ID hợp lệ đến người dùng, sau khi đăng nhập vào hệ thống thành công, attacker sẽ dùng lại session ID đó và nghiễm nhiên trở thành người dùng hợp lệ.

**Session Hijacking:** kĩ thuật tấn công cho phép hacker mạo danh người dùng hợp lệ sau khi nạn nhân đã đăng nhập vào hệ thống bằng cách giải mã session ID của họ được lưu trữ trong cookie hay tham số URL, biến ẩn của form.

* Ví dụ 1: Ứng dụng đặt vé máy bay cho phép viết lại URL, đặt id làm việc trong URL.

http://vabank.com/sale/saleitems;jsessionid=2P00C2JDPXM0OQSNDLPSKHCJUN2JV?dest=Hawai.

Attacker có thể chiếm session làm việc bằng cách sử dụng session id này.

* Ví dụ 2: Password lưu trữ của người dùng trên database không được mã hóa. Attacker hoặc người dùng trong nội bộ có thể xâm nhập vào hệ thống dễ dàng ăn cắp mật khẩu mà không bị phát hiện.

### 4.2.2 Cách phòng chống

Sử dụng hoặc tạo ra các phương thức định danh mạnh.

* Đáp ứng các yêu cầu theo định danh: Application Security Verification Standard (ASVS)

Giao diện đơn giản cho người phát triển ứng dụng có thể follow theo: ESAPI Authentication and User APIs.

* <https://www.owasp.org/index.php/ESAPI_Secure_Coding_Guideline>
* <https://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/Authenticator.html>
* <https://owasp-esapi-java.googlecode.com/svn/trunk_doc/latest/org/owasp/esapi/User.html>

## 4.3. Cross-Site Scripting (XSS)

### 4.3.1 Khái niệm

Phương pháp Cross Site Scripting (được viết tắt là XSS) là phương pháp tấn công bằng cách chèn thêm những đoạn mã có khả năng đánh cắp hay thiết lập được những thông tin quan trọng như cookies, mật khẩu,… vào mã nguồn ứng dụng web để từ đó chúng được chạy như là một phần của ứng dụng Web và có chức năng cung cấp hoặc thực hiện những những điều attacker muốn.

Phương pháp này không nhằm vào máy chủ hệ thống mà chủ yếu tấn công trên chính máy người sử dụng. Attacker sẽ lợi dụng sự kiểm tra lỏng lẻo từ ứng dụng và hiểu biết hạn chế của người dùng cũng như biết đánh vào sự tò mò của họ dẫn đến người dùng bị mất thông tin một cách dễ dàng. Thông thường hacker lợi dụng địa chỉ URL để đưa ra những liên kết là tác nhân kích hoạt những đoạn chương trình được viết bằng ngôn ngữ máy khách như VBScript, JavaScript…được thực thi trên chính trình duyệt của nạn nhân.

Ví dụ:

<http://vabank.vn/index1.html>

sau khi thêm

[https://vabank.vn/index1.html?tw= <script>alert(document.cookie);</script](https://vabank.vn/index1.html?tw=%20%3cscript%3ealert(document.cookie);%3c/script)>

hay

<http://vabank.com.vn/index1.html?=%3Cscript%3Ealert%28document.cookie%29%3C%2Fscript%3E>

Đoạn mã trên được thêm vào nhằm mục đính lấy cookies của nạn nhân.

### 4.3.2 Cách phòng chống

Với dữ liệu, thông tin nhập của người dùng, người thiết kế ứng dụng Web cần phải thực hiện vài bước cơ bản sau:

* Tạo ra danh sách những thẻ HTML được phép sử dụng.
* Xóa bỏ thẻ <script>
* Lọc ra bất kì một đoạn mã JavaScript/Java/VBScript/ActiveX/Flash Related nào.
* Lọc dấu nháy đơn hay kép.
* Lọc kí tự Null ( vì khả năng thêm một đoạn mã bất kì sau kí tự Null khiến cho ứng dụng dù đã lọc bỏ Script vẫn không nhận ra do ứng dụng nghĩ rằng chuôĩ đã kết thức từ kí tư Null này..

