Part I Secure Software Concepts

Chapter 1 Core Concept

Confidentiality

Dưới đây là nội dung chi tiết về phần **Tính bảo mật (Confidentiality)** dựa trên chương đầu tiên của tài liệu:

1. Định nghĩa và Khái niệm cốt lõi

- **Định nghĩa:** Tính bảo mật là khái niệm ngăn chặn việc tiết lộ thông tin cho các bên không được ủy quyền. Cốt lõi của nó là việc **"giữ bí mật những điều cần bí mật"**.
- **Mối quan hệ với Ủy quyền:** Việc đạt được tính bảo mật phụ thuộc vào khả năng xác định ai là bên được ủy quyền, do đó nó liên quan chặt chẽ đến khái niệm **Ủy quyền (Authorization)**.
- **Hậu quả khi thất bại:** Sự mất mát tính bảo mật chính là việc tiết lộ thông tin trái phép.

2. Các phương pháp bảo vệ dữ liệu

Kỹ thuật được sử dụng tùy thuộc vào trạng thái của dữ liệu:

- **Dữ liệu đang được sử dụng (in use) và Dữ liệu lưu trữ (at rest):** Thường ưu tiên sử dụng các biện pháp **Kiểm soát truy cập (Access controls)**.
- **Dữ liệu đang truyền đi (in transit) và Dữ liệu lưu trữ (at rest):** Thường sử dụng phương pháp **Mã hóa (Encryption)**.

3. Xác định yêu cầu bảo mật trong chính sách

Chính sách bảo mật sẽ định nghĩa các yêu cầu dựa trên việc **ai có thể trao đổi thông tin gì giữa các điểm cuối nào**. Các yếu tố then chốt cần xác định bao gồm:

- Ai được phép xem các phần tử dữ liệu cụ thể?
- Cơ chế nào nên được sử dụng để thực thi tính bảo mật?

- Các yêu cầu kinh doanh liên quan đến việc thu thập, chuyển giao, lưu trữ và sử dụng dữ liệu là gì?

4. Triển khai Tính bảo mật

- **Cơ chế chính:** Sử dụng mã hóa để làm cho dữ liệu không thể đọc được bằng cách xáo trộn các giá trị thực.
- **Yếu tố kỹ thuật:** Mã hóa dựa trên hai yếu tố chính là **Thuật toán (Algorithm)** và **Khóa (Key)**. Người sở hữu khóa đúng mới có thể truy cập được dữ liệu.
- **Nguyên tắc triển khai an toàn:**
    - **Không tự tạo mã hóa:** Luôn sử dụng các thư viện và hàm mã hóa đã được phê duyệt. Việc tự xây dựng thuật toán mã hóa (rolling your own crypto) hầu như luôn dẫn đến thất bại.
    - **Bảo vệ khóa:** Thất bại trong việc bảo vệ khóa sẽ dẫn đến thất bại toàn bộ hệ thống bảo mật. Tuyệt đối không lưu trữ khóa trong mã nguồn (code) hoặc trong các tệp tin dễ bị lấy trộm.
    - **Tính công khai của thuật toán:** Các quy trình mật mã hiện đại dựa trên nền tảng là sự triển khai công khai các thuật toán đã được phê duyệt, kết hợp với các bước bảo mật khóa.

Integrity

Dựa trên nội dung Chương 1 của tài liệu, **Tính toàn vẹn (Integrity)** được trình bày chi tiết với các khía cạnh sau:

1. Định nghĩa và Ý nghĩa

- **Định nghĩa:** Tính toàn vẹn đề cập đến việc bảo vệ dữ liệu khỏi các **thay đổi trái phép (unauthorized alteration)**. Theo tiêu chuẩn FIPS 199, việc mất tính toàn vẹn chính là sự sửa đổi hoặc hủy hoại thông tin trái phép.
- **Tầm quan trọng:** Việc kiểm soát các thay đổi (bao gồm cả việc xóa dữ liệu) là yếu tố thiết yếu để đảm bảo **tính ổn định và độ tin cậy** của hệ thống.

- **Mối quan hệ:** Tính toàn vẹn đóng vai trò quan trọng trong việc xác định **tính xác thực (authenticity)** và đảm bảo **chống chối bỏ (non-repudiation)**.

2. Các yêu cầu về Tính toàn vẹn (Integrity Requirements)

Để thiết lập chính sách bảo mật, các yêu cầu về tính toàn vẹn cần được phân rã dựa trên các yếu tố:

- Xác định **ai được ủy quyền** để thay đổi các phần tử thông tin hoặc luồng dữ liệu cụ thể.
- Xác định **cơ chế nào** nên được sử dụng để phát hiện lỗi và thực thi tính toàn vẹn.
- Xác định các **yêu cầu kinh doanh** đối với việc thu thập, chuyển giao, lưu trữ và sử dụng dữ liệu liên quan đến tính toàn vẹn.
- Cần xem xét các yêu cầu làm cơ sở để **theo dõi các điều kiện lỗi**.

3. Triển khai Tính toàn vẹn (Implementing Integrity)

Tính toàn vẹn yêu cầu các biện pháp kiểm soát chi tiết hơn so với chỉ cấp quyền truy cập thông thường. Các phương pháp chính bao gồm:

- **Cơ chế Kiểm soát Truy cập (Access Control):** Định nghĩa các mức độ truy cập cụ thể như đọc (read), ghi (write) và xóa (delete) cho từng đối tượng.
- **Hàm băm (Hashing):**
    - Được sử dụng để xác minh tính toàn vẹn của dữ liệu trong suốt vòng đời của nó.
    - Nguyên lý: Chạy dữ liệu qua hàm băm để tạo ra một giá trị đại diện (digest). Nếu digest hiện tại khớp với digest ban đầu, dữ liệu được coi là chưa bị thay đổi.
    - Digest có kích thước cố định bất kể dữ liệu đầu vào lớn hay nhỏ.
- **Chữ ký số và Ký số mã nguồn (Digital Signatures & Code-signing):**
    - Đảm bảo một đối tượng không bị sửa đổi trái phép kể từ khi được ký.

    - Việc bổ sung danh tính của bên ký giúp người dùng xác định được **tính xác thực**.
- **Thiết kế hệ thống:** Các quyết định về thời điểm tính toán mã băm, cách lưu trữ và truyền tải chúng qua các kênh riêng biệt cần được thực hiện ngay từ giai đoạn thiết kế để giai đoạn lập trình có thể triển khai đúng cách.

Availability

Dựa trên nội dung Chương 1 của tài liệu, **Tính sẵn sàng (Availability)** được trình bày chi tiết như sau:

1. Định nghĩa và Mục tiêu cốt lõi

- **Định nghĩa cơ bản:** Tính sẵn sàng liên quan đến việc đảm bảo hệ thống có thể được truy cập bởi những người có thẩm quyền khi họ cần.
- **Hai mục tiêu chính:**
    1. Đảm bảo hệ thống luôn sẵn sàng cho người dùng hợp lệ khi họ có nhu cầu sử dụng.
    2. Từ chối quyền truy cập của những người dùng không có thẩm quyền vào mọi thời điểm khác.
- **Định nghĩa theo FIPS 199:** Là việc "đảm bảo quyền truy cập và sử dụng thông tin một cách kịp thời và đáng tin cậy".

2. Xác định giá trị của Tính sẵn sàng

- **Phụ thuộc vào mức độ quan trọng:** Giá trị của tính sẵn sàng được quyết định bởi mức độ quan trọng (criticality) của dữ liệu và mục đích sử dụng của nó trong hệ thống.
- **Ví dụ so sánh:**
    - Với các hệ thống như email hoặc duyệt web, các vấn đề gián đoạn tạm thời có thể không phải là vấn đề nghiêm trọng.

- Với các hệ thống điều khiển các nhà máy công nghiệp lớn (như nhà máy lọc dầu), tính sẵn sàng có thể là **thuộc tính quan trọng nhất**.

3. Thất bại về Tính sẵn sàng (Availability Failure)

- **Khái niệm mất tính sẵn sàng:** Theo FIPS 199, đây là sự gián đoạn trong việc truy cập hoặc sử dụng thông tin hoặc một hệ thống thông tin.
- **Lưu ý quan trọng:** Nếu một hệ thống ngăn cản người dùng xem một tài liệu mà họ **có quyền** được xem, đó được coi là một thất bại an ninh, cụ thể là thất bại về tính sẵn sàng. Việc gọi đây là biện pháp "an ninh" là sai lầm vì nó có thể dẫn đến các quyết định sai do thiếu dữ liệu.

4. Thách thức trong Thiết kế và Triển khai

- **Xác định mức độ phù hợp:** Thách thức trong việc định nghĩa và thiết kế hệ thống là xác định mức độ sẵn sàng chính xác cho từng thành phần dữ liệu.
- **Ngăn chặn tấn công:** Các vấn đề phổ biến cần giải quyết bao gồm ngăn chặn các truy cập bất hợp pháp và các cuộc tấn công từ chối dịch vụ (DoS).
- **Sự cân bằng CIA:** Một trong những thách thức lớn trong giai đoạn yêu cầu và thiết kế là phân loại và xác định sự **cân bằng phù hợp** giữa ba yếu tố Confidentiality, Integrity và Availability. Mặc dù chúng khác nhau nhưng không nhất thiết phải mâu thuẫn; tuy nhiên, tất cả đều tiêu tốn tài nguyên hệ thống.

Authentication

Dựa trên nội dung Chương 1 của cuốn sách **"CSSLP® Certification All-in-One Exam Guide, Third Edition"**, dưới đây là chi tiết về phần **Authentication (Xác thực)**:

1. Định nghĩa và Vai trò

- **Xác thực** là quy trình xác định danh tính của một người dùng.
- Đây là yếu tố nền tảng của an ninh vì nó cung cấp phương tiện để phân tách người dùng, cho phép phân biệt giữa người dùng được ủy quyền và người dùng không được ủy quyền.

- **Phân biệt với Ủy quyền (Authorization):** Trong khi xác thực trả lời câu hỏi "Bạn là ai?", ủy quyền xác định các quy tắc về những gì cá nhân đó có thể làm trên hệ thống sau khi đã được xác thực.

2. Các yếu tố xác thực (Authentication Factors)

Người dùng có thể xác minh danh tính thông qua ba phương pháp (yếu tố) truyền thống:

- **Thứ bạn biết (Something you know):** Phổ biến nhất là tên người dùng và mật khẩu (hoặc mã PIN).
- **Thứ bạn có (Something you have):** Các vật hữu hình như khóa vật lý, thẻ thông minh hoặc các mã thông báo (tokens) chứa yếu tố mật mã.
- **Thứ thuộc về bạn (Something you are):** Sử dụng các đặc điểm sinh trắc học tĩnh như dấu vân tay, quét võng mạc hoặc hình dạng bàn tay.

Các danh mục mới phát triển nhờ công nghệ:

- **Thứ bạn làm (What you do):** Sinh trắc học động như kiểu gõ bàn phím hoặc dáng đi.
- **Nơi bạn ở (Where a user is):** Vị trí vật lý thực tế của người dùng.

3. Xác thực đa yếu tố (Multifactor Authentication - MFA)

- **Khái niệm:** Là sự kết hợp của hai hoặc nhiều loại xác thực khác nhau.
- **Xác thực hai yếu tố (2FA):** Kết hợp bất kỳ hai yếu tố nào (ví dụ: quẹt thẻ và sau đó quét vân tay).
- **Xác thực ba yếu tố:** Kết hợp ba loại, ví dụ: thẻ thông minh (thứ bạn có), mã PIN (thứ bạn biết) và quét võng mạc (thứ thuộc về bạn).
- **Lợi ích:** MFA tăng cường an ninh đáng kể bằng cách khiến kẻ tấn công khó lấy được tất cả các tài liệu cần thiết để xác thực và giúp bảo vệ chống lại rủi ro mất cắp mã thông báo hoặc sinh trắc học.

4. Quản lý định danh và quyền truy cập (IDM/IAM)

- **Identity Management (IDM):** Là bộ dịch vụ toàn diện liên quan đến việc quản lý sử dụng các định danh như một phần của giải pháp kiểm soát truy cập.
- **Quy trình:** Bao gồm các bước **Cấp phát (Provisioning)** - tạo định danh kỹ thuật số từ định danh thực tế; quản lý; và **Thu hồi (Deprovisioning)** định danh.
- **Tầm quan trọng:** Trong các hệ thống lớn, việc quản lý định danh cần được tự động hóa để đảm bảo các thay đổi về vai trò, quyền hạn và đặc quyền được thực hiện an toàn và được ghi nhật ký đầy đủ. Các kiểm soát IAM thường được kiểm toán hàng năm theo các quy định như **Sarbanes-Oxley (SOX)**.

5. Nhà cung cấp định danh (Identity Provider - IdP)

- **IdP** là hệ thống hoặc dịch vụ tạo ra, duy trì và quản lý thông tin định danh.
- IdP chịu trách nhiệm xác thực định danh và có thể hoạt động ở quy mô một hệ thống đơn lẻ hoặc trên toàn doanh nghiệp.
- Các tiêu chuẩn phổ biến cho dịch vụ này bao gồm **SAML, OpenID và OAuth**.

6. Chứng chỉ số và Tiêu chuẩn X.509

- **X.509** là một loạt các tiêu chuẩn liên quan đến việc xử lý chứng chỉ để chuyển giao các khóa bất đối xứng giữa các bên một cách có thể xác minh được.
- **Chứng chỉ kỹ thuật số:** Ràng buộc danh tính cá nhân với một khóa công khai (public key). Nó được tạo ra bởi **Cơ quan cấp chứng chỉ (CA)** sau khi **Cơ quan đăng ký (RA)** xác minh danh tính cá nhân.
- **Các trường thông tin trong chứng chỉ X.509:** Bao gồm số phiên bản, số sê-ri, thuật toán chữ ký, tên bên cấp (Issuer), thời hạn hiệu lực, chủ thể (Subject), và khóa công khai.

7. Hệ thống định danh liên hợp và SSO

- **Hệ thống định danh liên hợp (Federated ID):** Cho phép người dùng kết nối với các hệ thống thông qua các hệ thống đã biết (ví dụ: dùng tài khoản Facebook để đăng

nhập vào trang web khác). Có hai bên chính: **Bên tin cậy (Relying Party - RP)** và **Nhà cung cấp định danh (IdP)**.

  - **OpenID:** Được tạo ra để xác thực liên hợp, khẳng định "đây là tôi là ai".
  - **OAuth:** Được tạo ra để cho phép truy cập tài nguyên (ủy quyền) mà không cần chia sẻ mật khẩu với bên thứ ba thông qua mã thông báo truy cập (access token).

- **Đăng nhập một lần (Single Sign-On - SSO):** Giúp người dùng có thể sử dụng lại thông tin xác thực trên nhiều ứng dụng khác nhau mà không cần nhập lại mật khẩu. Tuy nhiên, SSO có thể tạo ra kịch bản **"điểm yếu duy nhất" (single-point-of-failure)** nên không được khuyến nghị cho các triển khai có rủi ro cao.

8. Quản lý thông tin xác thực (Credential Management)

- Thông tin định danh (credentials) là dữ liệu nhạy cảm, có thể dưới dạng mật khẩu, chuỗi kỹ thuật số trong mã thông báo phần cứng, sinh trắc học hoặc chứng chỉ.
- Việc quản lý bao gồm: tạo, lưu trữ, đồng bộ hóa, đặt lại và thu hồi thông tin xác thực. Tất cả các hoạt động này phải được ghi lại (log).
- **Nguyên tắc bảo mật:** Mật khẩu và các thông tin xác minh khác không bao giờ được phép truy cập bởi bất kỳ ai, kể cả quản trị viên hệ thống. Nếu hệ thống có thể gửi email mật khẩu cho bạn, điều đó có nghĩa là nó không được lưu trữ đúng cách (phải sử dụng mật mã để không thể tiết lộ).

Authorization

Dựa trên nội dung Chương 1 của cuốn sách **"CSSLP® Certification All-in-One Exam Guide, Third Edition"**, dưới đây là chi tiết về phần **Authorization (Ủy quyền)**:

1. Định nghĩa và Khái niệm cốt lõi

- **Định nghĩa:** Ủy quyền là quá trình áp dụng các quy tắc kiểm soát truy cập đối với một tiến trình của người dùng để xác định xem người dùng đó có thể truy cập vào một đối tượng cụ thể hay không.

- **Mối quan hệ với Xác thực (Authentication):** Hai khái niệm này thường đi đôi với nhau nhưng hoàn toàn khác biệt. Trong khi **Xác thực** tập trung vào việc xác minh danh tính ("Bạn là ai?"), **Ủy quyền** tập trung vào các quy tắc xác định những gì cá nhân đó có thể làm trên hệ thống sau khi đã được xác thực ("Bạn được làm gì?").

2. Ba yếu tố chính trong Ủy quyền

Quá trình ủy quyền xoay quanh ba yếu tố cơ bản:

1. **Chủ thể (Subject/Requestor):** Là người dùng đã được hệ thống xác thực và đang yêu cầu quyền truy cập.
2. **Đối tượng (Object):** Là tài nguyên mà người dùng muốn tương tác, ví dụ như tệp tin (file), chương trình, mục nhập cơ sở dữ liệu, hoặc trang web.
3. **Loại/Mức độ truy cập (Type/Level of Access):** Các hành động cụ thể được yêu cầu, phổ biến nhất là: **đọc (read), ghi (write), tạo (create), xóa (delete)**, hoặc quyền cấp quyền truy cập cho các chủ thể khác.

3. Mô hình Chủ thể-Đối tượng-Hoạt động (Subject-Object-Activity Model)

Đây được coi là phương pháp tốt nhất để phân rã quy trình ủy quyền:

- **Subject (Chủ thể):** Người dùng đã được xác định danh tính thành công.
- **Object (Đối tượng):** Bất kỳ tài nguyên nào—từ tệp tin hệ thống đến bản ghi cơ sở dữ liệu—mà chủ thể muốn thực hiện hành động lên đó.
- **Activity (Hoạt động):** Hành động mong muốn mà chủ thể muốn gọi đối với đối tượng (ví dụ: thực thi một chương trình hoặc chỉnh sửa một mục dữ liệu).

4. Cơ chế Kiểm soát Truy cập (Access Control Mechanisms)

- **Vai trò:** Hệ thống ủy quyền thực thi việc kiểm soát truy cập thông qua các cơ chế này.
- **Nguyên tắc:** Việc một người được cấp quyền vào hệ thống không có nghĩa là họ có quyền truy cập vào tất cả dữ liệu có trong hệ thống đó. Kiểm soát truy cập điều chỉnh các hoạt động thực tế mà cá nhân có thể thực hiện.

- **Các mô hình phổ biến:** Tài liệu đề cập sơ lược (và sẽ chi tiết ở Chương 2) về các mô hình như:
    - Kiểm soát truy cập bắt buộc (**MAC**).
    - Kiểm soát truy cập tùy quyền (**DAC**).
    - Kiểm soát truy cập dựa trên vai trò hoặc quy tắc (**RBAC**).

5. Sự liên kết với bộ ba CIA

- Hệ thống ủy quyền chính là nơi thực thi các yêu cầu về **Tính bảo mật (Confidentiality), Tính toàn vẹn (Integrity)** và **Tính sẵn sàng (Availability)**.
- Dù là thông qua một cơ chế kiểm soát truy cập đơn giản hay các hệ thống phức tạp hơn như lưới truy cập cơ sở dữ liệu (database access lattice), mục tiêu cuối cùng của ủy quyền vẫn là hiện thực hóa việc kiểm soát quyền truy cập để bảo vệ dữ liệu.

Accountability

Dựa trên nội dung Chương 1 của cuốn sách **"CSSLP® Certification All-in-One Exam Guide, Third Edition"**, dưới đây là chi tiết về phần **Accountability (Trách nhiệm giải trình)**:

1. Định nghĩa và Khái niệm cốt lõi

- **Accounting (Kế toán/Đo lường):** Là phương tiện để đo lường các hoạt động trong hệ thống.
- **Accountability (Trách nhiệm giải trình):** Là việc **ghi lại các hành động và những người thực hiện chúng**.
- **Cơ chế thực hiện:** Trong các hệ thống CNTT, điều này được thực hiện bằng cách ghi nhật ký (**logging**) các yếu tố quan trọng của hoạt động ngay khi chúng xảy ra.
- **Mục đích:** Việc theo dõi các hoạt động này là cần thiết để phục vụ cho việc kiểm toán (**auditing**) vào một thời điểm sau đó.

2. Kiểm toán (Auditing)

- **Vai trò của Quản lý:** Ban quản lý có trách nhiệm đảm bảo các quy trình làm việc diễn ra đúng thiết kế. Kiểm toán chính là "lăng kính" giúp quản lý quan sát hoạt động của hệ thống một cách khách quan và không thiên vị.
- **Xác minh thực tế:** Kiểm toán là quá trình xác minh những gì **thực sự đã xảy ra** trên hệ thống. Nó có thể thực hiện ở nhiều cấp độ, từ phân tích nhật ký hệ thống đến xác minh sự tồn tại và vận hành của các kiểm soát an ninh cụ thể.
- **Nguyên tắc thiết lập:** Việc thiết lập chức năng kiểm toán phải được thực hiện **trước khi sự cố xảy ra**, vì không thể ghi lại hành động của hệ thống sau khi sự việc đã kết thúc.
- **Mức độ kiểm toán:** Việc kiểm toán tiêu tốn tài nguyên hệ thống, vì vậy người vận hành cần xác định mức độ kiểm toán phù hợp dựa trên **tính quan trọng của thông tin** (criticality) được xử lý hoặc lưu trữ trong hệ thống đó.
- **Yêu cầu cơ bản:** Theo quy tắc chung, tất cả các giao dịch quan trọng phải được ghi nhật ký, bao gồm **thời điểm xảy ra** và **người dùng được ủy quyền** liên quan đến sự kiện đó.

3. Ghi nhật ký (Logging)

- **Giá trị của Nhật ký:** Nhật ký cho phép nhân viên kiểm tra thông tin từ nhiều nguồn khác nhau sau khi sự việc đã xảy ra, cung cấp chi tiết về: hành động nào đã diễn ra, tài khoản nào thực hiện, trên máy chủ nào và kết quả cụ thể ra sao.
- **Tuân thủ (Compliance):** Nhiều chương trình tuân thủ như **HIPAA, SOX, PCI DSS** yêu cầu phải có các cơ chế ghi nhật ký và quản lý nhật ký cụ thể.
- **Phục vụ phản ứng sự cố:** Nhật ký cần chứa những thông tin mà các điều tra viên cần để nghiên cứu nguyên nhân hoặc tác động của các thất bại và sự cố.
- **Hệ thống SIEM:** Nhật ký riêng lẻ thường không có nhiều giá trị; sức mạnh thực sự đến từ việc kết hợp các sự kiện trên nhiều nhật ký khác nhau để mô tả chi tiết hoạt động của một người dùng. Các công cụ **Quản lý sự kiện và thông tin an ninh**

(SIEM) cung cấp môi trường phân tích phong phú để tìm ra mối liên hệ giữa các tập dữ liệu lớn.

4. Quyết định Ghi nhật ký (To Log or Not to Log)

- **Lợi ích quản trị:** Ngay cả khi mọi thứ hoạt động bình thường, vẫn cần ghi nhật ký để theo dõi về mặt quản trị và cung cấp hồ sơ lịch sử về các giao dịch.
- **Hỗ trợ gỡ lỗi (Debugging):** Nhật ký tạo ra trong quá trình xử lý lỗi cung cấp cái nhìn sâu sắc về trạng thái hệ thống tại thời điểm xảy ra lỗi, giúp lập trình viên khắc phục vấn đề.
- **Tính kinh tế:** Vì mỗi mục nhật ký đều tiêu tốn thời gian và không gian lưu trữ, cần đưa ra quyết định kinh tế về việc liệu một dữ liệu có xứng đáng để ghi nhật ký hay không.
- **Bảo mật cho Nhật ký:** Các chi tiết trong nhật ký (như ID người dùng, giá trị biến quan trọng, đôi khi là cả mật khẩu nếu sơ suất) có thể bị lạm dụng nếu rơi vào tay kẻ xấu. Do đó, **bản thân các tệp nhật ký cũng cần được bảo vệ nghiêm ngặt**.

5. Rủi ro liên quan đến Kiểm toán

Khi phân rã chính sách an ninh cho các hoạt động kiểm toán, cần xem xét ba hình thức rủi ro liên quan:

- **Rủi ro tiềm tàng (Inherent risk):** Rủi ro liên quan đến quy trình và tỷ lệ lỗi vốn có của nó khi chưa có các kiểm soát nội bộ.
- **Rủi ro phát hiện (Detection risk):** Rủi ro mà việc kiểm toán không phát hiện được một vấn đề có thể dẫn đến sai sót nghiêm trọng.
- **Rủi ro kiểm soát (Control risk):** Rủi ro mà các kiểm soát hiện có không phát hiện hoặc ngăn chặn được các sai sót kịp thời.

Dựa trên nội dung Chương 1 của cuốn sách, dưới đây là chi tiết về phần **Nonrepudiation (Chống chối bỏ)**:

- **Định nghĩa:** Chống chối bỏ là khái niệm **ngăn chặn một chủ thể (subject) phủ nhận một hành động đã thực hiện trước đó** đối với một đối tượng (object) trong hệ thống,.
- **Cơ chế thực hiện:** Khả năng chống chối bỏ không hoạt động đơn lẻ mà được đảm bảo khi ba yếu tố sau được cấu hình đúng cách,:
    - **Xác thực (Authentication):** Xác minh danh tính người thực hiện.
    - **Ủy quyền (Authorization):** Xác minh quyền hạn của người đó.
    - **Kiểm toán (Auditing):** Ghi lại bằng chứng về hành động.
- **Mục đích cốt lõi:** Thiết lập một hệ thống có khả năng **chứng minh một sự kiện có thực sự diễn ra hay không**, từ đó ngăn người dùng phủ nhận trách nhiệm về hành vi của họ.
- **Xác định phạm vi yêu cầu:** Vì chống chối bỏ là một khái niệm rộng, các yêu cầu an ninh phần mềm cần phải quy định cụ thể **chủ thể, đối tượng và sự kiện nào** cần áp dụng tính năng này. Việc xác định rõ ràng phạm vi là rất quan trọng vì nó quyết định mức độ chi tiết của việc ghi nhật ký kiểm toán.
- **Thách thức về dữ liệu:** Nếu tổ chức yêu cầu tính chống chối bỏ tuyệt đối (ghi lại mọi hành động của mọi chủ thể lên mọi đối tượng), hệ thống sẽ phải tạo ra và lưu trữ một **tập dữ liệu nhật ký khổng lồ**. Do đó, các nhà thiết kế cần cân nhắc kỹ lưỡng giữa nhu cầu bảo mật và khả năng đáp ứng tài nguyên hệ thống.

Secure Development Lifecycle

Dựa trên nội dung Chương 1 của cuốn sách, dưới đây là chi tiết về **Vòng đời Phát triển Bảo mật (Secure Development Lifecycle - SDL)**:

1. Định nghĩa và Mục tiêu

- **Khái niệm:** SDL là thuật ngữ dùng để chỉ việc **thêm các yếu tố bảo mật vào vòng đời phát triển phần mềm** nhằm giảm thiểu số lượng lỗi bảo mật trong sản phẩm cuối cùng.

- **Cách tiếp cận:** Để tạo ra một mô hình SDL, các tổ chức chỉ cần thêm một loạt các **kiểm tra quy trình (process checks)** vào mô hình phát triển hiện tại của họ. Các thành phần cốt lõi của SDL là giống nhau, dù phương pháp triển khai có thể thay đổi tùy theo quy trình gốc của đơn vị phát triển.

2. Các mối quan hệ quan trọng

- **Bảo mật vs. Chất lượng (Security vs. Quality):** Chất lượng phần mềm thường được xem là "sự vắng mặt của các khuyết tật". Một phần mềm có chất lượng cao không đồng nghĩa với việc nó bảo mật, nhưng một phần mềm chất lượng kém chắc chắn sẽ không bảo mật. Mục tiêu của SDL là đạt được **cả chất lượng và bảo mật**.
- **Tính năng bảo mật != Phần mềm bảo mật:** Đây là một quan niệm sai lầm phổ biến.
    - **Tính năng bảo mật:** Là các yếu tố như mã hóa hoặc xác thực được thêm vào để cải thiện khả năng sử dụng hoặc tính năng của chương trình.
    - **Phần mềm bảo mật:** Là việc đảm bảo **tất cả các thành phần** của phần mềm được xây dựng để vận hành an toàn, tức là chúng thực hiện đúng những gì được thiết kế và **chỉ thực hiện những gì được thiết kế**.

3. Các thành phần chính của SDL

SDL bao gồm các thành phần chung giúp hiện thực hóa các nguyên tắc thiết kế bảo mật:

- **Nâng cao nhận thức và Giáo dục đội ngũ (Awareness and Education):** Tất cả các thành viên trong nhóm phát triển cần được đào tạo định kỳ và phù hợp với vai trò, trách nhiệm của mình. Đào tạo bao gồm kiến thức cơ bản (dành cho tất cả mọi người) và các chủ đề nâng cao (dành cho các vai trò cụ thể như thiết kế, lập trình, kiểm thử).
- **Các Cổng an ninh và Yêu cầu bảo mật (Gates and Security Requirements):** Trong SDL, các kỳ đánh giá định kỳ được gọi là "cổng". Để vượt qua một cổng an ninh, dự án phải đáp ứng các yêu cầu bảo mật tương ứng; nếu thiếu hụt, dự án sẽ không được chuyển sang giai đoạn phát triển tiếp theo.

- **Theo dõi lỗi (Bug Tracking):** Lỗi là những vấn đề trong mã nguồn dẫn đến hành vi không mong muốn, và một số lỗi có thể bị khai thác để tấn công. Việc theo dõi lỗi giúp xác định trạng thái và ưu tiên khắc phục các vấn đề dựa trên tính kinh tế và mức độ rủi ro.
- **Mô hình hóa mối đe dọa (Threat Modeling):** Một kỹ thuật thiết kế dùng để xác định và mô tả đầy đủ các mối đe dọa đối với hệ thống, đồng thời ghi lại cách giảm thiểu chúng. Hoạt động này bắt đầu từ đầu dự án và kéo dài xuyên suốt quá trình phát triển để đảm bảo cả nhóm có chung hiểu biết về các rủi ro.
- **Đánh giá an ninh (Security Reviews):** Đây là các sự kiện **"mini-audit"** nhằm kiểm tra xem các bước liên quan đến bảo mật trong quy trình SDL có thực sự được thực hiện hay bị bỏ qua để đẩy nhanh tiến độ.
- **Giảm thiểu rủi ro (Mitigations):** Khi phát hiện lỗi bảo mật, có 4 kỹ thuật giảm thiểu tiêu chuẩn:
    1. **Không làm gì (Do nothing):** Thừa nhận rủi ro nếu lỗi nằm dưới ngưỡng nghiêm trọng (bug bar).
    2. **Cảnh báo người dùng (Warn the user):** Đẩy trách nhiệm bảo vệ sang phía người dùng.
    3. **Loại bỏ vấn đề (Remove the problem):** Vô hiệu hóa hoặc gỡ bỏ tính năng gây lỗi.
    4. **Sửa lỗi (Fix the problem):** Phương pháp được mong đợi nhất, nên thực hiện ngay khi có thể để tránh hiệu ứng dây chuyền.

Việc hiểu rõ các nguyên tắc và thành phần này là kiến thức bắt buộc đối với các ứng viên chứng chỉ CSSLP.

Chapter 2 Security Design Principles

System Tenets

Dựa trên nội dung Chương 2 của tài liệu **"CSSLP® Certification All-in-One Exam Guide, Third Edition"**, các **Nguyên lý Hệ thống (System Tenets)** là những yếu tố nền tảng cần được xây dựng bên trong một hệ thống phần mềm để đảm bảo tính an ninh khi vận hành.

Dưới đây là nội dung chi tiết của ba nguyên lý cốt lõi này:

1. Quản lý Phiên (Session Management)

- **Định nghĩa:** Đây là việc quản lý các phiên giao tiếp giữa các thành phần khác nhau bên trong hệ thống hoặc giữa người dùng và hệ thống.
- **Mục tiêu:** Cung cấp các thành phần cần thiết để bảo vệ hệ thống trong quá trình vận hành. Quản lý phiên đảm bảo rằng các kênh giao tiếp được duy trì đúng cách và an toàn.

2. Quản lý Ngoại lệ (Exception Management)

- **Khái niệm:** Là quy trình xử lý các tình huống khi hệ thống gặp phải một điều kiện không xác định hoặc nhận đầu vào dẫn đến lỗi.
- **Tiêu chuẩn cho Quản lý Ngoại lệ an toàn:**
    1. **Phát hiện và xử lý:** Mọi ngoại lệ phải được hệ thống phát hiện và xử lý triệt để.
    2. **Trạng thái an toàn khi thất bại (Fail to a safe state):** Hệ thống phải được thiết kế sao cho khi xảy ra lỗi, nó không rơi vào trạng thái mất an ninh (ví dụ: không được vô tình cấp quyền truy cập khi cơ sở dữ liệu xác thực gặp lỗi).
    3. **Không rò rỉ thông tin:** Các thông điệp giao tiếp liên quan đến ngoại lệ không được phép làm lộ thông tin nhạy cảm (ví dụ: không được tiết lộ chuỗi kết nối - connection string hoặc thông tin đăng nhập cơ sở dữ liệu trong thông báo lỗi).

- **Lưu ý:** Nếu hệ thống không tự xử lý ngoại lệ mà để hệ điều hành (OS) xử lý, điều này thường dẫn đến các rủi ro an ninh như leo thang đặc quyền.

3. Quản lý Cấu hình (Configuration Management)

- **Vai trò:** Đảm bảo tính tin cậy của phần mềm thông qua việc quản lý các kết nối chức năng trong các hệ thống phức tạp và tích hợp.
- **Các mục tiêu cấu hình điển hình:** Bao gồm các tham số khởi tạo, chuỗi kết nối, đường dẫn, khóa (keys) và các biến liên quan khác.
- **Các biện pháp bảo vệ:**
    - **Ngăn chặn thay đổi trái phép:** Các cấu hình phải được bảo vệ chặt chẽ để hệ thống luôn ở trạng thái an toàn.
    - **Tách biệt nhiệm vụ (Separation of Duties):** Thường xuyên áp dụng việc tách biệt giữa nhân viên vận hành sản xuất (production) và nhân viên phát triển/kiểm thử (development/test) để tránh việc làm hỏng các cấu hình đã được phê duyệt trên môi trường thực tế.
    - **Kiểm soát thay đổi:** Sử dụng các hội đồng kiểm soát thay đổi (Change Control Boards) và hệ thống quy trình công việc để giám sát mọi điều chỉnh đối với cấu hình hệ thống.

Việc hiểu rõ và áp dụng các nguyên lý này là nền tảng để phát triển phần mềm an toàn và là kiến thức quan trọng cho kỳ thi CSSLP.

Secure Design Tenets

Dựa trên nội dung Chương 2 của cuốn sách, các **Nguyên lý Thiết kế Bảo mật (Secure Design Tenets)** là nền tảng cốt lõi được xây dựng một cách có tính toán để đảm bảo phần mềm vận hành an toàn,. Dưới đây là nội dung chi tiết của từng nguyên lý:

1. Bảo mật vừa đủ (Good Enough Security)

- **Nguyên tắc:** Không có bảo mật tuyệt đối; bảo mật luôn là một sự đánh đổi với các yếu tố khác của hệ thống.

- **Thực thi:** Cần xác định mức độ bảo mật phù hợp ngay từ giai đoạn thiết kế để cân bằng giữa chi phí bảo vệ và giá trị của tài sản (ví dụ: không dùng mật mã cấp quốc gia để bảo vệ thông tin công khai).

2. Đặc quyền tối thiểu (Least Privilege)

- **Nguyên tắc:** Một chủ thể chỉ nên được cấp **những quyền hạn vừa đủ** để thực hiện nhiệm vụ hiện tại, không hơn không kém.
- **Thực thi:** Hạn chế quyền hạn giúp giảm thiểu thiệt hại nếu hệ thống bị xâm nhập. Các chương trình nên chạy trong ngữ cảnh bảo mật tối thiểu cần thiết thay vì luôn chạy với quyền quản trị (như lỗi bảo mật của chương trình Sendmail trước đây).

3. Tách biệt nhiệm vụ (Separation of Duties)

- **Nguyên tắc:** Đảm bảo rằng một nhiệm vụ quan trọng phải có **nhiều hơn một cá nhân** tham gia để hoàn thành.
- **Thực thi:** Thiết kế phần mềm yêu cầu nhiều điều kiện phải được đáp ứng từ các bên khác nhau trước khi hoàn tất một giao dịch, nhằm ngăn chặn hành vi lạm dụng hệ thống từ một cá nhân duy nhất.

4. Phòng thủ theo chiều sâu (Defense in Depth)

- **Nguyên tắc:** Sử dụng **nhiều lớp phòng vệ chồng lên nhau**; nếu một lớp bị phá vỡ, các lớp khác vẫn có thể bảo vệ hệ thống.
- **Thực thi:** Các lớp phòng thủ nên có **tính chất khác nhau (diversity of defense)**, ví dụ như kết hợp giữa mã hóa và kiểm soát truy cập. Mục tiêu là làm cho chi phí tấn công vượt quá giá trị mà kẻ tấn công có thể thu được.

5. Thất bại an toàn (Fail-Safe)

- **Nguyên tắc:** Khi hệ thống gặp lỗi hoặc thất bại, nó phải rơi vào **trạng thái an toàn**
- **Thực thi:** Áp dụng khái niệm **"từ chối mặc định" (explicit deny)**—mọi chức năng không được phép cụ thể sẽ bị chặn. Hệ thống cần được thiết kế để suy giảm chức năng một cách linh hoạt (degrade gracefully) và quay lại trạng thái bình thường nhanh nhất có thể.

6. Kinh tế của cơ chế (Economy of Mechanism)

- **Nguyên tắc:** Giữ cho các cơ chế bảo mật **càng đơn giản và tinh tế càng tốt**.
- **Thực thi:** Sự phức tạp là kẻ thù của bảo mật vì nó khó hiểu và dễ tạo ra sai sót. Loại bỏ các dịch vụ và giao thức không thiết yếu để giảm thiểu diện tích bị tấn công (attack surface).

7. Hòa giải đầy đủ (Complete Mediation)

- **Nguyên tắc:** Mọi yêu cầu truy cập vào đối tượng phải được **kiểm tra quyền hạn mỗi khi thực hiện**.
- **Thực thi:** Hệ thống phải đảm bảo cơ chế ủy quyền không bao giờ bị bỏ qua, ngay cả khi truy cập lặp đi lặp lại. Điều này thường được thực hiện thông qua nhân bảo mật (security kernel) trong hệ điều hành.

8. Thiết kế mở (Open Design)

- **Nguyên tắc:** Bảo mật của hệ thống không nên phụ thuộc vào việc giữ bí mật thiết kế hay thuật toán.
- **Thực thi:** Trái ngược với "bảo mật thông qua sự mơ hồ" (security through obscurity), thiết kế mở dựa trên các thuật toán công khai và sự an toàn nằm ở **việc bảo vệ khóa (keys)**.

9. Cơ chế chung ít nhất (Least Common Mechanism)

- **Nguyên tắc:** Tránh việc chia sẻ các cơ chế chung giữa nhiều người dùng hoặc tiến trình để ngăn chặn việc rò rỉ thông tin ngoài ý muốn.
- **Thực thi:** Nên ưu tiên các tiến trình riêng biệt cho từng cấp độ bảo mật thay vì một tiến trình duy nhất xử lý cho tất cả.

10. Sự chấp nhận về tâm lý (Psychological Acceptability)

- **Nguyên tắc:** Các biện pháp bảo mật phải **dễ sử dụng và minh bạch** với người dùng.
- **Thực thi:** Nếu bảo mật quá gây cản trở, người dùng sẽ tìm cách vượt qua nó (ví dụ: mã hóa file đính kèm để tránh bộ lọc bảo mật của email). Bảo mật nên "vô hình" như không khí nhưng luôn hiện hữu để phục vụ nhu cầu.

11. Mắt xích yếu nhất (Weakest Link)

- **Nguyên tắc:** Một hệ thống chỉ mạnh bằng mắt xích yếu nhất của nó.
- **Thực thi:** Kẻ tấn công luôn tìm điểm yếu nhất để xâm nhập; do đó, việc tập trung tài nguyên bảo mật vào điểm này là hiệu quả nhất.

12. Tận dụng các thành phần hiện có (Leverage Existing Components)

- **Nguyên tắc:** Tái sử dụng các thành phần đã được kiểm chứng để giảm thiểu rủi ro từ các lỗ hổng mới.
- **Thực thi:** Việc tái sử dụng giúp giảm diện tích bề mặt tấn công nhưng cần cẩn trọng với kịch bản "độc canh" (monoculture)—nơi một lỗi duy nhất có thể ảnh hưởng đến toàn bộ hệ thống ở quy mô lớn.

13. Điểm yếu đơn nhất (Single Point of Failure)

- **Nguyên tắc:** Thiết kế hệ thống sao cho không có bất kỳ thành phần đơn lẻ nào mà khi nó hỏng sẽ làm **toàn bộ hệ thống sụp đổ**.
- **Thực thi:** Cần phân tích tất cả các điểm có khả năng gây lỗi cho cả ba thuộc tính CIA (Bảo mật, Toàn vẹn, Sẵn sàng) để ngăn chặn các thất bại thảm khốc.