## 4.4. Insecure Direct Object References

### 4.4.1 Khái niệm

Điều này xảy ra khi ứng dụng cho thấy có các tham chiếu trực tiếp đến một đối tượng nội bộ hay của người dùng khác, ví dụ như một tập tin, thư mục, hay cơ sở dữ liệu quan trọng, mà ko có sự kiểm tra hay bảo vệ an toàn cần thiết. Điều này cho phép kẻ tấn công có thể truy cập các tài liệu này một cách trái phép.

Ví dụ:

Ứng dụng truy vấn SQL truy cập đến thông tin tài khoản.

String query = “SELECT \* FROM accts WHERE account =?”;

PreparedStatement pstmt = connection.prepareStatement(query,…);

pstmt.setString(1, request.getparameter(“acct”));

ResultSet results = pstmt.executeQuery();

Attacker có thể dễ dàng thay đổi thông số acct trên trình duyệt để truy cập đến tài khoản khác.

### 4.4.2 Cách phòng chống

- Xác định mỗi liên kết đối tượng cho từng người dùng hoặc mỗi phiên làm việc. Việc này có thể ngăn chặn attacker nhắm đến các dữ liệu không được bảo vệ. Tham khảo thêm OWASP’s ESAPI.

- Kiểm tra việc truy cập. Đối với mỗi tham chiếu trực tiếp từ một người không xác thực phải được kiểm tra, chứng thực để chắc rằng người dùng được quyền truy cập đến đối tượng yêu cầu.

## 4.5. Sensitive Data Exposure

### 4.5.1 Khái niệm

Các dữ liệu nhạy cảm không được lưu trữ và bảo vệ cẩn thận, dẫn đến khi bị kẻ tấn công khai thác gây ra những ảnh hưởng to lớn cho hệ thống máy chủ, doanh nghiệp, khách hàng. Ví dụ như việc lưu trữ thẻ tín dụng mà ko thông qua các việc mã hóa, hay các gói tin TLS bị bẻ khóa và nghe lén thông qua lỗ hổng CRIME.

Ví dụ: Website không sử dụng giao thức TLS mạnh đối với việc chứng thực đăng nhập dễ dàng bị attacker nghe lén và cướp user’s session cookie.

### 4.5.2 Cách phòng chống

- Đảm bảo tất cả các dữ liệu nhạy cảm phải được mã hóa và dữ liệu truyền đi tới client cũng phải được mã hóa trên đường truyền.

- Không chứa, thể hiện các thông tin nhạy cảm ở những nơi không thật sự cần thiết.

- Đảm bảo các thuật toán mạnh và key mã hóa mạnh được sử dụng; nơi quản lý key phải được bảo vệ an toàn.

- Tắt các tính năng các form tự động lưu các thông tin nhạy cảm và tắt tính năng caching đối với các trang chứa thông tin nhạy cảm.

## 4.6. Missing Function Level Access Control

### 4.6.1 Khái niệm

Thiếu các điều khoản trong việc phân quyền quản trị các mức, dẫn đến việc kẻ tấn công có thể lợi dụng và truy ra các điểm yếu trên hệ thống, hay lợi dụng để leo thang đặc quyền.

Vi dụ:

Attacker tìm thấy trang đăng nhập cho nên cũng dễ dàng truy cập tới trang đăng nhập bằng admin

http://example.com/app/getappInfo  
<http://example.com/app/admin_getappInfo>

Đây là 1 lỗi mà attacker có thể khai thác để tìm cách dò tìm vào trang admin bằng các phương thức bẻ khóa hay dò tìm mật khẩu.

### 4.62 Cách phòng chống

- Đảm bảo quản lý toàn bộ việc phân quyền quản trị hay đăng nhập, dễ dàng cập nhật và giám sát.

- Thực thi cơ chế mặc định nên từ chối tất cả các truy cập, yêu cầu phân quyền rõ ràng tới từng vai trò cụ thể đối với mỗi chức năng truy cập.