Security Models

Dựa trên nội dung Chương 2 của tài liệu, các đầu mục con trong phần **Security Models (Các mô hình bảo mật)** được liệt kê chi tiết như sau:

1. Access Control Models (Các mô hình kiểm soát truy cập)

Phần này bao gồm các cơ chế và phương pháp xác định quyền hạn của chủ thể trên đối tượng:

- **MAC Model** (Mô hình kiểm soát truy cập bắt buộc)
- **DAC Model** (Mô hình kiểm soát truy cập tùy quyền)
- **Role-Based Access Control** (Kiểm soát truy cập dựa trên vai trò - RBAC)
- **Rule-Based Access Control** (Kiểm soát truy cập dựa trên quy tắc - RBAC)
- **Access Control Matrix Model** (Mô hình ma trận kiểm soát truy cập)

- **Attribute-Based Access Control** (Kiểm soát truy cập dựa trên thuộc tính - ABAC)

2. Các mô hình bảo mật và toàn vẹn cụ thể

- **Bell-LaPadula Confidentiality Model** (Mô hình bảo mật Bell-LaPadula)
- **Take-Grant Model** (Mô hình Take-Grant)
- **Multilevel Security Model** (Mô hình bảo mật đa cấp)
- **Integrity Models** (Các mô hình toàn vẹn)
    - **Biba Integrity Model** (Mô hình toàn vẹn Biba)
    - **Clark-Wilson Model** (Mô hình Clark-Wilson)

3. Các mô hình luồng thông tin và phân tích hệ thống

- **Information Flow Models** (Các mô hình dòng thông tin)
- **Brewer-Nash Model (Chinese Wall)** (Mô hình Brewer-Nash)
- **Data Flow Diagrams** (Sơ đồ dòng dữ liệu - DFD)
- **Use Case Models** (Các mô hình Use Case, bao gồm cả Misuse Cases),

4. Các mô hình đảm bảo và vận hành

- **Assurance Models** (Các mô hình đảm bảo)
- **The Operational Model of Security** (Mô hình vận hành bảo mật - Prevention + Detection + Response)
- **The NIST CSF Model** (Mô hình khung an ninh mạng NIST)

Dưới đây là nội dung chi tiết về các **Mô hình Kiểm soát Truy cập (Access Control Models)** dựa trên tài liệu Chương 2:

Tổng quan về Kiểm soát Truy cập

Kiểm soát truy cập là cơ chế xác định những hành động mà một **chủ thể** (subject) có thể thực hiện trên các **đối tượng** (objects) cụ thể. Quá trình này giả định rằng danh tính của người dùng đã được xác minh thông qua quá trình xác thực trước đó. Cơ chế phổ biến nhất

được sử dụng là **Danh sách Kiểm soát Truy cập (ACL)** – một danh sách định danh các chủ thể và quyền truy cập cụ thể của họ (như đọc, ghi, thực thi) đối với một đối tượng.

1. Mô hình MAC (Mandatory Access Control - Kiểm soát truy cập bắt buộc)

- **Nguồn gốc:** Có rễ từ các hệ thống kiểm soát quân sự.
- **Cơ chế:** Hạn chế truy cập dựa trên **độ nhạy cảm** của thông tin (được thể hiện bằng nhãn - labels) và **sự cho phép chính thức** (mức độ tin cậy - clearance) của chủ thể.
- **Đặc điểm:** Chủ sở hữu đối tượng không có quyền quyết định việc cấp quyền cho người khác; đây là nhiệm vụ của hệ điều hành. Mọi mối quan hệ giữa chủ thể và đối tượng phải được thiết kế và xác định trước khi đưa vào hệ thống.
- **Ví dụ:** SELinux là một hệ thống dựa trên mô hình MAC.

2. Mô hình DAC (Discretionary Access Control - Kiểm soát truy cập tùy quyền)

- **Đặc điểm:** Đây là mô hình được sử dụng phổ biến nhất.
- **Cơ chế:** Quyền truy cập được quyết định dựa trên danh tính của chủ thể hoặc nhóm mà họ thuộc về.
- **Quyền hạn: Chủ sở hữu đối tượng có toàn quyền quyết định** ai được phép truy cập và mức độ truy cập cụ thể là gì.
- **Ưu/Nhược điểm:** Điểm mạnh là sự đơn giản, nhưng điểm yếu là nó mang tính tùy chọn (tùy quyền). Việc quản lý có thể trở nên quá tải nếu chủ sở hữu phải đưa ra hàng ngàn quyết định truy cập cá nhân; vì vậy, các mô hình dựa trên nhóm thường được áp dụng để tăng khả năng mở rộng.

3. Kiểm soát Truy cập dựa trên Vai trò (Role-Based Access Control - RBAC)

- **Cơ chế:** Thay vì gán quyền cho từng cá nhân, người dùng được gán vào các **vai trò** (roles) cụ thể (ví dụ: lập trình viên, người kiểm thử, quản lý). Các quyền truy cập sau đó được gán cho chính các vai trò này.
- **Lợi ích:** Giúp giảm đáng kể khối lượng công việc quản trị. Trong một tổ chức lớn với hàng trăm nhân viên và hàng ngàn đối tượng, việc sử dụng vai trò giúp giảm số

lượng gán quyền đi nhiều bậc (orders of magnitude). Các vai trò không mang tính loại trừ; một người dùng có thể đảm nhận một hoặc nhiều vai trò cùng lúc.

4. Kiểm soát Truy cập dựa trên Quy tắc (Rule-Based Access Control - RBAC)

- **Cơ chế:** Sử dụng các **quy tắc** được thiết lập trong ACL để đưa ra quyết định cấp quyền.
- **Ví dụ:** Một quy tắc có thể quy định rằng nhân viên không thuộc cấp quản lý thì không được truy cập vào hồ sơ lương sau giờ làm việc hoặc vào cuối tuần.
- **Sự kết hợp:** Mô hình này thường được sử dụng bổ trợ cho các phương pháp khác. Ví dụ: Dùng vai trò để giới hạn quyền truy cập theo công việc, và dùng quy tắc để giới hạn theo thời gian hoặc vị trí mạng.

5. Mô hình Ma trận Kiểm soát Truy cập (Access Control Matrix Model)

- **Định nghĩa:** Là dạng ký hiệu đơn giản dưới dạng **bảng (ma trận)**, liệt kê các hành động được phép giữa chủ thể và đối tượng.
- **Ưu/Nhược điểm:** Thiết kế rất đơn giản nhưng cực kỳ khó triển khai thực tế và **không có khả năng mở rộng tốt**. Khi số lượng chủ thể và đối tượng tăng lên, số lượng ô trong ma trận tăng theo cấp số nhân (tích của hai đại lượng), dẫn đến việc quản lý ACL trở nên khổng lồ và phức tạp.

6. Kiểm soát Truy cập dựa trên Thuộc tính (Attribute-Based Access Control - ABAC)

- **Cơ chế:** Đây là một sơ đồ mới, đưa ra quyết định truy cập dựa trên các **thuộc tính** gắn liền với danh tính, tài nguyên hoặc môi trường (vị trí, thời gian).
- **Tính linh hoạt:** Quyền truy cập có thể thay đổi tùy ngữ cảnh. Ví dụ: Một bác sĩ có thể có quyền truy cập khác nhau tùy thuộc vào việc họ đang xem hồ sơ của bệnh nhân nào, vào lúc nào và ở đâu.
- **Triển khai:** Thường được thể hiện thông qua ngôn ngữ **XACML**, một tiêu chuẩn dùng để thực thi các sơ đồ kiểm soát dựa trên thuộc tính và chính sách.

Dưới đây là nội dung chi tiết về các mô hình bảo mật và toàn vẹn cụ thể được trình bày trong Chương 2 của tài liệu:

1. Mô hình Bảo mật Bell-LaPadula (Bell-LaPadula Confidentiality Model)

Đây là mô hình được thiết kế đặc biệt để **duy trì tính bảo mật** của thông tin. Mô hình này kết hợp cả cơ chế kiểm soát truy cập bắt buộc (MAC) và tùy quyền (DAC) để thực thi hai nguyên tắc bảo mật cơ bản:

- **Quy tắc Bảo mật Đơn giản (Simple Security Rule - No Read-Up):** Quy định rằng một chủ thể không thể đọc thông tin từ một đối tượng có mức phân loại bảo mật cao hơn mức mà chủ thể đó sở hữu. Điều này đảm bảo người dùng chỉ được xem những gì họ được phép.
- **Đặc tính * (*-Property - No Write-Down):** Quy định rằng một chủ thể chỉ có thể ghi vào một đối tượng nếu mức phân loại của chủ thể đó thấp hơn hoặc bằng mức phân loại của đối tượng. Nguyên tắc này ngăn chặn việc rò rỉ thông tin từ mức bảo mật cao xuống các mức thấp hơn (ví dụ: ngăn việc đăng thông tin mật lên một trang web công khai).

2. Mô hình Take-Grant

Mô hình này dựa trên **lý thuyết đồ thị** và mang tính lý thuyết toán học. Hệ thống được biểu diễn dưới dạng đồ thị có hướng, trong đó các đỉnh là chủ thể/đối tượng và các cạnh biểu thị quyền hạn giữa chúng.

- **Quyền đặc thù:** Có hai quyền duy nhất là *Take* (lấy) và *Grant* (cấp), bên cạnh các quyền cơ bản là *Read* (đọc) và *Write* (ghi).
- **Giá trị:** Mô hình này thường không dùng để triển khai hệ thống cụ thể mà dùng để **phân tích các thiết lập quyền hạn**, giúp trả lời câu hỏi liệu một triển khai có đầy đủ hay có khả năng làm rò rỉ thông tin hay không.

3. Mô hình Bảo mật Đa cấp (Multilevel Security Model)

Đây là mô hình mô tả bảo mật bằng cách sử dụng các **nhãn (labels)** để phân tách các nhóm thông tin và tiến trình.

- **Cơ chế "Vùng chứa":** Các nhóm nhãn hoạt động như những vùng chứa dữ liệu (ví dụ: Tối mật, Bí mật, Thân mật).

- **Quy tắc nhãn cao nhất:** Một tài liệu tổng hợp sẽ mang nhãn của thành phần có mức bảo mật cao nhất bên trong nó. Ví dụ, nếu một tài liệu có bất kỳ thông tin "Tối mật" nào, toàn bộ tài liệu đó phải được bảo vệ ở mức "Tối mật".

4. Các Mô hình Toàn vẹn (Integrity Models)

Ngược lại với mô hình bảo mật, các mô hình này tập trung vào việc đảm bảo **tính chính xác và độ tin cậy** của thông tin.

Mô hình Toàn vẹn Biba (Biba Integrity Model)

Sử dụng các mức độ toàn vẹn để phân chia quyền hạn, dựa trên nguyên tắc dữ liệu ở mức toàn vẹn cao hơn thì đáng tin cậy hơn. Biba có hai quy tắc chính:

- **Quy tắc No-Write-Up (Low-Water-Mark Policy):** Ngăn chặn chủ thể ghi dữ liệu vào các đối tượng có mức toàn vẹn cao hơn mình.
- **Quy tắc hạ thấp mức toàn vẹn:** Nếu một chủ thể tác động lên một đối tượng có mức toàn vẹn thấp hơn, mức toàn vẹn của chính chủ thể đó sẽ bị hạ xuống mức tương ứng. Điều này đảm bảo rằng sự tin cậy vào dữ liệu mới tạo ra không bao giờ cao hơn mức độ tin cậy của nguồn dữ liệu ban đầu.

Mô hình Clark-Wilson

Mô hình này tiếp cận theo hướng sử dụng các **giao dịch (transactions)** làm cơ sở cho các quy tắc toàn vẹn.

- **Thành phần dữ liệu:** Chia dữ liệu thành **CDI** (dữ liệu chịu kiểm soát toàn vẹn) và **UDI** (dữ liệu không chịu kiểm soát).
- **Thành phần quy trình:**
    - **IVPs (Integrity Verification Processes):** Quy trình xác minh đảm bảo dữ liệu CDI luôn ở trạng thái hợp lệ.
    - **TPs (Transformation Processes):** Các quy trình thay đổi trạng thái dữ liệu từ trạng thái hợp lệ này sang trạng thái hợp lệ khác.
- **Nguyên tắc cốt lõi:** Người dùng **không thể thay đổi dữ liệu trực tiếp** mà phải thông qua các quy trình TPs được tin tưởng. Điều này cho phép kiểm soát chặt chẽ

những cá nhân có quyền thực hiện các thay đổi quan trọng (ví dụ: thay đổi số dư tài khoản ngân hàng).

Dưới đây là nội dung chi tiết về **Các mô hình luồng thông tin và phân tích hệ thống** dựa trên Chương 2 của tài liệu:

1. Các mô hình dòng thông tin (Information Flow Models)

Mô hình này tập trung vào cách thức thông tin di chuyển và được xử lý trong toàn bộ hệ thống.

- **Mục tiêu:** Hiểu rõ cách thông tin di chuyển qua hệ thống giúp xác định các cơ chế bảo vệ cần thiết.
- **Các trạng thái cần bảo vệ:** Thông tin phải được bảo vệ trong cả ba trạng thái: **khi đang lưu trữ (at rest)**, **khi đang truyền tải (in transit)**, và **khi đang được sử dụng (in use)**.
- **Giá trị:** Bằng cách khám phá các khía cạnh của luồng dữ liệu, nhà phát triển có thể áp dụng các biện pháp bảo vệ phù hợp cho từng giai đoạn di chuyển của thông tin.

2. Mô hình Brewer-Nash (Chinese Wall)

Đây là mô hình được thiết kế đặc biệt để thực thi tính bảo mật trong các hoạt động của doanh nghiệp thương mại.

- **Ngăn chặn xung đột lợi ích:** Trong môi trường tài chính hoặc tư vấn, nhân viên có thể tiếp cận "thông tin nội bộ" của một khách hàng. Mô hình này ngăn chặn việc chia sẻ thông tin đó với các khách hàng khác hoặc các bộ phận có khả năng gây xung đột lợi ích.
- **Cách tiếp cận ba yếu tố:**
    - **Công nghệ:** Sử dụng các rào cản kỹ thuật để chặn quyền truy cập giữa các nhóm dữ liệu mâu thuẫn.
    - **Con người:** Đào tạo nhân viên để họ không làm tổn hại đến sự phân tách thông tin.

- **Quy trình:** Thiết lập các chính sách đảm bảo công nghệ và con người phối hợp hiệu quả để ngăn chặn sự xâm nhập dữ liệu.

3. Sơ đồ dòng dữ liệu (Data Flow Diagrams - DFD)

DFD là công cụ đồ họa dùng để ghi lại cách dữ liệu được lưu trữ, di chuyển và xử lý trong một hệ thống.

- **Cấu trúc phân lớp:** DFD được xây dựng theo các cấp độ để quản lý độ phức tạp:
  - **Level 0:** Cái nhìn bối cảnh cấp cao về dòng dữ liệu toàn hệ thống.
  - **Level 1:** Mở rộng và chi tiết hóa các yếu tố từ sơ đồ Level 0.
  - **Level 2:** Sơ đồ chi tiết nhất (mức thấp nhất), mô tả cụ thể từng quy trình xử lý dữ liệu.
- **Tầm quan trọng:** Việc hiểu rõ lộ trình di chuyển của dữ liệu là yếu tố thiết yếu để thiết kế các biện pháp bảo mật chức năng phù hợp.

4. Các mô hình Use Case và Misuse Case

Trong khi DFD nhìn hệ thống từ góc độ luồng thông tin, mô hình Use Case xem xét hệ thống từ **góc độ chức năng**.

- **Use Case (Trường hợp sử dụng):** Mô tả cách hệ thống sử dụng dữ liệu để thực hiện các chức năng bình thường theo ý định thiết kế.
- **Misuse Case (Trường hợp lạm dụng):** Mô tả các hành vi bất thường hoặc thù địch mà hệ thống không nên cho phép thực hiện.
- **Tầm quan trọng trong lập trình bảo mật:**
  - Việc hiểu những gì **không nên xảy ra (misuse cases)** cũng quan trọng như việc hiểu những gì **nên xảy ra (use cases)**.
  - Use Case giúp đại diện cho các yêu cầu bảo mật một cách rõ ràng.
  - Khi kết hợp với DFD, chúng cung cấp cái nhìn tổng thể toàn diện về cách hệ thống thao tác với dữ liệu, giúp hiểu thấu đáo mọi khía cạnh an ninh.

Dưới đây là nội dung chi tiết về **Các mô hình đảm bảo và vận hành** dựa trên tài liệu Chương 2:

1. Các mô hình đảm bảo (Assurance Models)

Đảm bảo phần mềm (Software Assurance) được định nghĩa là **"mức độ tin cậy rằng phần mềm không có lỗ hổng, dù là do cố ý thiết kế hay vô tình rơi vào ở bất kỳ thời điểm nào trong vòng đời, và phần mềm vận hành đúng như dự định"**.

- **Sự dịch chuyển trọng tâm:** Mô hình này dịch chuyển sự chú ý sang các yếu tố **ngăn ngừa** trong thiết kế và xây dựng hệ thống, thay vì chỉ tập trung vào việc vá lỗi sau khi sản phẩm đã ra mắt.
- **Trường hợp đảm bảo (Assurance Cases):** Được sử dụng để xây dựng một tập hợp các lập luận có cấu trúc và bằng chứng đi kèm nhằm chứng minh các đặc tính an ninh cụ thể.
- **Cấu trúc như một vụ kiện:** Một Assurance Case được tổ chức tương tự hồ sơ pháp lý:
    1. Xác định mục tiêu tổng thể.
    2. Trình bày các bằng chứng cụ thể để chứng minh một "ranh giới kết quả", nơi các kết quả không mong muốn bị loại bỏ và các kết quả mong muốn được duy trì.
- **Trạng thái được đảm bảo:** Hệ thống được coi là "đảm bảo" khi có đủ bằng chứng để loại bỏ tất cả các trạng thái hoặc kết quả không mong muốn.

2. Mô hình vận hành bảo mật (The Operational Model of Security)

Mô hình này cung cấp một phương pháp đơn giản cho công tác quản lý thông qua ba phương pháp hành động chính:

- **Ngăn ngừa (Prevention):** Đây là nơi tập trung nhiều nỗ lực nhất vì các sự cố được ngăn ngừa sẽ bị loại bỏ hoàn toàn, không còn là mối lo ngại. Ví dụ: **Kiểm soát truy cập, Tường lửa, Mã hóa**.

- **Phát hiện (Detection):** Dành cho những vấn đề thoát khỏi các biện pháp ngăn ngừa. Nếu một vấn đề không bị ngăn ngừa và cũng không bị phát hiện, nó sẽ diễn ra mà không có bất kỳ sự can thiệp nào từ các chức năng bảo mật. Ví dụ: **Nhật ký kiểm toán (Audit logs), Hệ thống phát hiện xâm nhập (IDS), Honeypots**.
- **Phản hồi (Response):** Các yếu tố đã bị phát hiện vẫn cần những nỗ lực phản hồi để xử lý hậu quả và khôi phục hệ thống. Ví dụ: **Sao lưu (Backups), Đội phản ứng sự cố, Pháp y máy tính**.

3. Mô hình NIST CSF (The NIST Cybersecurity Framework)

NIST CSF là một khung tự nguyện bao gồm các tiêu chuẩn, hướng dẫn và thực hành tốt nhất để quản lý rủi ro liên quan đến an ninh mạng trong tổ chức.

- **Năm chức năng chính:** CSF sử dụng năm chức năng cốt lõi để phân loại các hoạt động an ninh mạng:
    1. **Xác định (Identify):** Hiểu rõ bối cảnh kinh doanh và các nguồn lực để quản lý rủi ro.
    2. **Bảo vệ (Protect):** Triển khai các biện pháp bảo vệ để đảm bảo cung cấp các dịch vụ thiết yếu.
    3. **Phát hiện (Detect):** Xác định sự xuất hiện của một sự kiện an ninh mạng.
    4. **Phản hồi (Respond):** Thực hiện hành động đối với sự cố được phát hiện.
    5. **Khôi phục (Recover):** Duy trì kế hoạch phục hồi và khôi phục các khả năng hoặc dịch vụ bị suy giảm do sự cố.
- **Giá trị ứng dụng:** Hiểu rõ các khái niệm này giúp ích cho các dự án phát triển phần mềm dự kiến sẽ vận hành trong môi trường áp dụng khung NIST CSF.

Adversaries

Nội dung chi tiết của phần **Adversaries (Đối thủ/Kẻ tấn công)** trong Chương 2 của tài liệu tập trung vào việc phân loại các đối thủ dựa trên kỹ năng, động lực và khả năng của họ để giúp các nhà phát triển hiểu rõ bối cảnh đe dọa mà phần mềm phải đối mặt.

Dưới đây là chi tiết các đề mục:

1. Phân loại đối thủ theo cấp độ kỹ năng (Adversary Type)

Các đối thủ được phân loại theo trình độ kỹ năng. Khi trình độ kỹ năng tăng lên thì số lượng kẻ tấn công trong danh mục đó thường giảm xuống.