## 4.7. Cross-Site Request Forgery (CSRF)

### 4.7.1 Khái niệm

CSRF là kĩ thuật tấn công buộc một người dùng cuối để thực hiện các hành động không mong muốn trên ứng dụng web mà họ đang thực hiện. Kẻ tấn công có thể đánh lừa người dùng thực hiện các hành động mà attacker mong muốn. Nếu nạn nhân là một người dùng bình thường, một cuộc tấn công CSRF thành công có thể buộc người dùng thực hiện các yêu cầu như chuyển tiền, thay đổi địa chỉ email, v.v.. Nếu nạn nhân là tài khoản quản trị, CSRF có thể làm ảnh hưởng đến toàn bộ ứng dụng web.

Ví dụ:

Ứng dụng cho phép người dùng gửi đi những request không mang tính bảo mật như:

http://example.com/app/transferFunds?amount=1500&destinationAccount=467324324

Chính vì vậy attacker sẽ lợi dụng điều này để nhúng request không an toàn vào một thẻ ảnh hoặc iframe mà người dùng không hay biết với width=0, height=0.

<img src="http://example.com/app/transferFunds?amount=1500&destinationAccount=attackersAccount#" width="0" height="0"

Khi người dùng đã xác thực tại trang “example.com” và truy cập đến website đã nhúng đoạn mã trên thì request không hợp lệ trên sẽ được thực thi ngầm.

### 4.7.2 Cách phòng chống

Trong các tương tác của người dùng với cơ sở dữ liệu thông qua các form, liên kết, sử dụng thêm biến token (được tạo ra mỗi đầu phiên truy cập của người dùng) như một tham số trong phương thức GET hoặc POST và kiểm tra giá trị token này tại máy chủ để xác nhận hành vi của người dùng.

## 4.8. Using Known Vulnerable Components

### 4.8.1 Khái niệm

Sử dụng các thư viện, plugin, module… có chứa các lỗ hổng đã được công khai dẫn đến việc bị kẻ tấn công lợi dụng để tấn công vào hệ thống một cách nhanh chóng. Các developer không quan tâm đến các phiên bản cũng như lổ hổng được công khai của các thư viện, plugin, module… xây dựng ứng dụng.

Ví dụ:

Ứng dụng được viết trên một nền tảng phổ biến và mạnh. Khi lỗ hổng XSS được phát hiện và có bản vá nhưng bị bỏ qua. Attacker hoàn toàn có thể lợi dụng những lổ hổng này.

### 4.8.2 Cách phòng chống

Xác định các component và phiên bản đang sử dụng đã được cập nhật hoặc không có lổ hổng bảo mật.

## 4.9. Unvalidated Redirects and Forwards

### 4.9.1 Khái niệm

Attacker sẽ chuyển hướng người dùng đến một đường dẫn bên ngoài được chuẩn bị sẵn của kẻ tấn công hoặc chính website đã bị attacker lợi dụng để chuyển hướng đến một trang đích khác. Vì những đường dẫn bắt nguồn từ website hợp lệ nên khả năng tấn công rất cao.

Ví dụ 1:

Ứng dụng có một trang "redirect.jsp"có một tham số là"url". Kẻ tấn công tạo một URL độc hại để hướng người dùng đến một trang web độc hại để thực hiện lừa đảo (phishing) và cài đặt các phần mềm độc hại.

http://www.example.com/redirect.jsp?url=evil.com

Ví dụ 2:

Ứng dụng forward các request giữa các site của trang web bằng tham số “fwd”. Trong trường hợp này, attacker sẽ tạo ra một URL để vượt qua access control của ứng dụng và sau đó chuyển tiếp attacker vào giao diện quản trị.

http://www.example.com/boring.jsp?fwd=admin.jsp

### 4.9.2 Cách phòng chống

- Ngăn chặn việc chuyển hướng (redirect) và chuyển tiếp (forward) trong ứng dụng.

- Kiểm tra các URL, không cho phép người dùng cuối có thể can thiệp vào quá trình redirect từ ứng dụng này sang ứng dụng khác.

- Tạo danh sách các URL tin cậy.