- **Script Kiddie:** Đây là dạng tấn công cơ bản nhất, chiếm khoảng **80% đến 85%** cộng đồng tấn công. Họ không có nhiều kiến thức chuyên môn mà chỉ sử dụng các tập lệnh (scripts) có sẵn trên mạng để thực hiện tấn công. Mặc dù các kỹ thuật của họ đã được biết rõ và có thể phòng thủ, nhưng số lượng quá lớn của họ tạo ra "tiếng ồn nền" liên tục mà hệ thống phải xử lý.
- **Hacker:** Nhóm này chiếm khoảng **15% đến 20%** dân số tấn công. Họ có kiến thức về cách hệ thống vận hành và khả năng thao túng hệ thống để đạt mục đích. Thuật ngữ **"Cracker"** thường được dùng để chỉ những hacker có ý đồ xấu. Đây là nhóm đối thủ chính vì kỹ năng và động lực của họ có thể gây ra thiệt hại thảm khốc cho tổ chức.
- **Elite (Hacker tinh nhuệ):** Chỉ chiếm một phần rất nhỏ (**1% đến 3%**). Họ sở hữu kỹ năng cực kỳ cao, có khả năng che giấu dấu vết khiến hành động của mình gần như không thể bị phát hiện. Nhóm này thường đứng sau các lỗ hổng **zero-day** (các lỗ hổng chưa được công bố). Việc dành nguồn lực để phòng thủ trước nhóm này thường không hiệu quả trừ khi tổ chức thuộc một ngành cực kỳ nhạy cảm.

2. Các nhóm đối thủ theo cấu trúc và nguồn lực (Adversary Groups)

Việc phân nhóm này giúp phân tích hiệu quả của các biện pháp phòng thủ dựa trên quy mô và nguồn lực của kẻ tấn công.

- **Mối đe dọa không cấu trúc (Unstructured Threat):** Những kẻ tấn công đơn lẻ (thường là script kiddies) với nguồn lực hạn chế, thiếu khả năng tập trung tấn công một mục tiêu trong thời gian dài. Họ thường tấn công theo cơ hội hơn là nhắm vào một mục tiêu cụ thể.

- **Mối đe dọa có cấu trúc (Structured Threat):** Các nhóm có tổ chức, có nhiệm vụ cụ thể và nguồn lực để thực hiện các cuộc tấn công kéo dài nhiều tháng. Họ có khả năng phát triển mã độc riêng, sử dụng **botnet** và các kỹ thuật tinh vi để đạt được mục đích cuối cùng là đánh cắp thông tin có giá trị.
- **Mối đe dọa cấu trúc cao (Highly Structured Threat):** Các tổ chức tội phạm có nguồn lực tài chính lớn, thuê đội ngũ lập trình viên chuyên nghiệp để phát triển phần mềm độc hại (crimeware) phục vụ cho việc đánh cắp danh tính và thông tin.
- **Mối đe dọa từ quốc gia (Nation-State Threat):** Đây là nhóm có nguồn lực lớn nhất, thường liên quan đến các hoạt động gián điệp và tấn công có chủ đích kéo dài (**APT - Advanced Persistent Threat**). Mục tiêu của APT là xâm nhập âm thầm, sống trong hệ thống như một người nội bộ để trích xuất thông tin quý giá một cách tinh vi mà không bị phát hiện.

3. Đe dọa Nội bộ và Bên ngoài (Insider vs. Outsider Threat)

- **Insiders (Người nội bộ):** Là những người có quyền truy cập hợp pháp vào hệ thống. Đây là mối đe dọa nguy hiểm vì họ đã vượt qua được rào cản khó khăn nhất là quyền truy cập ban đầu, có kiến thức về điểm yếu của hệ thống và vị trí của các luồng dữ liệu giá trị.
- **Outsiders (Người bên ngoài):** Những người không có quyền truy cập hợp pháp.

4. Sự thay đổi của bối cảnh đe dọa (Threat Landscape Shift)

Trước đây, bối cảnh đe dọa được xác định bởi loại tác nhân (hacker khám phá, nhóm hoạt động chính trị - hacktivists, hoặc quốc gia gián điệp). Tuy nhiên, từ khoảng năm 2005, đã có một sự thay đổi lớn: **tội phạm hóa không gian mạng**. Các nhóm tội phạm đã tìm ra cách kiếm tiền từ các lỗ hổng, dẫn đến sự bùng nổ các phương thức tấn công.

Hiện nay, **mọi phần mềm đều là mục tiêu**. Tội phạm không chỉ nhắm vào các ngân hàng lớn hay cơ quan chính phủ mà còn nhắm vào các doanh nghiệp nhỏ và cá nhân vì lợi nhuận từ số lượng lớn nạn nhân là rất đáng kể và khó bị truy tố.

Part II Secure Software Requirements

Chapter 3 Define Software Security Requirements

Functional Requirements

Dưới đây là nội dung chi tiết về các thành phần cốt lõi của **Yêu cầu chức năng (Functional Requirements)** trong Chương 3 của tài liệu:

1. Định nghĩa Vai trò và Người dùng (Role and User Definitions)

Định nghĩa vai trò và người dùng là các tuyên bố về việc **ai sẽ sử dụng chức năng nào** của phần mềm.

- **Cấp độ khái quát:** Ban đầu, các định nghĩa này ở dạng chung, chẳng hạn như nhóm người dùng nào được phép sử dụng hệ thống.
- **Cấp độ chi tiết:** Các bước tinh chỉnh tiếp theo sẽ liệt kê cụ thể người dùng nào được phép thực hiện chức năng nào như một phần công việc của họ.
- **Thuật ngữ chuyên môn:** Trong khoa học máy tính, người dùng thường được gọi là **chủ thể (subjects)**. Những định nghĩa chi tiết này tạo thành một phần của việc xác định các trường hợp sử dụng (use cases).

2. Đối tượng (Objects)

Đối tượng là những **mục mà người dùng (chủ thể) tương tác** trong quá trình vận hành hệ thống.

- **Phạm vi:** Bất cứ thứ gì có thể truy cập được đều là đối tượng, bao gồm: tập tin, bản ghi cơ sở dữ liệu, hệ thống hoặc các thành phần của chương trình.
- **Kiểm soát truy cập:** Một phương pháp phổ biến để kiểm soát là thông qua các **danh sách kiểm soát truy cập (ACL)** được gán cho các đối tượng.
- **Tầm quan trọng:** Việc xác định cụ thể các đối tượng và chức năng của chúng đảm bảo toàn bộ nhóm phát triển sử dụng một bộ đối tượng chung và kiểm soát các tương tác một cách thích hợp.

3. Hoạt động/Hành động (Activities/Actions)

Đây là các **sự kiện được phép** mà một chủ thể có thể thực hiện trên một đối tượng liên quan.

- **Đặc thù theo đối tượng:** Bộ hoạt động cụ thể được xác định bởi chính đối tượng đó. Ví dụ: một bản ghi cơ sở dữ liệu có thể được tạo, đọc, cập nhật hoặc xóa (CRUD); một tập tin có thể được truy cập, sửa đổi, xóa, v.v..
- **Yêu cầu tài liệu:** Mọi hành động có thể xảy ra đối với từng đối tượng cần được định nghĩa và ghi chép lại.
- **Rủi ro:** Các chức năng không được lập hồ sơ thường là nguyên nhân dẫn đến thất bại của hệ thống khi người dùng tìm thấy một hoạt động không được xem xét trong thiết kế nhưng vẫn xảy ra, cho phép các hành vi nằm ngoài tầm kiểm soát.

4. Ma trận Chủ thể-Đối tượng-Hoạt động (Subject-Object-Activity Matrix)

Ma trận này là một công cụ giúp các nhà thiết kế và lập trình viên định nghĩa chính xác mối quan hệ tương tác trong hệ thống.

- **Cấu trúc:** Chủ thể đại diện cho "ai", Đối tượng đại diện cho "cái gì", và Hoạt động đại diện cho "như thế nào".
- **Cách xây dựng:** Với mỗi chủ thể, tài liệu sẽ liệt kê tất cả các đối tượng cùng với các hoạt động tương ứng cho từng đối tượng đó, sau đó xác định yêu cầu bảo mật cho từng trạng thái.
- **Kết quả:** Quá trình này tạo ra hai danh sách quan trọng:
    - **Danh sách tổng (Master list) các hành động được cho phép:** Hữu ích để tạo các trường hợp sử dụng (use cases).
    - **Danh sách tổng các hành động bị từ chối:** Hữu ích để tạo các trường hợp lạm dụng (misuse cases).
- **Giá trị:** Ma trận này cho phép truyền đạt một cách súc tích và rõ ràng về các tương tác hệ thống được phép.

Dưới đây là nội dung chi tiết về **Trường hợp sử dụng (Use Cases)** trong phần yêu cầu chức năng của Chương 3:

1. Khái niệm và Mục đích

- **Định nghĩa:** Một Use Case là một ví dụ cụ thể về một **hành vi dự kiến** của hệ thống.
- **Mục tiêu:** Đây là một kỹ thuật mạnh mẽ để xác định các yêu cầu chức năng dưới dạng thuật ngữ thân thiện với lập trình viên. Nó giúp định nghĩa hành vi dự kiến cho cả **đội ngũ phát triển và đội ngũ kiểm thử**.

2. Giá trị trong Lập trình Bảo mật

- **Giải quyết sự mơ hồ:** Use Case đặc biệt hữu ích trong việc mô tả các tình huống phức tạp, gây nhầm lẫn hoặc mơ hồ liên quan đến tương tác của người dùng với hệ thống.
- **Hỗ trợ thiết kế và kiểm thử:** Việc xây dựng Use Case tạo điều kiện cho việc thiết kế chính xác cả phần mềm và các thiết bị kiểm thử, bao phủ được những phần mà nếu không có Use Case sẽ dễ bị bỏ sót do yêu cầu được diễn đạt kém.
- **Không thay thế tài liệu yêu cầu:** Lưu ý rằng Use Case không nhằm mục đích thay thế việc lập hồ sơ các yêu cầu cụ thể và cũng không dùng cho *mọi* tương tác giữa chủ thể và đối tượng vì khối lượng tài liệu sẽ vượt quá tính tiện dụng của nó.

3. Mô hình hóa Use Case (Use-Case Modeling)

- **Thành phần chính:** Mô hình bao gồm các **tác nhân (actors)** đại diện cho người dùng và các **hành vi dự kiến** của hệ thống.
- **Biểu diễn đồ họa:** Mối quan hệ giữa tác nhân và hành vi thường được trình bày dưới dạng sơ đồ đồ họa để đơn giản hóa các quy trình kinh doanh phức tạp:
    - **Người dùng (Tác nhân):** Được vẽ bằng hình người que (stick figures). Các tác nhân này có thể là con người, vai trò hoặc quy trình (hệ thống không phải con người).
    - **Chức năng hệ thống:** Được vẽ bằng các hình elip.

- **Trình tự và Thời gian:** Khi trình tự của các hành động là quan trọng, một sơ đồ bổ sung (như sơ đồ trình tự) có thể được thêm vào để giải thích rõ hơn.

Tóm lại, Use Case giúp chuyển đổi các yêu cầu kinh doanh khô khan thành các kịch bản tương tác cụ thể, giúp nhóm phát triển hiểu rõ hệ thống phải làm gì và nhóm kiểm thử biết cần kiểm tra những gì để đảm bảo an ninh.

Dưới đây là nội dung chi tiết về **Trình tự và Thời gian (Sequencing and Timing)** trong phần yêu cầu chức năng của Chương 3:

1. Khái niệm và Bối cảnh

Trong mô hình vận hành đa luồng (multithreaded) và đồng thời (concurrent) ngày nay, các hệ thống khác nhau có thể cố gắng tương tác với cùng một đối tượng tại cùng một thời điểm. Điều này dẫn đến khả năng các sự kiện xảy ra **không đúng trình tự** do sự khác biệt về thời gian giữa các luồng chương trình khác nhau.

2. Các vấn đề an ninh trọng yếu

Các vấn đề về trình tự và thời gian ảnh hưởng trực tiếp đến cả thiết kế và triển khai các hoạt động dữ liệu:

- **Điều kiện đua (Race Conditions):** Đây là những lỗi phần mềm phát điểm từ việc các luồng hoặc quy trình khác nhau phụ thuộc vào một đối tượng hoặc tài nguyên chung.
    - **Ví dụ:** Một luồng phụ thuộc vào giá trị (A) đang bị thay đổi bởi một quy trình riêng biệt khác. Nếu hai quy trình này phụ thuộc lẫn nhau để hoàn thành công việc, một "khóa" (lock) có thể được tạo ra, dẫn đến các vòng lặp logic phức tạp.
    - **Cửa sổ đua (Race windows):** Là khoảng thời gian mà các luồng đồng thời có thể cạnh tranh để thay đổi cùng một đối tượng.
    - **Giải pháp:** Bước đầu tiên là xác định các "cửa sổ đua", sau đó thiết kế hệ thống sao cho chúng không bị gọi đồng thời, một quy trình được gọi là **loại trừ tương hỗ (mutual exclusion)**.

- **Tấn công TOC/TOU (Time of Check/Time of Use):** Từ góc độ tấn công, hệ thống dễ bị tổn thương bởi kiểu tấn công này. Đây là cuộc tấn công tận dụng khoảng cách thời gian giữa lúc chương trình **kiểm tra một giá trị** và lúc nó **sử dụng giá trị đó**, cho phép kẻ gian thực hiện các thao tác trái phép làm thay đổi kết quả của quy trình.
- **Vòng lặp vô tận (Infinite Loops):** Khi logic chương trình trở nên phức tạp (ví dụ: xử lý ngày cho năm nhuận), nếu không bao quát hết các điều kiện hoặc cơ chế thoát lỗi không hoạt động, chương trình có thể rơi vào trạng thái vòng lặp vô tận và ngừng phản ứng (treo máy).

3. Tầm quan trọng trong phát triển phần mềm an toàn

Việc hiểu rõ các điều kiện này xảy ra ở đâu và như thế nào là vô cùng quan trọng đối với đội ngũ phát triển:

- Phải đảm bảo mọi điều kiện trong từng vòng lặp lồng nhau đều được xử lý tích cực.
- Việc quản lý và hiểu rõ các cơ chế **khóa bản ghi (record locks)** là yếu tố thiết yếu trong môi trường lập trình đối tượng đa dạng hiện đại.

Tóm lại, yêu cầu về trình tự và thời gian đòi hỏi các nhà thiết kế phải tính toán đến sự tương tác phức tạp giữa các luồng để ngăn chặn các lỗ hổng logic có thể bị khai thác hoặc gây đình trệ hệ thống.

Dưới đây là nội dung chi tiết về **Tiêu chuẩn lập trình an toàn (Secure Coding Standards)** trong phần yêu cầu chức năng của Chương 3:

1. Định nghĩa và Mục đích

- **Khái niệm:** Tiêu chuẩn lập trình an toàn là tập hợp các quy tắc đặc thù cho từng ngôn ngữ lập trình và các thực hành được khuyến nghị để phục vụ lập trình an toàn.
- **Mục tiêu cốt lõi:** Việc mô tả các nguồn gây ra lỗ hổng và lỗi là một chuyện, nhưng việc **quy định các biểu mẫu cụ thể** khi triển khai mới là yếu tố giúp ngăn chặn các tập hợp lỗ hổng và các điều kiện có thể bị khai thác trong mã nguồn thông thường.

2. Các tiêu chuẩn theo ngôn ngữ (Language-Specific)

- Nhiều tiêu chuẩn lập trình an toàn đã được công bố bởi các tổ chức uy tín như **Viện Kỹ thuật Phần mềm (SEI)/CERT** thuộc Đại học Carnegie Mellon.
- Các tiêu chuẩn này cung cấp quy tắc riêng biệt cho các ngôn ngữ như **C, C++, và Java**, bao gồm cả các thực hành tốt nhất để lập trình an toàn trong từng ngôn ngữ đó.

3. Triển khai trong tổ chức

- **Tính lặp lại:** Lập trình ứng dụng có thể được coi là một dạng sản xuất. Việc kiểm soát và làm cho các quy trình này có khả năng lặp lại là một mục tiêu của vòng đời phát triển an toàn (SDL).
- **Tiêu chuẩn riêng của doanh nghiệp:** Các tổ chức nên áp dụng một bộ tiêu chuẩn lập trình an toàn riêng biệt cho doanh nghiệp mình và sử dụng các **khung phát triển ứng dụng an toàn (secure application development framework)**.
- **Thực hành tốt nhất (Best Practices):** Việc điều chỉnh và áp dụng các thực hành tốt nhất của ngành là một phần quan trọng của tiêu chuẩn lập trình an toàn trong doanh nghiệp.

4. Các lĩnh vực trọng tâm cụ thể

- **Bẫy và xử lý lỗi (Error Trapping and Handling):** Một vấn đề phổ biến trong nhiều chương trình là việc bẫy và xử lý lỗi kém. Tiêu chuẩn doanh nghiệp cần quy định quy tắc: mọi ngoại lệ và lỗi phải được bẫy bởi chính hàm tạo ra chúng và xử lý sao cho **không làm rò rỉ thông tin nội bộ** cho người dùng bên ngoài. Mỗi hàm nên thực hiện giảm thiểu lỗi hoàn toàn trước khi trả kết quả về quy trình gọi (calling routine).
- **Ghi nhật ký (Logging):** Tiêu chuẩn lập trình an toàn cần quy định cụ thể **cái gì, ở đâu và khi nào** các vấn đề cần được ghi nhật ký. Điều này giúp đảm bảo mức độ ghi nhật ký thích hợp và đơn giản hóa việc quản lý cơ sở hạ tầng nhật ký.

5. Giá trị mang lại

- **Tính nhất quán:** Việc sử dụng các tiêu chuẩn này giúp thực thi sự thống nhất trong toàn bộ quy trình phát triển.
- **Khả năng bảo trì:** Đây là yếu tố nền tảng giúp định nghĩa phương pháp luận của doanh nghiệp, hỗ trợ cả tính an ninh và khả năng bảo trì phần mềm lâu dài.
- **Hiểu biết chung:** Nó hỗ trợ tất cả các thành viên trong đội ngũ phát triển hiểu rõ cách thức hệ thống vận hành và các quy tắc cần tuân thủ.

Operational and Deployment Requirements

Phần **Yêu cầu Vận hành và Triển khai (Operational and Deployment Requirements)** trong Chương 3 tập trung vào cách thức phần mềm tương tác với môi trường thực tế và các hệ thống hiện có trong doanh nghiệp. Dưới đây là nội dung chi tiết:

1. Môi trường triển khai doanh nghiệp

Phần mềm hiếm khi vận hành độc lập hoàn toàn mà thường được triển khai trong một môi trường doanh nghiệp có sẵn.

- **Tiêu chuẩn công nghệ:** Các doanh nghiệp thường có tiêu chuẩn cụ thể về nền tảng (Linux, Windows), các phiên bản máy chủ cơ sở dữ liệu, máy chủ web và các thành phần hạ tầng khác.
- **Chiến lược tổng thể:** Các lựa chọn công nghệ này là một phần của chiến lược tổng thể của doanh nghiệp; dù có thể không tối ưu cho một trường hợp đơn lẻ, nhưng chúng tối ưu cho toàn bộ hệ thống chung.

2. Sự tương tác và kết nối hệ thống

Phần mềm mới cần phải kết nối với các hệ thống hiện có như cơ sở dữ liệu khách hàng, bản ghi nhân sự hoặc hồ sơ bộ phận.

- **Giao thức và kênh truyền thông:** Các yêu cầu vận hành được xây dựng dựa trên ý tưởng rằng hệ thống mới phải tương tác được với các hệ thống cũ thông qua các kênh và giao thức hiện có.

- **Độ chi tiết:** Yêu cầu vận hành chỉ thực sự có giá trị khi các thông số kỹ thuật chi tiết được công bố, thay vì chỉ dừng lại ở các định nghĩa cấp cao.

3. Triết lý "Bảo mật trong triển khai"

Một giải pháp SDLC hoàn chỉnh phải đảm bảo hệ thống **an toàn ngay từ thiết kế (secure by design)**, **an toàn theo mặc định (secure by default)** và **an toàn khi triển khai (secure in deployment)**.

- **Rủi ro từ cấu hình:** Một hệ thống được thiết kế an toàn nhưng lại được triển khai với cấu hình hoặc phương pháp không an toàn có thể khiến mọi nỗ lực bảo mật trước đó trở nên vô nghĩa.
- **An toàn theo mặc định:** Đảm bảo rằng nếu người dùng chọn các thiết lập mặc định của hệ thống, ứng dụng vẫn duy trì được trạng thái an toàn.

4. Tuân thủ tiêu chuẩn hạ tầng

Các tiêu chuẩn của tập đoàn, được quy định bởi bộ phận hạ tầng và nhân sự, sẽ dẫn dắt việc lựa chọn công nghệ.