- Tạo trang thông báo xác nhận cho người dùng khi có chuyển hướng (redirect) và chuyển tiếp (forward) đến ứng dụng trên một trang khác.

- Không cho phép người dùng cuối có thể can thiệp vào quá trình redirect từ ứng dụng này sang ứng dụng khác.

# 5. Các lổ hổng bảo mật trên mobile

## 5.1. Insecure Data Storage

### 5.1.1 Khái niệm

Hầu hết các lập trình viên đều cho rằng việc lưu trữ dữ liệu ở client-site sẽ giới hạn không cho các người dùng khác có thể truy cập đến dữ liệu đó. Nhưng hầu hết các lỗ hổng bảo mật ứng dụng mobile hàng đầu đã được thực hiện bởi những dữ liệu không được lưu trữ an toàn tại client-side. Các tập tin hệ thống trên thiết bị không còn như ở môi trường sandbox và việc root hoặc jailbreak sẽ tạo ra lỗ hổng bảo mật.

Các loại dữ liệu không an toàn gồm: Username, Authentication Token hoặc Password, Cookies, Location data, Stored application logs hoặc Debug information, Cached application messages hoặc Transaction history, UDID hoặc EMEI, Personal Information (DoB, Address, Social, …), Device Name, Network Connection Name, private API calls for high user roles, Credit Card Data hoặc Account Data, ...

Những nơi thường lưu trữ dữ liệu không an toàn: SQLite database, Log file, Plist file, XML Data Store hoặc Manifest file, SD Card, Đồng bộ Cloud…

### 5.1.2 Cách phòng chống

Quy định cơ bản cho các ứng dụng mobile là không được lưu trữ dữ liệu trừ khi thật cần thiết.

* Đối với iOS:

- Không lưu thông tin chứng thực trên thiết bị. Đảm bảo người dùng phải xác thực bằng standard web hoặc API đăng nhập (HTTPS) khi kết nối ứng dụng, đảm bảo session timeout phải được thiết lập tối thiểu để đáp ứng yêu cầu của người sử dụng.

- Nơi lưu trữ hoặc caching của thông tin xem xét sử dụng một thư viện mã hóa chuẩn iOS như CommonCrypto. Tuy nhiên, đối với các ứng dụng nhạy cảm (Finance,…) nên xem xét sử dụng các giải pháp mã hóa whitebox cryptography để tránh bị rò rỉ chữ ký nhị phân trong các thư viện mã hóa phổ biến.

- Nếu dữ liệu lưu trữ nhỏ, có thể sử dụng apple keychain API nhưng nếu thiết bị bị jailbreak hoặc bị khai thác thì keychain vẫn có thể dễ dàng đọc được.

- Đối với cơ sở dữ liệu, nên xem xét sử dụng SQLcipher cho việc mã hóa dữ liệu Sqlite.

- Đối với dữ liệu lớn nói chung, dữ liệu của khách hàng, cơ chế Apple’s File Protection được sử dụng một cách an toàn (xem NSData Class Reference)

- Tránh sử dụng NSUserDefaults để lưu trữ các thông tin nhạy cảm vì nó lưu trữ dưới dạng file plist. Nhận biết là tất cả các dữ liệu sử dụng NSManagedObects sẽ được lưu trữ trong một tập tin cơ sở dữ liệu không được mã hóa.

- Tránh sử dụng key mã hóa hoặc giải mã đã được hardcoded khi lưu giữ những thông tin nhạy cảm.

- Xem xét cung cấp thêm một lớp mã hóa hơn bất kỳ cơ chế mã hóa mặc định được cung cấp bởi hệ điều hành.

* Đối với Android:

- Đối với lưu trữ cục bộ, API quản trị thiết bị Android có thể được sử dụng để buộc mã hóa các tập tin lưu trữ nội bộ bằng “setStorageEncryption”.

- Đối với lưu trữ trên SD Card, có thể bảo mật bằng cách sử dụng thư viện “javax.crypto”. Sẽ có một vài lựa chọn, nhưng để đơn giản có thể mã hóa dữ liệu văn bản bằng cách sử dụng mật khẩu và thuật toán AES 128.

- Đảm bảo mọi thuộc tính chia sẻ phải là NOT MODEWORLDREADABLE trừ khi có yêu cầu chia sẻ tường minh giữa các ứng dụng.

- Tránh sử dụng key mã hóa hoặc giải mã đã được hardcoded khi lưu giữ những thông tin nhạy cảm.

- Xem xét cung cấp thêm một lớp mã hóa hơn bất kỳ cơ chế mã hóa mặc định được cung cấp bởi hệ điều hành.

## 5.2. Insufficient Transport Layer

### 5.2.1 Khái niệm

Thông thường, khi một ứng dụng mobile được thiết kế, dữ liệu được trao đổi theo kiểu client-server. Khi dữ liệu này được trao đổi nó có thể đi qua carrier network và internet. Đối với dữ liệu nhạy cảm, nếu ứng dụng được mã hóa kém, các mối đe dọa từ có thể sử dụng các kỹ thuật để xem dữ liệu nhạy cảm này trong khi nó đang được trao đổi. Rõ ràng, bạn sẽ không muốn thông tin nhạy cảm như mật khẩu, số thẻ tín dụng, hoặc các dữ liệu nhạy cảm khác gửi và nhận mà không một số loại mã hóa, thường HTTPS.

### 5.2.2 Cách phòng chống

- Áp dụng SSL/TLS cho các kênh trao đổi thông tin mà ứng dụng mobile sẽ sử dụng để truyền tải các thông tin nhạy cảm, session token, hoặc các dữ liệu nhạy cảm khá đến backend API hoặc webservice.

- Sử dụng thuật toán, mã hóa mạnh với chiều dài khóa thích hợp.

- Không cho phép sử dụng self-signed certificate.

- Luôn yêu cầu xác minh SSL chain.

- Chỉ thiết lập kết nối an toàn sau khi đã xác minh được server sử dụng trusted certificate.

- Cảnh báo cho người dùng qua giao diện trên ứng dụng mobile khi phát hiện invalid certificate.

- Không gửi các dữ liệu nhạy cảm treen các kênh khác (SMS, notification…)

* Đối với iOS:

- Khi sử dụng CFNetwork, xem xét sử dụng Secure Transport API để chỉ định trusted client certificate. Trong hầu hết các trường hợp, NSStreamSocketSecurityLevelTLSv1 nên được sử dụng.

- Sau khi phát triển, đảm bảo mọi NSURL calls (hoặc wrapper of NSURL) không được sử dụng self-signed certificate hoặc invalid sertificate như method setAllowAnyHTTPSCertificate trong NSURL class.

- Xem xét gắn certificate bằng cách: xuất certificate, gắn vào gói ứng dụng và liên kết nó với 1 đối tượng tin cậy. Sử dụng hàm kết nối NSURL: willSendRequestForAuthenticationChallenge.

* Đối với Android:

- Sau quá trình phát triển, xóa các đoạn code có thể cho phép ứng dụng chấp nhận tất cả các certificate như: *org.apache.http.conn.ssl.AllowAllHostnameVerifier* hoặc *SSLSocketFactory.ALLOWALLHOSTNAME\_VERIFIER.*

- Nếu sử dụng một lớp kế thừa SSLSocketFactory, đảm bảo hàm checkServerTrusted được hiện thực để certificate của server được kiểm tra một cách chính xác.

## 5.3. Unintended Data Leakage

### 5.3.1 Khái niệm

Việc rò rỉ dữ liệu ngoài ý muốn xảy rả khi lập trình viên vô tình đặt thông tin nhạy cảm hoặc các dữ liệu trong một nơi trên thiết bị mobile có thể dễ dàng truy cập bởi các ứng dụng khác trên thiết bị. Những nơi không an toàn này có thể được truy cập bởi các ứng dụng độc hại khác chạy trên cùng thiết bị, do đó thiết bị sẽ trong tình trạng nguy cơ nghiêm trọng.