- **Tính kết nối liền mạch:** Việc hiểu và tuân thủ tất cả các yêu cầu hạ tầng bắt buộc là yếu tố cần thiết để cho phép sự **kết nối liền mạch (seamless interconnectivity)** giữa các hệ thống khác nhau.

5. Quản lý triển khai hiện đại

Các thực hành phát triển phần mềm an toàn hiện đại bao gồm cả việc quản lý quá trình triển khai phần mềm.

- **Đường ống CI/CD:** Việc này thường được thực hiện thông qua đường ống **tích hợp liên tục (CI)** và **phân phối liên tục (CD)** an toàn. (Nội dung này được trình bày chi tiết hơn ở Chương 16).

Tóm lại, các yêu cầu vận hành và triển khai đảm bảo rằng hệ thống **vận hành đúng như thiết kế** khi được đưa vào môi trường thực tế và không tạo ra lỗ hổng do sai sót trong quá trình cài đặt hoặc tương tác hệ thống.

Connecting the Dots

Dưới đây là nội dung chi tiết cho phần **Connecting the Dots (Kết nối các điểm mấu chốt)** trong Chương 3 của tài liệu, giúp tổng kết và liên kết các khái niệm về yêu cầu bảo mật phần mềm:

1. Bản chất của các yêu cầu

- **Yêu cầu là nền tảng:** Các yêu cầu là những yếu tố căn bản được sử dụng trong quá trình phát triển bất kỳ dự án nào.
- **Nguồn gốc đa dạng:** Chúng đến từ nhiều nguồn khác nhau, bao gồm yêu cầu kinh doanh (business requirements), yêu cầu chức năng (functional requirements), và yêu cầu vận hành (operational requirements).
- **Kết quả từ mô hình hóa đe dọa:** Một trong những đầu ra quan trọng nhất của quá trình **mô hình hóa đe dọa (threat modeling)** chính là một tập hợp các yêu cầu cụ thể nhằm giảm thiểu (mitigate) các mối đe dọa đã biết hoặc dự kiến.

2. Thách thức "Phần chìm của tảng băng trôi"

- **Yêu cầu tường minh (Explicit):** Đây là những tính năng mà khách hàng trực tiếp yêu cầu. Tuy nhiên, chúng chỉ là "phần nổi của tảng băng trôi".
- **Yêu cầu ngầm định (Implied):** Khách hàng sẽ không bao giờ đưa ra yêu cầu như "phần mềm cần phải hoạt động được" vì đó là điều mặc nhiên. Thách thức lớn nhất đối với người làm bảo mật là phải **liệt kê và lập hồ sơ** tất cả các yêu cầu về an ninh, chức năng và vận hành vốn không được nêu ra vì chúng được coi là "ngầm hiểu".

3. Quá trình xây dựng danh mục yêu cầu

- **Sự khởi đầu khó khăn:** Việc tạo ra một danh sách tốt các yêu cầu bảo mật ban đầu rất khó khăn do có quá nhiều chi tiết và nguồn thông tin gây choáng ngợp.
- **Cải tiến theo thời gian:** Thông qua quá trình làm việc từ dự án này sang dự án khác, đội ngũ phát triển sẽ sử dụng một danh sách cốt lõi, sau đó tinh chỉnh và bổ sung thêm để dần xây dựng được một bộ yêu cầu toàn diện hơn theo thời gian.

4. Nguyên tắc cốt lõi: "Phải có trong hồ sơ"

- Thông điệp cuối cùng và quan trọng nhất của phần này là: **"Nếu bạn muốn đội ngũ phát triển làm một điều gì đó, điều đó cần phải được liệt kê cụ thể trong các yêu cầu của dự án"**.
- Bất kỳ chức năng nào không được lập hồ sơ rõ ràng đều có nguy cơ không được thực hiện hoặc thực hiện sai, gây ra lỗ hổng cho hệ thống.

Tóm lại, phần này nhấn mạnh rằng việc hiểu các yêu cầu không chỉ là thu thập mong muốn của khách hàng, mà là quá trình **kết nối tất cả các kiến thức** về thiết kế an toàn, các rủi ro từ đe dọa và các ràng buộc vận hành thành một bộ tài liệu hướng dẫn cụ thể cho đội ngũ thực thi.

Chapter 4 identify and Analyze Compliance Requirements

Regulations and Compliance

Dưới đây là nội dung chi tiết phần đầu của mục **Regulations and Compliance (Quy định và Tuân thủ)** trong Chương 4, từ lúc bắt đầu cho đến trước phần Tiêu chuẩn bảo mật (Security Standards):

1. Vai trò của Quy định và Tuân thủ

Các quy định và yêu cầu tuân thủ là yếu tố thúc đẩy nhiều hoạt động trong doanh nghiệp. Nguyên nhân chính là do việc **không tuân thủ** các quy tắc và quy định có thể dẫn đến các hình phạt tài chính trực tiếp và đôi khi rất lớn. Ngoài ra, thất bại trong tuân thủ còn gây ra các chi phí bổ sung như:

- Bị giám sát chặt chẽ hơn từ các cơ quan chức năng.
- Phải đối mặt với nhiều quy định khắt khe hơn trong tương lai.
- Gây ảnh hưởng xấu đến uy tín công chúng (bad publicity).

Vì phần mềm là động lực chính của nhiều quy trình kinh doanh, một chuyên gia CSSLP cần hiểu cơ sở của các quy tắc này và cách chúng ảnh hưởng đến nỗ lực phát triển của doanh nghiệp.

2. Tuân thủ so với Bảo mật

Có một quan điểm phổ biến rằng **tuân thủ (compliance) không đồng nghĩa với bảo mật (security)**. Một hệ thống có thể đáp ứng đầy đủ các yêu cầu tuân thủ nhưng vẫn không an toàn.

- Dưới góc độ quản trị rủi ro, cả bảo mật và tuân thủ đều là các bài toán quản lý rủi ro.
- Khi kết hợp tất cả các yếu tố này, doanh nghiệp sẽ có cách tiếp cận "mọi mối nguy hại" (all hazards), giúp ban quản lý cấp cao kiểm soát rủi ro tồn dư từ mọi nguồn rủi ro.

3. Nguồn gốc và Hình phạt

- **Nguồn gốc:** Các quy định có thể đến từ nhiều nguồn khác nhau, bao gồm các nhóm ngành nghề, hiệp hội thương mại và các cơ quan chính phủ.
- **Hình phạt:** Các hình phạt cho việc không tuân thủ thay đổi tùy theo mức độ nghiêm trọng của vi phạm hoặc các yếu tố chính trị.
- **Lộ trình thực hiện:** Thông thường, các quy tắc này được công bố trước một thời gian dài để doanh nghiệp có thể lập kế hoạch kiểm soát và tối ưu hóa các lựa chọn quản lý rủi ro.

4. Sự chồng chéo của các quy định

Một doanh nghiệp thường phải chịu sự điều chỉnh của nhiều bộ quy định khác nhau, đôi khi chúng còn chồng chéo lên nhau.

- Nhiều dự án phát triển phần mềm có thể chịu tác động từ nhiều quy định cùng lúc.
- **Ví dụ:** Một ứng dụng xử lý cả thông tin y tế và thông tin thanh toán sẽ phải tuân thủ đồng thời cả **PCI DSS** (Thẻ thanh toán) và **HIPAA** (Y tế). Việc ánh xạ các yêu cầu khác nhau này vào từng luồng dữ liệu cụ thể mà chúng tác động là rất quan trọng.

5. Lưu ý quan trọng cho chuyên gia CSSLP

Điều quan trọng là không được nhầm lẫn giữa **chức năng bảo mật (security functionality)** với **mục tiêu của phát triển phần mềm an toàn**. Các chức năng bảo mật được thúc đẩy

bởi các yêu cầu tuân thủ là cần thiết, nhưng mục tiêu cốt lõi của vòng đời phát triển phần mềm an toàn (SDL) là **giảm số lượng và mức độ nghiêm trọng của các lỗ hổng** trong phần mềm.

Phần **Security Standards (Tiêu chuẩn Bảo mật)** trong Chương 4 tập trung vào vai trò của các khuôn khổ kỹ thuật và thực hành tốt nhất trong việc đảm bảo an ninh và khả năng tương tác. Dưới đây là nội dung chi tiết:

1. Định nghĩa và Mục đích của Tiêu chuẩn

- **Khái niệm**: Tiêu chuẩn là một mức độ hoạt động đã được xác định, có thể đo lường và giám sát việc tuân thủ bởi một bên thứ ba.
- **Vai trò tương tác**: Tiêu chuẩn giúp các tổ chức khác nhau tương tác với nhau theo một cách thức đã biết và có ý nghĩa, đồng thời tạo cơ sở để so sánh năng lực giữa các tổ chức.
- **Thúc đẩy thực hành tốt nhất**: Quá trình bảo mật trong doanh nghiệp được tăng cường thông qua việc sử dụng các tiêu chuẩn cho phép triển khai các hoạt động gắn liền với các thực hành tốt nhất (best practices).

2. Tiêu chuẩn và Khả năng tương tác (Interoperability)

- Trong thiết kế và phát triển phần mềm, tiêu chuẩn đóng vai trò thiết yếu khi cần kết nối các mô-đun từ nhiều nguồn khác nhau.
- **Ví dụ**: Tiêu chuẩn **WS-security** cung cấp một phương thức truyền thông an toàn giữa các dịch vụ web (web services), đảm bảo các thành phần phần mềm khác nhau có thể "nói chuyện" với nhau một cách an toàn.

3. Nguồn gốc của các Tiêu chuẩn

Các tiêu chuẩn bảo mật đến từ nhiều nguồn đa dạng, bao gồm:

- Các cơ quan tiêu chuẩn hóa quốc tế và quốc gia.
- Các nhóm ngành nghề và hiệp hội thương mại.

4. Các tổ chức tiêu chuẩn hóa tiêu biểu

Tài liệu đi sâu vào phân tích các tiêu chuẩn từ hai tổ chức quan trọng nhất:

- **ISO (Tổ chức Tiêu chuẩn hóa Quốc tế)**: Phát triển và công bố các tiêu chuẩn quốc tế như dòng **ISO 2700X** (quản lý an toàn thông tin), **ISO/IEC 15408** (Common Criteria) để đánh giá thuộc tính bảo mật của sản phẩm, và các tiêu chuẩn về chất lượng phần mềm như **ISO/IEC 9126**,,,.
- **NIST (Viện Tiêu chuẩn và Công nghệ Quốc gia Hoa Kỳ)**: Phát triển các tiêu chuẩn bắt buộc đối với các cơ quan liên bang (FIPS) và các hướng dẫn phổ biến cho ngành công nghiệp thông qua dòng **Special Publication (SP) 800** (ví dụ: SP 800-53 về kiểm soát bảo mật và quyền riêng tư),,.

Tóm lại, tiêu chuẩn bảo mật không chỉ là thước đo để kiểm tra tính tuân thủ mà còn là **ngôn ngữ chung** giúp các hệ thống phần mềm phức tạp có thể tích hợp và vận hành an toàn trong môi trường doanh nghiệp.

Dưới đây là nội dung chi tiết về phần **ISO (Tổ chức Tiêu chuẩn hóa Quốc tế)** trong Chương 4 của tài liệu:

1. Tổng quan về ISO

ISO là Tổ chức Tiêu chuẩn hóa Quốc tế, chuyên phát triển và công bố các tiêu chuẩn quốc tế. Các tiêu chuẩn này được xem xét theo **chu kỳ 5 năm** để đảm bảo chúng luôn phù hợp với sự thay đổi của công nghệ và các mối đe dọa. Các lĩnh vực liên quan đến chuyên gia bảo mật thường nằm trong danh mục của **JTC 1 – Công nghệ thông tin**, cụ thể là các tiểu ban:

- **Tiểu ban 7:** Kỹ thuật Phần mềm và Hệ thống.
- **Tiểu ban 27:** Các kỹ thuật Bảo mật CNTT.

2. Dòng tiêu chuẩn ISO 2700X

Dòng tiêu chuẩn này đóng vai trò đối với an toàn thông tin tương tự như dòng ISO 900X đối với quản lý chất lượng. Đây là một "gia đình" tiêu chuẩn đang phát triển với hơn 20 tiêu chuẩn hiện có, được thiết kế để áp dụng cho mọi loại hình và quy mô tổ chức. Các thành phần chính bao gồm:

- **Định nghĩa từ vựng liên quan.**

- **Quy tắc thực hành (Code of practice).**
- **Hướng dẫn triển khai hệ thống quản lý.**
- **Các phép đo (metrics) và nguyên tắc quản lý rủi ro.**

3. ISO/IEC 15408 – Common Criteria (Tiêu chuẩn Chung)

Đây là một khuôn khổ cho phép các nhà cung cấp đưa ra các tuyên bố chính xác về thuộc tính bảo mật của sản phẩm và cho phép các phòng thí nghiệm kiểm thử đánh giá các tuyên bố đó. Các thuật ngữ quan trọng trong khuôn khổ này gồm:

- **Target of Evaluation (TOE):** Sản phẩm hoặc hệ thống đang được đánh giá.
- **Security Target (ST):** Các đặc tính bảo mật liên quan đến một TOE cụ thể.
- **Protection Profile (PP):** Tập hợp các yêu cầu bảo mật cho một **loại sản phẩm** (ví dụ: hồ sơ bảo mật riêng cho tường lửa, hệ điều hành).
- **Evaluation Assurance Level (EAL):** Mức độ đảm bảo đánh giá gồm 7 cấp độ (từ EAL 1 cơ bản đến EAL 7 toàn diện). **Lưu ý:** Chỉ số EAL cao hơn có nghĩa là độ tin cậy vào việc sản phẩm đáp ứng tuyên bố bảo mật cao hơn, chứ không nhất thiết là sản phẩm đó có tính bảo mật cao hơn.

4. ISO/IEC 9126: Chất lượng sản phẩm phần mềm

Tiêu chuẩn quốc tế này cung cấp một khuôn khổ để đánh giá chất lượng sản phẩm phần mềm thông qua các phép đo nội bộ và bên ngoài trong quá trình vận hành. Nó định nghĩa **6 đặc tính chất lượng** để đo lường phần mềm:

1. **Tính chức năng (Functionality)**.
2. **Tính tin cậy (Reliability)**.
3. **Tính khả dụng (Usability)**.
4. **Tính hiệu quả (Efficiency)**.
5. **Tính bảo trì (Maintainability)**.
6. **Tính di động (Portability)**.

5. Các tiêu chuẩn quy trình quan trọng khác

- **ISO/IEC/IEEE 12207:** Thiết lập một bộ quy trình bao quát **vòng đời của phần mềm**, cung cấp cấu trúc và từ vựng chung cho tất cả các bên tham gia phát triển.
- **ISO/IEC 33001 (SPICE):** Được sử dụng để đánh giá năng lực và cải tiến quy trình phát triển phần mềm. Tiêu chuẩn này định nghĩa thang đo năng lực từ **mức 0 (Quy trình chưa hoàn thiện)** đến **mức 5 (Quy trình tối ưu hóa)**.

Ngoài ra, tài liệu còn liệt kê một danh sách dài các tiêu chuẩn ISO nổi bật khác hỗ trợ cho việc đảm bảo chất lượng, đo lường và quản lý rủi ro an toàn thông tin.

Dưới đây là nội dung chi tiết về phần **NIST (Viện Tiêu chuẩn và Công nghệ Quốc gia Hoa Kỳ)** trong Chương 4 của tài liệu:

1. Tổng quan về NIST

NIST là một cơ quan liên bang thuộc Bộ Thương mại Hoa Kỳ, có nhiệm vụ phối hợp với ngành công nghiệp để phát triển công nghệ, đo lường và các tiêu chuẩn nhằm hỗ trợ nền kinh tế.

- **Bộ phận Bảo mật Máy tính (Computer Security Division):** Là đơn vị trực thuộc NIST chịu trách nhiệm về các vấn đề an ninh máy tính, bao gồm việc đảm bảo tuân thủ **Đạo luật Quản lý An ninh Thông tin Liên bang (FISMA)**.
- **Các loại tài liệu chính:** NIST phát triển và công bố hai loại tài liệu quan trọng nhất là **FIPS** và dòng **Special Publication (SP) 800**. Ngoài ra, họ còn xuất bản các bản tin bảo mật (Security Bulletins) khoảng 6 lần/năm và các báo cáo nội bộ/liên cơ quan (NISTIRs) về nghiên cứu kỹ thuật.

2. Tiêu chuẩn Xử lý Thông tin Liên bang (FIPS)

- **Tính bắt buộc:** Các tiêu chuẩn FIPS là bộ yêu cầu **bắt buộc** đối với các cơ quan liên bang và các nhà thầu cụ thể của chính phủ.
- **Phạm vi ảnh hưởng:** Mặc dù số lượng tiêu chuẩn FIPS không nhiều, nhưng chúng có thẩm quyền và phạm vi tác động cực kỳ rộng lớn.

- **Quy trình miễn trừ:** Kể từ khi đạo luật FISMA được thông qua, tất cả các khía cạnh của FIPS đều trở thành bắt buộc và quy trình xin miễn trừ (waiver process) trước đây không còn được áp dụng.

3. Dòng Ấn bản Đặc biệt NIST SP 800 (NIST SP 800 Series)

Đây là bộ tài liệu của NIST được **ngành công nghiệp sử dụng phổ biến nhất**.

- **Mục đích:** Truyền đạt các kết quả nghiên cứu liên quan và cung cấp các hướng dẫn về bảo mật hệ thống thông tin.
- **Nội dung:** Các tài liệu SP 800 trải dài trên nhiều lĩnh vực, từ mô tả các giao thức mật mã, yêu cầu bảo mật cho các thành phần hệ thống, đến các yếu tố của khung quản lý rủi ro (Risk Management Framework) trong quản trị an toàn thông tin.

4. Các ấn bản NIST nổi bật dành cho CSSLP

Tài liệu liệt kê một số ấn bản quan trọng mà các chuyên gia cần nắm vững:

- **FIPS 199:** Tiêu chuẩn phân loại an ninh cho thông tin và hệ thống thông tin liên bang.
- **FIPS 200:** Các yêu cầu bảo mật tối thiểu cho hệ thống thông tin liên bang.
- **FIPS 140 series:** Các yêu cầu an ninh đối với các mô-đun mật mã.
- **FIPS 197:** Tiêu chuẩn Mã hóa Nâng cao (AES).
- **SP 800-53:** Các biện pháp kiểm soát bảo mật và quyền riêng tư cho các tổ chức và hệ thống thông tin.
- **SP 800-30:** Hướng dẫn thực hiện đánh giá rủi ro.
- **SP 800-63:** Hướng dẫn về danh tính số (Digital Identity Guidelines).
- **SP 800-100:** Sổ tay an ninh thông tin dành cho các nhà quản lý.

Tóm lại, NIST cung cấp một hệ thống tài liệu hướng dẫn và tiêu chuẩn kỹ thuật vô cùng chi tiết, đóng vai trò là nền tảng cho việc thiết lập các chương trình bảo mật không chỉ trong chính phủ mà còn trong toàn bộ khu vực tư nhân.

Dưới đây là nội dung chi tiết về **FISMA (Đạo luật Quản lý An ninh Thông tin Liên bang)** từ Chương 4 của tài liệu:

1. Tổng quan về FISMA

- **Định nghĩa:** Đạo luật Quản lý An ninh Thông tin Liên bang năm 2002 (FISMA) là một luật liên bang yêu cầu mỗi cơ quan liên bang phải triển khai một chương trình an ninh thông tin trên toàn cơ quan.
- **Vai trò của NIST:** Viện Tiêu chuẩn và Công nghệ Quốc gia (NIST) được chỉ định là cơ quan phát triển các hướng dẫn triển khai cho đạo luật này thông qua việc công bố một **Khung quản lý rủi ro (RMF)** để tuân thủ.

2. Các mục tiêu tuân thủ ban đầu

Trong khuôn khổ tuân thủ ban đầu, các mục tiêu sau đây được Văn phòng Tổng Thanh tra (Inspector General) chấm điểm hàng năm:

- **Kiểm kê hệ thống:** Lập danh mục tất cả các hệ thống thông tin.
- **Phân loại thông tin và hệ thống:** Dựa trên mức độ rủi ro.
- **Kiểm soát an ninh:** Thiết lập các biện pháp kiểm soát.
- **Chứng nhận và công nhận hệ thống:** Bao gồm đánh giá rủi ro và các kế hoạch an ninh hệ thống.
- **Đào tạo:** Đào tạo nhận thức về an ninh cho nhân sự.

3. Sự phát triển của chương trình

- **Tự động hóa:** NIST đã bổ sung Chương trình Tự động hóa An ninh Thông tin và Giao thức Tự động hóa Nội dung An ninh (SCAP).
- **Giám sát liên tục:** Hiện nay, tất cả các hệ thống được công nhận phải có một bộ kiểm soát an ninh được giám sát để cung cấp khả năng giám sát liên tục (continuous monitoring).
- **Phạm vi:** FISMA là bắt buộc đối với các cơ quan liên bang và mở rộng cho cả các **nhà thầu** triển khai hoặc vận hành hệ thống thông tin liên bang.

- **Hiệu quả:** Hiệu quả của FISMA liên quan trực tiếp đến sự nghiêm túc của ban quản lý cấp cao. Nó có tác dụng giảm rủi ro toàn hệ thống tốt nhất khi được coi là một công cụ kiểm soát hơn là một bảng kiểm (checklist) tuân thủ thuần túy.

4. Quy trình Khung Quản lý Rủi ro (RMF) 6 bước

NIST đã công bố các ấn bản chi tiết (đặc biệt là NIST SP 800-39) về quy trình 6 bước để tạo ra một RMF có cấu trúc nhưng linh hoạt:

1. **Phân loại (Categorize):** Phân loại hệ thống thông tin.
2. **Lựa chọn (Select):** Chọn các biện pháp kiểm soát an ninh.
3. **Triển khai (Implement):** Thực hiện các biện pháp kiểm soát an ninh.
4. **Đánh giá (Assess):** Đánh giá hiệu quả của các biện pháp kiểm soát.
5. **Cấp phép (Authorize):** Cấp phép vận hành hệ thống thông tin.
6. **Giám sát (Monitor):** Giám sát các biện pháp kiểm soát an ninh một cách liên tục.

Lưu ý cho chuyên gia CSSLP: Các chuyên gia CSSLP cần phải tích hợp công việc phát triển phần mềm của mình vào khung công tác này khi làm việc trong các tổ chức vận hành dưới mô hình RMF.

Dưới đây là nội dung chi tiết về ba bộ luật quan trọng trong mục **Identify and Analyze Compliance Requirements** (Xác định và Phân tích các yêu cầu tuân thủ) của Chương 4:

1. Sarbanes-Oxley (Đạo luật Sarbanes-Oxley - SOX)

Đạo luật Sarbanes-Oxley năm 2002 ra đời như một phản ứng trước nhiều vụ bê bối kế toán và tập đoàn lớn, gây thiệt hại hàng tỷ đô la cho các nhà đầu tư và làm lung lay niềm tin của công chúng vào thị trường chứng khoán.

- **Thành phần chính về an ninh thông tin:** Mặc dù bao gồm nhiều phần, yếu tố quan trọng nhất liên quan đến an ninh thông tin là **Mục 404 (Section 404)**.
- **Yêu cầu cốt lõi:** Mục này bắt buộc áp dụng một mức độ cụ thể cho các biện pháp **kiểm soát nội bộ (internal control measures)**.

- **Mục tiêu đối với phần mềm:** Các hệ thống thông tin được sử dụng cho kế toán tài chính phải có các cơ chế kiểm soát bảo mật về **tính toàn vẹn (integrity)** để mọi người có thể tin tưởng vào các con số mà hệ thống báo cáo. Dù bị chỉ trích vì chi phí triển khai cao, đây vẫn là luật hiện hành mà các hệ thống báo cáo tài chính bắt buộc phải tuân thủ.

2. Gramm-Leach-Bliley (Đạo luật Gramm-Leach-Bliley - GLBA)

Đạo luật Hiện đại hóa Tài chính năm 1999, hay GLBA, chứa các yếu tố được thiết kế để bảo vệ **thông tin tài chính cá nhân (PFI)** của người tiêu dùng. Từ góc độ phần mềm, đạo luật này quy định các quy tắc về thu thập, xử lý, lưu trữ và hủy bỏ PFI. Có ba quy tắc chính cần lưu ý:

- **Quy tắc Quyền riêng tư Tài chính (Financial Privacy Rule):** Quản lý việc thu thập và tiết lộ PFI, áp dụng cho cả các công ty không thuộc lĩnh vực tài chính.
- **Quy tắc Bảo vệ (Safeguards Rule):** Áp dụng cho các tổ chức tài chính, bao quát việc thiết kế, triển khai và bảo trì các biện pháp bảo vệ được triển khai để bảo vệ PFI.
- **Các biện pháp bảo vệ chống giả mạo (Pretexting Protections):** Giải quyết việc sử dụng các thủ đoạn giả mạo (pretexting) để thu thập PFI trái phép.

3. HIPAA and HITECH (Đạo luật liên quan đến thông tin y tế)

Trong khi GLBA xử lý thông tin tài chính (PFI), các đạo luật này tập trung vào **thông tin sức khỏe cá nhân (PHI)**, loại thông tin có giá trị rất lớn đối với các tổ chức tội phạm.

- **HIPAA (Đạo luật về Khả năng di chuyển và Trách nhiệm Giải trình Bảo hiểm Y tế):** Được ban hành năm 1996, các quy định về quyền riêng tư ban đầu của HIPAA chưa được chuẩn bị kỹ cho xu hướng chuyển dịch sang hồ sơ điện tử của ngành y tế.
- **HITECH (Đạo luật Công nghệ Thông tin Y tế vì Sức khỏe Kinh tế và Lâm sàng):** Là một phần của Đạo luật Phục hồi và Tái đầu tư Hoa Kỳ năm 2009 (ARRA), được thiết kế để **tăng cường các quy định về quyền riêng tư** cho các hồ sơ thông tin sức khỏe cá nhân điện tử.

- **Giá trị của dữ liệu PHI:** Hồ sơ y tế là nguồn thông tin giá trị cho tội phạm mạng vì chúng chứa cả thông tin bảo hiểm, trách nhiệm tài chính (bao gồm thẻ tín dụng) và đủ dữ liệu định danh cá nhân (PII) để thực hiện hành vi đánh cắp danh tính.

Việc hiểu rõ các nguồn yêu cầu bảo mật này là cực kỳ quan trọng đối với chuyên gia CSSLP để tích hợp chúng vào quá trình phát triển phần mềm ngay từ đầu.

Dưới đây là nội dung chi tiết về **Tiêu chuẩn Bảo mật Dữ liệu Ngành Thẻ Thanh toán (Payment Card Industry Data Security Standard - PCI DSS)** từ Chương 4 của tài liệu:

1. Tổng quan về PCI

- **Định nghĩa:** PCI viết tắt của Payment Card Industry, là một nhóm ngành được thành lập để tạo ra, quản lý và thực thi các quy định liên quan đến việc bảo vệ dữ liệu chủ thẻ.
- **Ba tiêu chuẩn chính:** Nhóm này duy trì ba tiêu chuẩn cốt lõi để bảo vệ dữ liệu:
    1. **PCI DSS:** Tiêu chuẩn Bảo mật Dữ liệu.
    2. **PA DSS:** Tiêu chuẩn Bảo mật Dữ liệu Ứng dụng Thanh toán.
    3. **PTS:** Bảo mật Giao dịch mã PIN.

2. Tiêu chuẩn PCI DSS

- **Bản chất:** Đây là tài liệu quản lý chi tiết các yêu cầu hợp đồng đối với các thành viên chấp nhận và xử lý thẻ ngân hàng.
- **Nội dung bao quát:** Tiêu chuẩn này bao gồm các yêu cầu về quản lý an ninh, chính sách và quy trình, kiến trúc mạng, thiết kế phần mềm và các biện pháp bảo vệ quan trọng khác cho tất cả các hệ thống liên quan đến việc xử lý và lưu trữ dữ liệu chủ thẻ.
- **Cấu trúc:** Được sắp xếp thành **6 nhóm mục tiêu kiểm soát** với **12 yêu cầu cấp cao** Dưới mỗi yêu cầu là một số lượng lớn các yêu cầu phụ và quy trình kiểm tra được sử dụng để xác định nền tảng bảo mật cơ sở.

3. Tiêu chuẩn PA DSS

- **Mục tiêu:** Đây là bộ yêu cầu dành cho các **nhà cung cấp phần mềm** để xác nhận rằng một ứng dụng thanh toán tuân thủ các yêu cầu liên quan của PCI DSS.
- **Tầm quan trọng đối với phần mềm:** Việc tuân thủ PA DSS là tín hiệu cho thấy phần mềm được thiết kế đúng cách để xử lý dữ liệu thẻ.
- **Lưu ý:** Việc chỉ sử dụng phần mềm tuân thủ PA DSS là chưa đủ để đạt được tính tuân thủ hoàn toàn, vì doanh nghiệp vẫn phải thực hiện các yêu cầu không liên quan đến phần mềm trong PCI DSS.

4. Tiêu chuẩn PTS

- **Trọng tâm:** Tập trung vào việc bảo vệ **mã PIN**, một trong những thành phần quan trọng nhất của dữ liệu chủ thẻ.
- **Đối tượng áp dụng:** Phần lớn tiêu chuẩn này áp dụng cho các thiết bị phần cứng được gọi là **thiết bị nhập mã PIN (PEDs)**.

5. Tầm quan trọng và Hình phạt

- **Tính bắt buộc:** Các tiêu chuẩn PCI là yêu cầu bắt buộc theo hợp đồng. Nếu một doanh nghiệp chấp nhận thẻ thanh toán, lưu trữ dữ liệu thẻ hoặc tạo ra các sản phẩm liên quan đến thẻ thanh toán, họ bắt buộc phải tuân thủ.
- **Rủi ro tài chính:** Việc không tuân thủ có thể dẫn đến các **hình phạt tài chính nghiêm trọng**. Do đó, PCI DSS thường được xếp hàng đầu trong các nỗ lực quản lý rủi ro và tuân thủ của doanh nghiệp.

6. Thông tin bổ sung từ các chương khác

- **Yêu cầu về WAF hoặc Code Review:** Theo PCI DSS, các ứng dụng web bắt buộc phải có tường lửa ứng dụng web (WAF) giữa máy chủ và người dùng **hoặc** phải thực hiện đánh giá mã nguồn ứng dụng (code reviews).
- **Yêu cầu về ghi nhật ký:** PCI DSS là một trong những chương trình tuân thủ yêu cầu cụ thể về việc ghi và quản lý nhật ký (logging).

Dưới đây là nội dung chi tiết cho các phần bạn yêu cầu trong Chương 4 của tài liệu:

1. Các quy định khác (Other Regulations)

Bên cạnh các bộ luật lớn, còn có rất nhiều quy định ít được biết đến nhưng cũng không kém phần quan trọng:

- **Quy định của FFIEC:** Các dịch vụ ngân hàng qua Internet được điều chỉnh bởi các quy tắc từ **Hội đồng Kiểm tra các Tổ chức Tài chính Liên bang (FFIEC)**.
- **Yêu cầu xác thực:** Các quy định hiện hành của FFIEC bắt buộc việc xác thực phải mang tính chất **đa yếu tố (multifactor)** ở mức tối thiểu.
- **Tác động đến thiết kế:** Bất kỳ hệ thống nào được thiết kế để sử dụng trong môi trường này đều phải đưa yêu cầu xác thực đa yếu tố vào như một yêu cầu bắt buộc ngay từ đầu.

2. Các vấn đề pháp lý (Legal Issues)

Các vấn đề pháp lý tạo ra khung hành vi và môi trường làm việc cho doanh nghiệp:

- **Giải quyết tranh chấp:** Hệ thống pháp luật đóng vai trò là phương thức để giải quyết các tranh chấp phát sinh giữa các bên.
- **Lộ trình hành vi:** Theo thời gian, một hệ thống các luật và quy định đã được tạo ra để điều chỉnh các hoạt động, cung cấp một **lộ trình cho hành vi (roadmap for behavior)** giữa các bên.

3. Sở hữu trí tuệ (Intellectual Property)

Sở hữu trí tuệ là thuật ngữ pháp lý công nhận rằng những sáng tạo từ trí tuệ có thể là tài sản và người tạo ra chúng có quyền kiểm soát độc quyền. Các hình thức bảo hộ phổ biến bao gồm:

- **Bằng sáng chế (Patents):** Quyền độc quyền do chính phủ cấp cho nhà phát minh trong một khoảng thời gian nhất định để đổi lấy việc công bố sáng chế. Sáng chế phải đảm bảo tính **mới, hữu ích và không hiển nhiên**. Luật bằng sáng chế cho phần mềm vẫn là một chủ đề gây tranh cãi gay gắt, nhưng nó có thể bảo hộ các thuật toán và phương pháp cơ bản độc lập với ngôn ngữ lập trình.

- **Bản quyền (Copyrights):** Bảo hộ cho bất kỳ hình thức thể hiện ý tưởng hoặc thông tin nào mang tính thực chất và riêng biệt. Trong phần mềm, bản quyền bảo vệ mã nguồn như một tác phẩm văn học, ngăn chặn việc sao chép trực tiếp nhưng không ngăn cản người khác tự viết phiên bản riêng của họ.
- **Nhãn hiệu (Trademarks):** Bảo vệ một chất lượng có thể nhận diện được gắn liền với một sản phẩm hoặc công ty (như tên thương hiệu Amazon.com) nhằm xây dựng sự liên tưởng thương hiệu và cấm người khác sao chép.
- **Bí mật kinh doanh (Trade Secrets):** Cung cấp sự bảo hộ tối ưu dựa trên thời gian bằng cách giữ bí mật thông tin (ví dụ: công thức Coca-Cola). Bí mật kinh doanh khó áp dụng cho phần mềm vì việc phân phối phần mềm có thể cho phép người dùng cuối tiếp cận nhiều thông tin thông qua dịch ngược.
- **Bảo hành (Warranties):** Là lời hứa (ngầm định hoặc theo hợp đồng) rằng sản phẩm sẽ hoạt động như mong đợi. Đối với phần mềm, hầu hết các nhà cung cấp thường từ chối bảo hành bằng các cụm từ như "as-is" (nguyên trạng) hoặc "không chịu trách nhiệm với bất kỳ lỗi nào phát sinh từ việc sử dụng".

Việc hiểu rõ các yếu tố này giúp chuyên gia CSSLP đảm bảo phần mềm không chỉ tuân thủ pháp luật mà còn bảo vệ được giá trị kinh tế và trí tuệ của doanh nghiệp.

Dưới đây là nội dung chi tiết về **Bằng sáng chế (Patents)** và **Bản quyền (Copyrights)** từ Chương 4 của tài liệu nguồn:

1. Bằng sáng chế (Patents)

Bằng sáng chế cung cấp quyền độc quyền do chính phủ cấp cho nhà phát minh trong một khoảng thời gian nhất định để đổi lấy việc công bố công khai sáng chế đó.

- **Yêu cầu cơ bản (tại Hoa Kỳ):** Để được bảo hộ, một sáng chế phải đáp ứng ba tiêu chí: **mới (new), hữu ích (useful) và không hiển nhiên (nonobvious)**. Sáng chế có thể là một quy trình, máy móc, vật phẩm sản xuất hoặc thành phần vật chất.

- **Quyền hạn của chủ sở hữu:** Bằng sáng chế cho phép chủ sở hữu ngăn cản người khác sử dụng sáng chế đã đăng ký, **ngay cả khi bên kia tuyên bố họ tự phát triển độc lập** mà không hề sao chép.
- **Vấn đề về phần mềm:**
    - Luật bằng sáng chế cho phần mềm hiện vẫn đang gây tranh cãi gay gắt.
    - Tại Hoa Kỳ, bằng sáng chế thường bị từ chối đối với các "ý tưởng trừu tượng". Tại Châu Âu, các chương trình máy tính thuần túy thường bị loại trừ khỏi khả năng cấp bằng sáng chế.
    - Tuy nhiên, bằng sáng chế có thể bảo hộ các **thuật toán và phương pháp cơ bản** ẩn sau phần mềm, cũng như chức năng mà phần mềm đó hướng tới, bất kể ngôn ngữ lập trình hay mã nguồn cụ thể là gì.
- **Lưu ý khi đăng ký:** Đơn đăng ký bằng sáng chế là tài liệu pháp lý chuyên sâu đòi hỏi nhiều nguồn lực. Việc nộp đơn phải được thực hiện **trước khi công bố** sáng chế ra công chúng.

2. Bản quyền (Copyrights)

Bản quyền là hình thức bảo hộ sở hữu trí tuệ áp dụng cho bất kỳ hình thức thể hiện ý tưởng hoặc thông tin nào mang tính **thực chất và riêng biệt**.

- **Mục tiêu bảo hộ:** Cung cấp cho tác giả các quyền độc quyền đối với tác phẩm gốc, bao gồm quyền được đứng tên (ghi nhận tác giả), quyền quyết định ai có thể chuyển thể tác phẩm, quyền biểu diễn và quyền hưởng lợi tài chính.
- **Quy định quốc tế:** Bản quyền được quản lý quốc tế thông qua **Công ước Berne**, yêu cầu các quốc gia ký kết phải công nhận bản quyền của tác giả từ các quốc gia thành viên khác giống như tác giả trong nước mình.
- **Bảo hộ phần mềm:**
    - Dưới Công ước Berne, phần mềm được bảo hộ như một **tác phẩm văn học (works of literature)**.

    - Bản quyền cho phép tác giả ngăn chặn người khác thực hiện hành vi **sao chép trực tiếp** (copying) một phần hoặc toàn bộ phiên bản phần mềm.
    - **Giới hạn:** Bản quyền **không ngăn cản** các nhà phát triển khác tự viết các phiên bản riêng của họ một cách độc lập dựa trên cùng một ý tưởng hoặc chức năng.
- **Thực thi pháp luật:** So với bằng sáng chế, việc đăng ký bản quyền tương đối đơn giản và ít tốn kém hơn. Luật bản quyền nghiêm cấm việc phân phối bản sao phần mềm không phép, với các hình phạt có thể lên tới 100.000 USD và 5 năm tù cho mỗi lần vi phạm.
- **Điểm lưu ý trong ngành:** Một thực hành phổ biến là công bố các **đặc tả giao diện (interface specifications)** để các chương trình khác có thể kết nối đúng cách mà không vi phạm bản quyền.

Dưới đây là nội dung chi tiết về **Trademarks**, **Trade Secrets** và **Warranties** từ Chương 4 của tài liệu:

Trademarks (Nhãn hiệu)

- **Định nghĩa và Mục đích:** Nhãn hiệu là một chất lượng có thể nhận diện được gắn liền với một sản phẩm hoặc một công ty. Bản chất của nhãn hiệu là xây dựng sự liên tưởng thương hiệu, do đó việc người khác sao chép nhãn hiệu bị nghiêm cấm.
- **Phân loại:** Nhãn hiệu có thể dựa trên thông luật (common law) hoặc đã được đăng ký chính thức. Việc đăng ký nhãn hiệu với chính phủ cung cấp sự bảo vệ pháp lý mạnh mẽ hơn và nhiều phương án để khôi phục quyền lợi hơn.
- **Quản lý quốc tế:** Ở phạm vi quốc tế, nhãn hiệu được quản lý thông qua Tổ chức Sở hữu Trí tuệ Thế giới, sử dụng các giao thức được phát triển tại Madrid, được gọi là Hệ thống Madrid.
- **Ứng dụng thực tế:** Các tên gọi như Amazon.com thường được đăng ký nhãn hiệu để bảo vệ hình ảnh của công ty. Tuy nhiên, các thuật ngữ phổ biến (common terms) hoặc mang tính mô tả đơn thuần sẽ không đủ điều kiện để được bảo hộ nhãn hiệu.

- **Duy trì nhãn hiệu:** Chủ sở hữu phải bảo vệ nhãn hiệu của mình khỏi việc sử dụng chung chung (generic use) không gắn liền với sản phẩm, vì họ có thể mất quyền sở hữu nếu nhãn hiệu đó trở thành một thuật ngữ phổ biến.

Trade Secrets (Bí mật kinh doanh)

- **Đặc điểm bảo hộ:** Bí mật kinh doanh cung cấp hình thức bảo hộ tối ưu dựa trên thời gian cho sở hữu trí tuệ. Để được pháp luật bảo vệ, doanh nghiệp bắt buộc phải giữ kín bí mật đó hoặc ít nhất là thực hiện những nỗ lực hợp lý để giữ bí mật.
- **Rủi ro từ phát hiện độc lập:** Một vấn đề quan trọng là nếu một bên khác độc lập tìm ra cùng một công thức hoặc phương pháp, chủ sở hữu bí mật kinh doanh ban đầu sẽ không có quyền khiếu nại hay can thiệp pháp lý.
- **Khó khăn đối với phần mềm:** Bí mật kinh doanh rất khó áp dụng hiệu quả cho phần mềm vì việc phân phối (ngay cả ở dạng đã biên dịch) vẫn cho phép người dùng cuối tiếp cận được nhiều thông tin thông qua phân tích.
- **Trường hợp ngoại lệ:** Các thuật toán mật mã hoặc mã hạt giống (seeds) có thể được coi là bí mật kinh doanh nếu chúng không được chuyển giao cho khách hàng và được bảo vệ nghiêm ngặt.
- **Hỗ trợ pháp lý liên quan:** Tại Hoa Kỳ, Đạo luật Bản quyền Thiên niên kỷ Kỹ thuật số (DMCA) có các quy định nghiêm cấm việc dịch ngược (reverse-engineering) các biện pháp bảo vệ an ninh, cung cấp một mức độ bảo hộ bổ sung cho các bí mật liên quan đến an ninh.