### 5.3.2 Cách phòng chống

Có một số điểm rò rỉ phổ biến có thể lưu ý:

* URL Caching (Request và Response).
* Keyboard Press Caching.
* Copy/Paste buffer Caching.
* Application backgrounding.
* Logging.
* HTML5 data storage.
* Browser cookie objects.
* Analytics data sent to 3rd parties.

## 5.4. Broken Cryptography

### 5.4.1 Khái niệm

Broken Cryptograply để cập đến việc sử dụng mã hóa mật mã không đúng cách cho các ứng dụng dành cho thiết bị di động. Tài nguyên phần cứng dành cho thiết bị di động luôn thấp hơn so với các thiết bị khác như máy trạm, laptop,... Vì vậy các thuật toán mã hóa, cách phương thức được áp dụng để đảm bảo an ninh cho các thiết bị di động luôn yếu hơn so với khi áp dụng cho các thiết bị khác, vì điều này mà lỗ hổng về mã hóa mật mã trên thiết bị di động luôn dễ khai thác hơn.

Ví dụ:

- Sử dụng các thuật toán mã hóa mã nguồn cơ bản.

- Quản lý key không đúng cách.

- Tạo/Sử dụng thuật toán/giao thức cá nhân.

- Sử dụng các thuật toán mã hóa yếu như RC2, MD4, ...

### 5.4.2 Cách phòng chống.

- Sử dụng các thuật toán mã hóa đủ mạnh để đảm bảo an ninh cho các thiết bị di động.

- Cần có chính sách về việc quản lý, lưu trữ và sử dụng key trong ứng dụng.

- Không tự ý thay đổi thuật toán mã hóa, thay vào đó nên sử dụng các thuật toán chuẩn.

## 5.5. Client Side Injection

### 5.5.1 Khái niệm

Client Side Injection đề cập đến kiểu khai thác lỗ hổng bằng cách chèn thêm dữ liệu vào ứng dụng để thay đổi cách thức hoạt động của ứng dụng, làm cho ứng dụng thực thi theo mong muốn của người tấn công. Kiểm tra lại mã nguồn là cách nhanh nhất để chắc rằng ứng dụng xử lý dữ liệu đầu vào một cách đúng đắn, có các công cụ tự động để hỗ trợ kiểm tra mã nguồn một cách tự động. Người kiểm thử có thể xác nhận lại các vấn đề này bằng cách thử khai thác ứng dụng.

Dữ liệu đầu vào của ứng dụng đến từ nhiều nguồn, ví dụ như:

- Dữ liệu trên thiết bị: dữ liệu có thể được lưu trong cơ sở dữ liệu như SQLite hoặc lưu thành các tập tin. Vì vậy ứng dụng có thể khai thác giống như lỗi SQL Injection trên ứng dụng web, hoặc như lỗi Local File Inclusion.

- Session của người dùng: trình duyệt trên thiết bị di động là đối tượng của kiểu tấn công này, việc trình duyệt có thể truy cập đến giá trị cookie của ứng dụng dẫn đến có thể bị đánh cắp giá trị này.

- Giao diện ứng dụng hoặc các chức năng của ứng dụng: nhiều giao diện ứng dụng hoặc chức năng chấp nhận dữ liệu đầu vào và có thể bị lỗi hoặc bị khai thác, tuy chỉ khai thác được vào môi trường người dùng (userland) nhưng từ đây người tấn công có thể khai thác mở rộng thêm.

- Mã nhị phân của ứng dụng: các malware hay các ứng dụng chứa mã độc có thể thực hiện khai thác lên HTML, JavaScript, CSS,... hoặc lên chính mã nhị phân của ứng dụng di động.

### 5.5.2 Cách phòng chống

Một cách tổng quát, để bảo vệ ứng dụng khỏi các cuộc tấn công Client Side Injection cần phải tìm kiếm tất cả các khu vực mà ứng dụng nhận dữ liệu đầu vào và thực hiện xử lý đối với dữ liệu này.

* Đối với iOS:

- SQLite Injection: khi thiết kế câu truy vấn cho SQLite, phải chắc chắn rằng dữ liệu người dùng nhập vào đang được truyền như một tham số truy vấn. Điều này có thể được phát hiện bằng việc kiểm tra định dạng của dữ liệu truyền vào, dữ liệu nguy hiểm sẽ được chèn vào bởi "% @" thay cho "?"

- JavaScript Injection: đảm bảo rằng tất cả UIWebView sẽ không thực thi nếu không được xử lý đối với dữ liệu đầu vào. Áp dụng bộ lọc dữ liệu cho các ký tự Javascript nguy hiểm (nếu có thể), sử dụng whitelist thay cho blacklist các ký tự được cho phép trước khi render.

- Local File Inclusion: xử lý dữ liệu đầu vào đối với NSFileManager.

- XML Injection: sử dụng libXML2 thay cho NSXMLParser.

- Format String Injection: có nhiều phương thức trong Objective C bị lỗi định dạng chuỗi như NSLog, [NSString stringWithFormat:], [NSString initWithFormat:], [NSMutableString appendFormat:], [NSAlert informativeTextWithFormat:], [NSPredicate predicateWithFormat:], [NSException format:], NSRunAlertPanel. Tránh sử dụng các hàm bị lỗi bảo mật bằng cách kiểm tra trước xem hàm cần sử dụng có tồn tại lỗ hổng nào hay không

- Ngoài ra, tránh sử dụng ngôn ngữ C vì đây là ngôn ngữ có tồn tại nhiều hàm bị lỗi như strcat, strcpy, strncat, strncpy, sprint, vsprintf, gets, ...

* Đối với Android:

- SQL Injection: khi xử lý các câu truy vấn động hoặc Content-Providers, đảm bảo rằng dữ liệu truyền vào dưới dạng tham số hóa.

- JavaScript Injection: đảm bảo rằng JavaScript và các plugin hỗ trợ đã được tắt đối với bất kỳ WebViews (thường là mặc định).

- Local File Inclusion: đảm bảo rằng File System Access đã được tắt đối với bất kỳ WebViews (webview.getSettings().setAllowFileAccess(false);).

- Intent Injection/Fuzzing: đảm bảo rằng các hành động và dữ liệu đầu vào được xử lý thông qua Intent Filter đối với tất cả Activities.

## 5.6. Security Decisions Via Untrusted Inputs

### 5.6.1 Khái niệm

Ứng dụng di động có thể chấp nhận dữ liệu từ tất cả các nguồn, trong phần lớn trường hợp là cơ chế Inter Process Communication (IPC). Thông thường, cần phải có tài liệu mô tả thiết kế cho IPC như sau:

- Có yêu cầu đặc biệt nào từ bên kinh doanh cho giao tiếp IPC, yêu cầu đặc biệt phải sử dụng các ứng dụng từ các nguồn white-list ngoài.

- Các hành động nhạy cảm phải được thông qua IPC và yêu cầu xác nhận của người dùng trước khi thực thi.

- Tất cả dữ liệu nhận được từ IPC phải được xử lý trước khi tiếp tục thực thi.

- Không gửi bất kỳ thông tin nhạy cảm qua cơ cế IPC vì dữ liệu này có thể được đọc bởi một bên thứ ba.

### 5.6.2 Cách phòng chống

Để tránh lỗ hổng Security Decisions Via Untrusted Inputs, không nên sử dụng phương thức handleOpenURL để xử lý URL Scheme vì phương thức này không an toàn, ngoài ra không được dùng iOS Pasteboard cho giao tiếp IPC vì ứng dụng thứ ba có thể đọc được dữ liệu này trên thiết bị.

## 5.7. Improper Session Handling

### 5.7.1 Khái niệm

Vấn đề này là do session token không được xử lý một cách tốt nhất. Các lập trình viên cho phép session của người dùng quá dài hoặc non-expire, hoặc session token quá dễ đoán được. Để tạo điều kiện một stateful transaction giữa người dùng và backend server, ứng dụng mobile sử dụng session token để duy trì trạng thái thông qua các giao thức như HTTP hoặc SOAP. Để duy trì trạng thái, ứng dụng mobile phải chứng thứng thông tin người dùng trước thông qua backend. Khi xác thực thành công, server cấp 1 session cookie cho ứng dụng mobile. Ứng dụng mobile thêm cookie này vào tất cả các giao dịch trong tương lai giữa ứng dụng mobile và server. Nó cho phép server thuận tiện thực thi xác thực và ủy quyền cho bất cứ yêu cầu dịch vụ từ ứng dụng mobile. Xử lý session không đúng cách xảy ra khi session token được vô tình chia sẻ với người khác trong một giao dịch sau này giữa ứng dụng mobile và backend server.