Warranties (Bảo hành)

- **Bản chất:** Bảo hành đại diện cho một lời hứa (ngầm định hoặc được chỉ định rõ trong hợp đồng) rằng sản phẩm sẽ hoạt động đúng như mong đợi.
- **Mối liên hệ với tính phù hợp sử dụng:** Bảo hành là yếu tố cần thiết nhưng chưa đủ để đảm bảo tính phù hợp cho mục đích sử dụng (fitness for use) của sản phẩm.

- **Bảo hành phần cứng:** Đối với phần cứng máy tính, bảo hành thường nêu rõ sản phẩm sẽ hoạt động theo một đặc tả kỹ thuật nhất định trong một khoảng thời gian và quy định trách nhiệm của nhà cung cấp nếu có lỗi xảy ra.
- **Thực tế trong ngành phần mềm:** Trong khi người dùng cuối thường mong đợi phần mềm sẽ giải quyết được vấn đề của họ (tính phù hợp sử dụng), hầu hết các nhà cung cấp phần mềm đều từ chối cam kết này. Phần lớn các giấy phép phần mềm thường bác bỏ trách nhiệm bằng cách sử dụng các cụm từ như:
    - **"as-is"** (nguyên trạng).
    - **"no warranty as to use"** (không bảo hành cho việc sử dụng).
    - **"no vendor responsibility with respect to any failures resulting from use"** (nhà cung cấp không chịu trách nhiệm đối với bất kỳ lỗi nào phát sinh từ việc sử dụng).

Data Classification

Phần **Data Classification (Phân loại dữ liệu)** trong Chương 4 là một công cụ quản lý rủi ro quan trọng, giúp doanh nghiệp điều chỉnh mức độ bảo vệ và chi phí tương ứng với giá trị của tài sản dữ liệu. Dưới đây là nội dung chi tiết:

1. Tổng quan về Phân loại dữ liệu

- **Mục tiêu:** Giảm chi phí bảo vệ bằng cách căn chỉnh nỗ lực bảo mật với giá trị thực tế của dữ liệu.
- **Cách tiếp cận:** Doanh nghiệp nên thực hiện cái nhìn "lấy dữ liệu làm trung tâm" (data-centric view), xem xét các yêu cầu bảo vệ xuyên suốt toàn bộ vòng đời của dữ liệu để phát hiện các điểm yếu.

2. Các trạng thái của dữ liệu (Data States)

Dữ liệu được phân loại dựa trên trạng thái tĩnh hoặc động tại một thời điểm nhất định:

- **At rest:** Dữ liệu đang được lưu trữ.

- **Being created:** Dữ liệu đang được tạo ra.
- **In transit:** Dữ liệu đang được truyền tải từ địa điểm này sang địa điểm khác.
- **Being changed or deleted:** Dữ liệu đang bị thay đổi hoặc xóa bỏ.
- **Nơi lưu trú:** Cần xem xét dữ liệu nằm trên phương tiện cố định (HDD, CD), phương tiện di động (USB, Cloud) hay trong RAM.

3. Phân loại theo mục đích sử dụng (Data Usage)

- **Internal data:** Dữ liệu khởi tạo hoặc tính toán bên trong ứng dụng.
- **Input data:** Dữ liệu được đọc vào hệ thống.
- **Output data:** Dữ liệu được ghi ra đích đến sau khi xử lý.
- **Security-sensitive data:** Dữ liệu có giá trị cao đối với kẻ tấn công.
- **PII data:** Dữ liệu chứa thông tin định danh cá nhân.
- **Hidden data:** Dữ liệu cần được che giấu bằng các kỹ thuật xáo trộn (obfuscation).

4. Tác động rủi ro của dữ liệu (Data Risk Impact)

Dữ liệu được phân loại dựa trên mức độ nghiêm trọng nếu bị mất hoặc rò rỉ:

- **High (Cao):** Có thể gây hậu quả thảm khốc cho tài sản, vận hành hoặc con người (ví dụ: tổn thất > 1 triệu USD, gây tử vong/thương tật).
- **Medium (Trung bình):** Gây ra những hậu quả nghiêm trọng (ví dụ: tổn thất > 100.000 USD, chấn thương nặng).
- **Low (Thấp):** Hậu quả hạn chế hoặc không đáng kể.
- **Lưu ý:** Các khung tiêu chuẩn như **NIST FIPS 199** và **SP 800-18** cung cấp cơ sở để phân loại tác động dựa trên ba khía cạnh: Bảo mật (Confidentiality), Toàn vẹn (Integrity) và Khả dụng (Availability).

5. Vòng đời và Quyền sở hữu (Lifecycle & Ownership)

- **Vòng đời dữ liệu:** Bao gồm các giai đoạn: Tạo ra (Generation), Sử dụng (Use), Lưu trữ (Storage) và Tiêu hủy (Destruction).

- **Data Owner (Chủ sở hữu dữ liệu):** Là người xác định mức độ truy cập, định nghĩa việc phân loại dữ liệu, lựa chọn các biện pháp kiểm soát bảo mật và đảm bảo chúng hoạt động hiệu quả.
- **Data Custodian (Người giám hộ dữ liệu):** Chịu trách nhiệm triển khai các biện pháp kỹ thuật theo yêu cầu của chủ sở hữu, thực hiện các tác vụ vận hành như sao lưu, lưu kho và tiêu hủy dữ liệu.

6. Nhãn dán và Độ nhạy cảm (Labeling & Sensitivity)

- **Labeling:** Sử dụng các trường siêu dữ liệu (metadata) để gắn nhãn mô tả tầm quan trọng của dữ liệu, đảm bảo dữ liệu được xử lý đúng cách trong thời gian dài.
- **Sensitivity:** Độ nhạy cảm dựa trên mục đích kinh doanh. Nếu một cá nhân có mục đích kinh doanh hợp pháp, họ sẽ được cấp quyền truy cập tương ứng, nếu không thì sẽ bị từ chối.

Dưới đây là nội dung chi tiết và đầy đủ của phần **Impact (Tác động)** thuộc mục Phân loại dữ liệu trong Chương 4:

1. Định nghĩa về Tác động (Impact)

Tác động là một thước đo rộng hơn so với độ nhạy cảm, áp dụng cho hầu hết mọi dữ liệu trong doanh nghiệp. Dữ liệu được phân loại dựa trên **hậu quả mà tổ chức phải gánh chịu** trong trường hợp dữ liệu đó bị mất, bị tiết lộ trái phép hoặc bị thay đổi.

Đây là một chức năng do doanh nghiệp định hướng và mang tính chất định tính cao. Nếu các mức độ tác động (Cao, Trung bình, Thấp) được định nghĩa rõ ràng, việc áp dụng chúng sẽ trở nên đơn giản.

2. Các mức độ tác động tiêu chuẩn

Thông thường, có ba mức độ được sử dụng để phân loại:

- **High (Cao):** Việc tiết lộ hoặc mất dữ liệu có thể gây ra hậu quả thảm khốc hoặc nghiêm trọng cho tài sản, vận hành hoặc con người. Mục tiêu là đặt ngưỡng này đủ cao để chỉ một số lượng nhỏ các thành phần dữ liệu quan trọng nhất được xếp vào nhóm này.

- **Medium/Moderate (Trung bình):** Việc tiết lộ dữ liệu sẽ dẫn đến những hậu quả nghiêm trọng.
- **Low (Thấp):** Dữ liệu có tác động hạn chế, hoặc không có hậu quả đáng kể nếu bị mất hoặc tiết lộ.

Lưu ý quan trọng: Các mức độ tác động có thể được xác định riêng biệt cho từng thuộc tính bảo mật (Bảo mật, Toàn vẹn, Khả dụng). Ví dụ: Một thành phần dữ liệu cụ thể có thể có tác động **Cao** về tính bảo mật và tính toàn vẹn, nhưng lại có tác động **Thấp** về tính khả dụng.

3. Tiêu chí phân loại chi tiết (Bảng 4-1)

Các yếu tố để phân biệt giữa các mức độ tác động dựa trên ba khía cạnh chính: Con người, Khách hàng và Tài chính.

Mức độ tác động	Nhân sự (Personnel)	Khách hàng (Customer)	Tài chính (Financial)*
High (Cao)	Tử vong hoặc thương tật vĩnh viễn	Tác động thảm khốc đến quan hệ khách hàng hiện tại và tương lai	Tổn thất từ 1 triệu USD trở lên
Medium (Trung bình)	Chấn thương nặng, mất khả năng vận hành	Tác động đáng kể đến quan hệ khách hàng hiện tại và tương lai	Tổn thất từ 100.000 USD trở lên
Low (Thấp)	Chấn thương nhẹ	Tác động nhỏ đến quan hệ khách hàng hiện tại và tương lai	Tổn thất ít hơn 100.000 USD

**Giá trị tài chính cụ thể có thể thay đổi tùy theo quy mô của từng doanh nghiệp.*

4. Khung tham chiếu và Tùy chỉnh

- **Khung tham chiếu:** Các tiêu chuẩn **NIST FIPS 199** và **SP 800-18** cung cấp khung công tác để phân loại dữ liệu dựa trên tác động đối với ba khía cạnh tiêu chuẩn: tính bảo mật, tính toàn vẹn và tính khả dụng.

- **Tính tùy chỉnh:** Mỗi tổ chức cần tự định nghĩa các giới hạn tài chính và mức độ nghiêm trọng đối với khách hàng của riêng mình. Một khoản lỗ có thể là thảm họa với công ty nhỏ nhưng có thể chỉ là một sai số nhỏ đối với một tập đoàn đa quốc gia.

Việc phân loại theo tác động giúp đội ngũ phát triển nhận diện được các yêu cầu chuyên biệt liên quan đến các thành phần dữ liệu (như PII hoặc dữ liệu tuân thủ pháp luật) để áp dụng các biện pháp bảo vệ tương ứng.

Privacy

Dưới đây là nội dung chi tiết về phần Privacy (Quyền riêng tư) **trong Chương 4 của tài liệu:**

1. Tổng quan về Quyền riêng tư (Privacy)

Quyền riêng tư là nguyên tắc kiểm soát thông tin cá nhân của một người: chia sẻ với ai, vì mục đích gì, và cách thức thông tin đó được sử dụng hoặc chuyển giao cho các bên khác. Trong phát triển phần mềm, các vấn đề quyền riêng tư thường trở thành vấn đề về **xử lý dữ liệu (data disposition)** — tức là điều gì xảy ra với dữ liệu sau khi nó đã được sử dụng cho giao dịch tức thời.

2. Chính sách quyền riêng tư (Privacy Policy)

Đây là tài liệu cấp cao mô tả các nguyên tắc liên quan đến việc thu thập, lưu trữ, sử dụng và chuyển giao thông tin cá nhân trong phạm vi kinh doanh. Nó đóng vai trò là hướng dẫn cho nhân viên về trách nhiệm đối với thông tin khách hàng.

3. Thông tin định danh cá nhân (PII - Personally Identifiable Information)

PII là thông tin có thể được sử dụng để phân biệt hoặc truy vết danh tính của một cá nhân. Các yếu tố PII phổ biến bao gồm:

- Họ tên đầy đủ (nếu không quá phổ biến).
- Số định danh quốc gia (như SSN).
- Địa chỉ IP (trong một số trường hợp), địa chỉ nhà.
- Biển số xe, số bằng lái xe.
- Hình ảnh khuôn mặt, dấu vân tay, chữ viết tay.
- Số thẻ tín dụng và số tài khoản ngân hàng.

- Ngày sinh, nơi sinh.

4. Thông tin sức khỏe cá nhân (PHI - Personal Health Information)

PHI là tập hợp các thành phần dữ liệu liên quan đến việc chăm sóc sức khỏe của một cá nhân. Loại dữ liệu này là mục tiêu giá trị cao của tội phạm mạng vì chúng chứa đủ thông tin bảo hiểm, trách nhiệm tài chính (bao gồm thẻ tín dụng) và PII để thực hiện hành vi đánh cắp danh tính.

5. Thông báo vi phạm (Breach Notifications)

Khi an ninh thất bại dẫn đến việc dữ liệu cá nhân chưa mã hóa bị các bên không có thẩm quyền thu thập, một vụ vi phạm (breach) được xác định là đã xảy ra. Các luật như SB 1386 của California bắt buộc tổ chức phải thông báo cho cư dân nếu thông tin cá nhân chưa mã hóa của họ có khả năng đã bị xâm phạm.

6. Quy định chung về bảo vệ dữ liệu (GDPR)

GDPR là quy định của EU về bảo vệ quyền cơ bản của con người đối với dữ liệu cá nhân, ngay cả khi dữ liệu đó được chuyển ra ngoài EU. Các yêu cầu quan trọng bao gồm:

- **Chỉ định Cán bộ bảo vệ dữ liệu (DPO):** Phải báo cáo trực tiếp cho cấp quản lý cao nhất và hoạt động độc lập.
- **Sự chấp thuận (Consent):** Phải mang tính khẳng định, rõ ràng, tự nguyện và cụ thể cho từng hoạt động xử lý dữ liệu.
- **Quyền được quên (Right to be Forgotten):** Cá nhân có quyền yêu cầu xóa dữ liệu khi nó không còn cần thiết cho mục đích ban đầu.

7. Đạo luật quyền riêng tư của người tiêu dùng California (CCPA/AB 375)

Tương tự GDPR, CCPA cung cấp cho người tiêu dùng các quyền:

- Biết dữ liệu cá nhân đang được sử dụng như thế nào.
- Biết dữ liệu đã được cung cấp cho ai.
- Phản đối việc bán dữ liệu (Right to object).
- Không bị phân biệt đối xử nếu từ chối bán dữ liệu.

8. Công nghệ tăng cường quyền riêng tư (Privacy-Enhancing Technologies - PET)

PET là các ứng dụng hoặc công cụ hỗ trợ bảo vệ quyền riêng tư:

- **Encryption (Mã hóa):** Lựa chọn hàng đầu để bảo vệ thông tin ở mọi giai đoạn của vòng đời dữ liệu.
- **Data Minimization (Giảm thiểu dữ liệu):** Nguyên tắc chỉ thu thập và giữ lại những gì thực sự cần thiết.
- **Data Masking (Che dấu dữ liệu):** Ẩn dữ liệu bằng cách thay thế giá trị (xáo trộn ký tự, thay thế bằng biểu tượng như "*") mà không làm mất khả năng sử dụng dữ liệu.
- **Tokenization (Mã báo hiệu):** Thay thế dữ liệu bằng một giá trị ngẫu nhiên có tính truy xuất nhưng không có mối liên hệ toán học với dữ liệu gốc.
- **Anonymization (Ẩn danh hóa):** Loại bỏ các định danh kết nối dữ liệu với một cá nhân cụ thể.
- **Pseudo-anonymization (Ẩn danh giả):** Thay thế các định danh cá nhân bằng biệt danh, giúp bảo vệ quyền riêng tư trong khi vẫn duy trì tính chính xác của dữ liệu thống kê cho thử nghiệm và phân tích.

Chapter 5 Misuse and Abuse Case

Misuse/Abuse Cases

Chương 5 của tài liệu tập trung vào việc phát triển và sử dụng **Misuse/Abuse Cases** (Trường hợp lạm dụng hoặc sử dụng sai mục đích) như một công cụ quan trọng trong việc xác định các yêu cầu bảo mật phần mềm. Dưới đây là nội dung chi tiết:

1. Định nghĩa và Mục đích

- **Misuse/Abuse Cases** là một dạng của Use Case nhưng minh họa các **hành động cụ thể bị cấm**.

- Trong khi Use Case truyền tải những gì nên xảy ra khi người dùng hợp lệ yêu cầu các chức năng được phép, thì **Misuse Case mô tả khi một người dùng cố gắng thực hiện điều gì đó lẽ ra phải bị ngăn chặn**.
- Vai trò giá trị của chúng là **truyền đạt các yêu cầu bảo mật một cách rõ ràng cho các nhà phát triển và kiểm thử viên**.

2. Tác nhân trong Misuse Case (Misuse Actor)

- Tác nhân trong sơ đồ misuse case thường được dán nhãn là **"người dùng trái phép" (unauthorized user)**.
- Cần lưu ý rằng tác nhân này **có thể là một người dùng đã được xác thực (authenticated)** nhưng cố gắng thực hiện các hành động không được phép.
- **Ví dụ:** Một quản trị viên hệ thống có đặc quyền cao có thể lạm dụng quyền hạn để tạo một người thụ hưởng mới trái phép trong hệ thống thanh toán hóa đơn tự động. Misuse case giúp xác định các biện pháp giảm thiểu cho tình huống này, chẳng hạn như cơ chế xác thực ngoại băng (out-of-band) qua email.

3. Quá trình Phát triển

- **Thời điểm thực hiện:** Misuse cases được tạo ra sớm trong quy trình phát triển, cụ thể là ở **giai đoạn xác định yêu cầu**.
- **Phương pháp:** Cách thực tế và đơn giản nhất là thông qua **quá trình động não (brainstorming)** dựa trên thông tin có sẵn.
- **Thành phần tham gia:** Quá trình này thường là một phần của việc phát triển **mô hình đe dọa (threat model)**, thực hiện bởi một nhóm bao gồm cả nhà phát triển và nhân viên bảo mật.
- **Góc nhìn:** Nhóm sẽ xem xét hệ thống từ **quan điểm của kẻ tấn công**, bất kể đó là mối đe dọa từ bên trong hay bên ngoài tổ chức.

4. Ứng dụng và Lợi ích

- **Xác định yêu cầu phi chức năng:** Misuse cases giúp lập tài liệu cho các yêu cầu về chất lượng như **độ tin cậy (reliability), khả năng phục hồi (resiliency), tính bảo trì và tính kiểm thử**.
- **Mô hình hóa các tình huống lỗi:** Ngoài các tác nhân thù địch, chúng còn dùng để mô hình hóa các lỗi không do con người gây ra như: lỗi con người vô ý, thiên tai, lỗi thiết kế (bugs) hoặc nhiễu trên đường truyền thông tin.
- **Thúc đẩy kiểm thử:** Các misuse cases được xây dựng đúng cách sẽ **kích hoạt các kịch bản kiểm thử cụ thể**, đảm bảo các điểm yếu đã biết được nhận diện và xử lý trước khi phần mềm được triển khai.

5. Mối quan hệ với Use Case và Giảm thiểu rủi ro

- Trong các sơ đồ chức năng, use cases và misuse cases có thể kết nối với nhau.
- Khi một misuse case giao cắt với một use case (ví dụ: chức năng mã hóa dùng để xác thực người dùng), thì use case đó được gọi là **biện pháp giảm thiểu (mitigate)** cho misuse case.
- **Nguyên tắc quan trọng:** Tất cả các misuse cases phải được **giảm thiểu một cách thỏa đáng**, nếu không chúng sẽ đại diện cho một mối đe dọa và rủi ro tiềm tàng đối với hệ thống.

Requirements Traceability Matrix

Phần **Requirements Traceability Matrix (RTM - Ma trận truy xuất nguồn gốc yêu cầu)** trong Chương 5 được mô tả như một công cụ quản lý rủi ro và tài liệu quan trọng nhằm đảm bảo tính toàn vẹn của quá trình phát triển phần mềm. Dưới đây là nội dung chi tiết:

1. Định nghĩa và Vai trò

- **Định nghĩa:** RTM là một mạng lưới (grid) giúp đội ngũ phát triển theo dõi và quản lý các yêu cầu cũng như các chi tiết triển khai cụ thể.

- **Mục đích chính:** Nó hỗ trợ việc lập tài liệu về mối quan hệ giữa **yêu cầu bảo mật**, **biện pháp kiểm soát (controls)** và **nỗ lực kiểm thử/xác minh**.
- **Công cụ quản lý:** RTM cung cấp cho các quản trị viên dự án thông tin cần thiết để đảm bảo tất cả các yêu cầu đều được quản lý thích hợp và không có yêu cầu nào bị bỏ sót.

2. Các thành phần chính của một RTM (Bảng 5-1)

Theo cấu trúc mẫu trong tài liệu, một ma trận RTM điển hình bao gồm các trường thông tin sau:

- **Requirement ID Number:** Mã định danh duy nhất cho mỗi yêu cầu.
- **Requirement Description:** Mô tả chi tiết về yêu cầu cần được xác minh.
- **Requirement Source:** Nguồn gốc của yêu cầu (ví dụ: từ khách hàng, quy định pháp luật, hoặc tiêu chuẩn nội bộ).
- **Test Objective(s):** Các mục tiêu kiểm thử cụ thể để chứng minh sự tuân thủ yêu cầu.
- **Verification Method(s):** Phương pháp được sử dụng để xác minh mục tiêu kiểm thử (ví dụ: kiểm tra mã nguồn, chạy thử phần mềm).
- **Use Cases:** Các trường hợp sử dụng (Use Cases) tương ứng liên quan đến yêu cầu đó.