### 5.7.2 Cách phòng chống

Để quản lý các session đúng cách, đảm bảo rằng mã ứng dụng di động tạo ra, duy trì, và hủy session token đúng theo vòng đời của một session ứng dụng mobile của người dùng. Một số lời khuyên như sau:

* 15 phút cho high-security applications
* 30 phút for med-security applications
* 1 tiếng cho low-security applications

## 5.8. Lack of Binary Protections

### 5.8.1 Khái niệm

Một môi trường được xem là không đáng tin cậy khi mà môi trường đó chủ sở hữu không có quyền kiểm soát vật lý, bao gồm các thiết bị di động, không gian trên internet, trung tâm dữ liệu,... Đối với kiểu tấn công Binary Attack trong môi trường không đáng tin cậy, cần phải xem xét các yếu tố:

- Người nào có thể giải mã mã nguồn của ứng dụng, ví dụ như trên iPhone thì sử dụng công cụ tự động ClutchMod hoặc giải mã bằng tay bằng công cụ GDB.

- Người nào có thể dịch ngược ứng dụng, ví dụ như trên Android thì sử dụng công cụ dex2jar.

- Người nào có thể sử dụng công cụ tự động như Hopper hay IDA Pro để giả lập môi trường chạy ứng dụng.

- Người nào có thể thay đổi lớp hiển thị của ứng dụng (HTML/JS/CSS) của ứng dụng trong thiết bị di động và thực thi ứng dụng đã bị thay đổi.

- Người nào có thể sử dụng thay đổi tập tin thực thi của ứng dụng bằng công cụ như hex editor để vượt qua kiểm soát an ninh.

### 5.8.2 Cách phòng chống

Để chống lại kiểu tấn công Binary Attack, ứng dụng cần phải sử dụng các kỹ thuật bảo vệ cho các ứng dụng di động, như:

- Jailbreak Detection Controls.

- Checksum Controls.

- Certificate Pinning Controls.

- Debugger Detection Controls.

Đối với các ứng dụng Android, cần áp dụng kỹ thuật Android Root Detection

- Kiểm tra test-keys: kiểm tra build.prop có tồn tại 'ro.build.tags=test-keys' hay không?

- Kiểm tra chứng chỉ OTA: kiểm tra tập tin /etc/security/otacerts.zip có tồn tại hay không?

- Kiểm tra các apk thuộc root: com.noshufou.android.su, com.thirdparty.superuser, eu.chainfire.supersu, com.koushikdutta.superuser

- Kiểm tra tập tin SU: /system/bin/su, /system/xbin/su, /sbin/su, /system/su, /system/bin/.ext/.su

# 6. Tham khảo

Tài liệu hướng dẫn lập trình an toàn của các hãng:

* Apple: Introduction to Secure Coding Guide

<https://developer.apple.com/library/mac/documentation/Security/Conceptual/SecureCodingGuide/Introduction.html>

* Oracle: Secure Coding Guidelines for Java SE

<http://www.oracle.com/technetwork/java/seccodeguide-139067.html>

* Microsoft: Writing Secure Code

<https://msdn.microsoft.com/en-us/security/aa570401.aspx>

* Dự án OWASP (Open Web Application Security Project)

<https://www.owasp.org/>

<https://www.owasp.org/index.php/Top_10_2013-Top_10>

<https://www.owasp.org/index.php/Projects/OWASP_Mobile_Security_Project_-_Top_Ten_Mobile_Risks>

<https://blog.appknox.com/10-measures-to-meet-owasp-security-guidelines-for-your-mobile-app/>