3. Lợi ích trong Quy trình Phát triển (SDLC)

- **Tự động hóa:** RTM cho phép tự động hóa nhiều yêu cầu, giúp đội ngũ tập hợp các bộ yêu cầu từ những hệ thống tập trung. Các yêu cầu bảo mật có thể được đưa vào hàng loạt dựa trên đánh giá về các hệ thống và người dùng liên quan.
- **Đảm bảo phạm vi bao phủ:** Sử dụng RTM giúp đảm bảo các yêu cầu quan trọng không bị bỏ qua, đặc biệt khi các môi trường khác nhau (như người dùng nội bộ so với giao diện web) có các bộ yêu cầu khác nhau.

- **Hỗ trợ các công cụ khác:** RTM hỗ trợ việc xây dựng các **Use Case** và đảm bảo rằng mọi thành phần đều được bao phủ trong quá trình kiểm thử.

Việc sử dụng RTM giúp kết nối "các dấu chấm" giữa kiến thức về rủi ro, thiết kế bảo mật và việc xác nhận thực tế rằng phần mềm đã đáp ứng các tiêu chuẩn đó trước khi triển khai.

Software Acquisition

Phần **Software Acquisition (Mua sắm/Tiểu nạp phần mềm)** trong Chương 5 tập trung vào việc hiểu rằng phần mềm hiện đại hiếm khi được xây dựng hoàn toàn mới từ đầu mà thường là sự kết hợp của nhiều thành phần có sẵn. Dưới đây là nội dung chi tiết:

1. Định nghĩa và Thuật ngữ (Definitions and Terminology)

Trong môi trường phần mềm an toàn, có hai thuật ngữ quan trọng cần lưu ý:

- **COTS (Commercial Off-The-Shelf):** Phần mềm thương mại có sẵn để mua và tích hợp ngay vào hệ thống.
- **GOTS (Government Off-The-Shelf):** Phần mềm được phát triển riêng cho nhu cầu của chính phủ. GOTS thường chuyên biệt hơn và có chi phí đơn vị cao hơn do cơ sở người dùng nhỏ.

2. Quyết định Tự xây dựng hay Mua (Build vs. Buy Decision)

Việc mua sắm phần mềm có thể được thực hiện qua hai cách: tự xây dựng hoặc mua. Trong thế giới module hóa ngày nay, ranh giới này thường bị xóa nhòa vì một ứng dụng có thể bao gồm cả thành phần tự xây dựng và thành phần mua ngoài.

- **Thành phần nên mua:** Các yếu tố phổ thông như phần mềm cơ sở dữ liệu.
- **Thành phần nên tự xây dựng:** Các hoạt động cốt lõi mang tính sống còn đối với nhiệm vụ, liên quan đến thông tin kinh doanh độc quyền.
- **Yếu tố then chốt:** Độ phù hợp giữa phần mềm và các quy trình kinh doanh hiện tại của doanh nghiệp.

3. Thuê ngoài (Outsourcing)

Phát triển phần mềm là một nỗ lực tốn kém, dẫn đến xu hướng thuê ngoài (offshoring) để tận dụng nhân lực kỹ năng cao với chi phí thấp hơn. Tuy nhiên, thuê ngoài mang lại những thách thức:

- **Khoảng cách địa lý:** Gây khó khăn trong quản lý và tăng chi phí phối hợp.
- **Tri thức ngầm:** Sự hiểu biết sâu sắc về dự án khó duy trì khi đội ngũ bị chia cắt về địa lý.
- **Chi phí ẩn:** Nhiều chi phí phát triển không thể giảm bớt chỉ bằng cách chuyển việc code sang một khu vực địa lý rẻ hơn.

4. Điều khoản Hợp đồng và Thỏa thuận Mức độ Dịch vụ (SLA)

Hợp đồng và SLA dùng để thiết lập kỳ vọng về hiệu suất trong tương lai:

- **Điều khoản hợp đồng:** Phải bao gồm các tham chiếu đến các tiêu chuẩn hoặc biện pháp kiểm soát an ninh cụ thể (ví dụ: các tiêu chuẩn ISO hoặc NIST).
- **SLA:** Có thể bao gồm các tiêu chí chấp nhận mà phần mềm phải vượt qua trước khi được tích hợp.

5. Chuyển tiếp Yêu cầu xuống Nhà cung cấp (Requirements Flow Down)

Phần mềm thường sử dụng các thư viện bên thứ ba, điều này mang lại cả chức năng tốt lẫn các lỗi và lỗ hổng tiềm ẩn.

- **SBOM (Software Bill of Materials):** Một danh sách tài liệu chi tiết về những gì có trong phần mềm (giống như nhãn dinh dưỡng), giúp người dùng nhận biết các rủi ro từ mã nguồn bên thứ ba.
- **Phụ lục Hợp đồng Phần mềm An toàn (theo OWASP):** Các công ty đang đưa ngôn ngữ an ninh vào hợp đồng mua sắm, bao gồm các triết lý như: quyết định an ninh dựa trên rủi ro, các hoạt động an ninh phải được tích hợp (không tách rời), và việc công bố đầy đủ thông tin an ninh.
- **Các mục tiêu cụ thể cần thỏa thuận:** Bao gồm xác thực đầu vào, quản lý phiên, kiểm soát truy cập, xử lý lỗi, mã hóa và quy trình khắc phục lỗi.

Việc quản lý tốt quá trình mua sắm giúp đảm bảo an ninh xuyên suốt chuỗi cung ứng phần mềm, tránh những lỗ hổng "vết dầu loang" từ các thành phần bên thứ ba.

Part III Secure Software Architecture and Design

Chapter 6 Secure Software Architecture

Perform Threat Modeling

Phần **Perform Threat Modeling** (Thực hiện mô hình hóa mối đe dọa) thuộc Chương 6 của tài liệu bao gồm các đầu mục con chính sau đây:

- **Threat Model Development (Phát triển mô hình mối đe dọa):** Đây là một quy trình dựa trên làm việc nhóm, diễn ra qua nhiều giai đoạn. Các bước cụ thể trong mục này gồm:
    - **Identify Security Objectives (Xác định mục tiêu bảo mật):** Xác định các yêu cầu từ luật pháp, hợp đồng và tiêu chuẩn doanh nghiệp.
    - **System Decomposition (Phân rã hệ thống):** Sử dụng các mô hình như Sơ đồ luồng dữ liệu (DFD) để lập tài liệu về luồng dữ liệu, kho lưu trữ và các ranh giới tin cậy (trust boundaries).
    - **Threat Identification (Nhận diện mối đe dọa):** Sử dụng các phương pháp như **STRIDE** (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege) để tìm ra các rủi ro tiềm tàng.
    - **Mitigation Analysis (Phân tích giảm nhẹ):** Xác định các biện pháp khắc phục như thiết kế lại, áp dụng biện pháp kiểm soát tiêu chuẩn, hoặc chấp nhận lỗ hổng.
    - **Threat Model Validation (Xác nhận mô hình mối đe dọa):** Đánh giá chất lượng của các mối đe dọa và biện pháp giảm nhẹ đã được xác định.
- **Attack Surface Evaluation (Đánh giá bề mặt tấn công):** Đánh giá các phần mã nguồn hoặc tài nguyên mà các bên trái phép có thể tiếp cận.

- **Attack Surface Measurement (Đo lường bề mặt tấn công):** Đếm và đo lường số lượng các điểm tiếp cận (như sockets mở, các endpoint RPC, các tài khoản được kích hoạt, v.v.).
- **Attack Surface Minimization (Giảm thiểu bề mặt tấn công):** Tìm cách hạ thấp các chỉ số bề mặt tấn công bằng cách tắt các tính năng không cần thiết hoặc hạn chế đặc quyền.
- **Threat Intelligence (Tình báo mối đe dọa):** Sử dụng thông tin hành động về các tác nhân độc hại, công cụ và phương pháp của chúng để ưu tiên các nỗ lực phản ứng.
- **Threat Hunting (Săn tìm mối đe dọa):** Quy trình lặp đi lặp lại việc chủ động tìm kiếm các mối đe dọa bên trong mạng dựa trên việc tạo và kiểm tra các giả thuyết.

Threat Model Development

Phần **Threat Model Development (Phát triển mô hình mối đe dọa)** trong Chương 6 mô tả một quy trình làm việc nhóm, diễn ra xuyên suốt vòng đời phát triển phần mềm (SDL) để nhận diện và lập tài liệu về các mối đe dọa cũng như biện pháp giảm nhẹ. Quy trình này gồm 5 giai đoạn chính:

1. Identify Security Objectives (Xác định mục tiêu bảo mật)

Đây là bước thiết yếu diễn ra trong quá trình xác định yêu cầu để hiểu rõ hệ thống cần bảo vệ điều gì.

- **Nguồn gốc:** Các mục tiêu này đến từ các yêu cầu pháp lý, hợp đồng, tiêu chuẩn doanh nghiệp và các yếu tố về quyền riêng tư.
- **Tài liệu hóa:** Cần ghi lại lý do kinh doanh của việc thu thập/lưu trữ dữ liệu và cách sử dụng chúng.
- **Lưu ý:** Không nên để đội ngũ phát triển tự quyết định việc bảo vệ dữ liệu vì họ có thể thiếu kiến thức chuyên sâu về các quy định pháp lý phức tạp.

2. System Decomposition (Phân rã hệ thống)

Giai đoạn này giúp các nhà thiết kế đảm bảo các mục tiêu bảo mật được bao phủ trong thiết kế thực tế.

- **Công cụ: Sơ đồ luồng dữ liệu (DFD)** được xem là lựa chọn tốt nhất để lập tài liệu vì mục tiêu tấn công thường là thông tin đang được xử lý.
- **Các thành phần cần nhận diện:** Bao gồm tất cả các tiến trình (processes), kho lưu trữ dữ liệu (data stores), luồng dữ liệu (data flows) và các **ranh giới tin cậy (trust boundaries)**.
- **Ranh giới tin cậy:** Đây là điểm mấu chốt vì chúng đại diện cho nơi kẻ tấn công có thể can thiệp vào hệ thống. Bên trong ranh giới này, các thực thể chia sẻ cùng đặc quyền và quyền truy cập.
- **Ví dụ về các yếu tố DFD:** Thực thể bên ngoài (người dùng, hệ thống khác), kho dữ liệu (tệp, cơ sở dữ liệu, registry), và luồng dữ liệu (lời gọi hàm, RPC, lưu lượng mạng).

3. Threat Identification (Nhận diện mối đe dọa)

Khi hệ thống đã được lập tài liệu đầy đủ, nhóm bắt đầu tìm kiếm các rủi ro tiềm tàng.

- **Phương pháp:** Thường sử dụng phương pháp **STRIDE** để kiểm tra các tiến trình, luồng dữ liệu và ranh giới tin cậy.
- **Mô hình STRIDE:**
    - **S**poofing (Mạo danh): Đe dọa tính xác thực (Authentication).
    - **T**ampering (Xáo trộn): Đe dọa tính toàn vẹn (Integrity).
    - **R**epudiation (Chối bỏ): Đe dọa tính chống chối bỏ (Nonrepudiation).
    - **I**nformation disclosure (Tiết lộ thông tin): Đe dọa tính bảo mật (Confidentiality).
    - **D**enial of service (Từ chối dịch vụ): Đe dọa tính khả dụng (Availability).
    - **E**levation of privilege (Leo thang đặc quyền): Đe dọa tính ủy quyền (Authorization).

4. Mitigation Analysis (Phân tích giảm nhẹ)

Mỗi mối đe dọa cần có một biện pháp xử lý. Có 4 loại hình giảm nhẹ:

- **Thiết kế lại (Redesign):** Loại bỏ hoàn toàn lỗ hổng (phương pháp tốt nhất).
- **Áp dụng biện pháp tiêu chuẩn:** Sử dụng các biện pháp đã biết như danh sách kiểm soát truy cập (ACL).
- **Sáng tạo biện pháp mới:** Rủi ro và tốn kém hơn do cần kiểm thử kỹ lưỡng.
- **Chấp nhận lỗ hổng:** Lựa chọn cuối cùng nếu không còn cách nào khác.

Công cụ hỗ trợ:

- **Attack Tree (Cây tấn công):** Sơ đồ đồ họa biểu diễn một cuộc tấn công với mục tiêu là nút gốc, từ đó liệt kê các điều kiện cần thiết để thực hiện.
- **Ưu tiên mối đe dọa:** Sử dụng mô hình **Xác suất x Tác động** (thang điểm 1-9) hoặc mô hình **DREAD** (Damage, Reproducibility, Exploitability, Affected users, Discoverability) để xếp hạng các rủi ro cần xử lý trước.

5. Threat Model Validation (Xác nhận mô hình mối đe dọa)

Diễn ra tại các cổng (gates) giữa các giai đoạn của SDL để đánh giá chất lượng của mô hình.

- Đảm bảo các mối đe dọa mô tả chi tiết về cách tấn công và tác động trong ngữ cảnh ứng dụng.
- Kiểm tra xem tất cả các biện pháp giảm nhẹ đã được liên kết với các mối đe dọa cụ thể hay chưa.
- Lập tài liệu cho tất cả các phụ thuộc (mã nguồn bên thứ ba) và các giả định được đưa ra trong quá trình phát triển.

Attack Surface Evaluation

Phần **Attack Surface Evaluation** (Đánh giá bề mặt tấn công) trong Chương 6 tập trung vào việc xác định và đo lường các điểm mà kẻ tấn công có thể sử dụng để xâm nhập vào hệ thống. Dưới đây là nội dung chi tiết dựa trên tài liệu:

1. Định nghĩa về Bề mặt tấn công

- **Bề mặt tấn công (Attack Surface)** của phần mềm là tập hợp các đoạn mã bên trong hệ thống mà các bên trái phép có thể tiếp cận được.
- Nó không chỉ bao gồm mã nguồn mà còn bao gồm một phạm vi rộng lớn các tài nguyên liên quan như: các **trường nhập liệu (input fields)** của người dùng, các **giao thức (protocols)**, các **giao diện (interfaces)**, các **tệp tài nguyên (resource files)** và các **dịch vụ (services)**.
- Bề mặt tấn công không đại diện cho chất lượng mã nguồn; nó không có nghĩa là mã nguồn có lỗi, mà chỉ là thước đo có bao nhiêu tính năng hoặc thành phần khả dụng để bị tấn công.

2. Nguyên tắc Đánh giá và Đo lường

- **Mục tiêu đo lường:** Để hiểu bề mặt tấn công, nhóm phát triển cần đếm số lượng các cách mà phần mềm có thể bị "truy cập".
- **Mối liên hệ với rủi ro:** Một thước đo phổ biến là số lượng tuyệt đối hoặc số lượng có trọng số của các mục có thể tiếp cận được. Nguyên tắc cốt lõi rất đơn giản: càng nhiều thành phần có thể bị tấn công thì rủi ro càng lớn.
- **Thông tin cơ sở (Baseline):** Việc tính toán và đo lường bề mặt tấn công cung cấp thông tin cơ sở để đưa ra các quyết định bảo mật. Mặc dù mỗi sản phẩm có đặc thù khác nhau và việc so sánh điểm số giữa các sản phẩm là không có giá trị, nhưng nó giúp nhóm phát triển theo dõi sự thay đổi rủi ro trong nội bộ dự án.

3. Các thành phần chính trong phép đo (Ví dụ từ Windows)

Tài liệu liệt kê một số yếu tố điển hình cần kiểm tra khi đo lường bề mặt tấn công:

- Các **sockets** đang mở và các **endpoint RPC** đang mở.
- Các **dịch vụ (services)**, đặc biệt là những dịch vụ chạy mặc định hoặc chạy với quyền SYSTEM.
- Các trình xử lý web (web handlers) đang hoạt động như các tệp ASP, HTR.

- Các tài khoản được kích hoạt (enabled accounts), đặc biệt là trong nhóm quản trị (admin group).
- Các danh sách kiểm soát truy cập (ACL) yếu trong hệ thống tệp hoặc registry.
- Lịch sử các lỗ hổng đã biết từ các lần phát triển trước đó cũng là nguồn thông tin quan trọng để xác định bề mặt tấn công.

4. Giảm thiểu bề mặt tấn công (Minimization)

- **Chiến lược:** Sau khi có thông tin cơ sở, bước tiếp theo là tìm cách hạ thấp chỉ số này bằng cách tắt các thành phần không cần thiết đối với đa số người dùng.
- **Đặc quyền tối thiểu:** Việc giảm thiểu này là một hình thức áp dụng **nguyên tắc đặc quyền tối thiểu (least privilege)**. Ví dụ: nếu ứng dụng có khả năng nhận biết mạng, việc giới hạn truy cập vào một số endpoint nhất định hoặc dải IP cụ thể sẽ làm giảm bề mặt tấn công.
- **Vòng đời phát triển:** Bề mặt tấn công nên được tính toán và lập tài liệu xuyên suốt quá trình phát triển (SDL). Việc thực hiện các thay đổi để giảm thiểu bề mặt tấn công càng sớm thì tác động đến các thành phần xung quanh càng ít.

5. Vai trò và Lợi ích

- Việc hiểu rõ bề mặt tấn công giúp hỗ trợ đưa ra các quyết định bảo mật tốt hơn liên quan đến chức năng của phần mềm.
- Nó giúp đội ngũ phát triển nhận diện rủi ro liên quan đến các hệ quả của thiết kế và quá trình phát triển.
- Việc lập tài liệu về bề mặt tấn công hiện tại giúp mọi thành viên hiểu rõ hệ quả bảo mật từ các quyết định cụ thể của họ.

Attack Surface Measurement

Phần **Attack Surface Measurement (Đo lường bề mặt tấn công)** trong Chương 6 tập trung vào việc định lượng các cách thức mà một sản phẩm phần mềm có thể bị truy cập trái phép. Dưới đây là nội dung chi tiết dựa trên tài liệu:

1. Nguyên tắc đo lường

Để hiểu rõ bề mặt tấn công của một sản phẩm, nhóm phát triển cần **đo lường số lượng các cách thức mà phần mềm có thể bị "truy cập"**. Tài liệu nhấn mạnh rằng việc đo lường này không nhằm mục đích so sánh giữa các sản phẩm khác nhau vì mỗi sản phẩm là duy nhất, nhưng nó cung cấp một **thông tin cơ sở (baseline)** để đưa ra các quyết định bảo mật nội bộ.

2. Các thành phần điển hình cần đo lường

Tài liệu cung cấp một danh sách các yếu tố (ví dụ từ môi trường Windows) thường được kiểm tra khi đo lường bề mặt tấn công:

- **Kết nối mạng:** Các sockets đang mở, các endpoint RPC đang mở, các named pipes đang mở.
- **Dịch vụ (Services):** Tổng số dịch vụ, các dịch vụ chạy mặc định, và đặc biệt là các **dịch vụ chạy với quyền SYSTEM** (đây là những điểm rủi ro cao hơn).
- **Giao diện Web:** Các trình xử lý web (web handlers) đang hoạt động (như tệp ASP, HTR), các bộ lọc ISAPI đang hoạt động, các trang web động và các thư mục ảo có quyền thực thi.
- **Tài khoản người dùng:** Các tài khoản được kích hoạt, đặc biệt là các tài khoản nằm trong **nhóm quản trị (admin group)**; tài khoản khách (guest) được kích hoạt.
- **Kiểm soát truy cập (ACL):** Các danh sách kiểm soát truy cập yếu trong hệ thống tệp, trong registry hoặc trên các tài nguyên chia sẻ (shares); các phiên đăng nhập ẩn danh (null sessions) tới pipes và shares.

3. Tầm quan trọng của dữ liệu lịch sử

Ngoài việc đếm các điểm truy cập hiện tại, **lịch sử các lỗ hổng đã biết** từ những lần phát triển trước đó là một nguồn thông tin quý giá. Việc xác định nguyên nhân gốc rễ của các lỗ hổng cũ giúp đội ngũ xác định chính xác hơn các yếu tố cấu thành bề mặt tấn công hiện tại.

4. Đánh giá rủi ro thông qua đo lường

- **Ước tính khả năng bị tổn thương:** Việc đo lường bề mặt tấn công được coi là một phương pháp ước tính vững chắc cho **bề mặt lỗ hổng tiềm tàng**. Mặc dù một thành phần trong bề mặt tấn công không nhất thiết phải là một lỗ hổng, nhưng đó là nơi kẻ tấn công sẽ cố gắng can thiệp.
- **Trọng số rủi ro:** Các yếu tố khác nhau có giá trị rủi ro khác nhau. Ví dụ: một dịch vụ chạy với quyền hệ thống sẽ rủi ro hơn dịch vụ không có đặc quyền, và dịch vụ chạy mặc định rủi ro hơn dịch vụ chỉ chạy khi có yêu cầu.

Việc tính toán và lập tài liệu về bề mặt tấn công nên được thực hiện **xuyên suốt quá trình phát triển (SDL)** để giúp mọi người hiểu rõ hệ quả bảo mật từ các quyết định thiết kế của mình.

Attack Surface Minimization

Phần **Attack Surface Minimization (Giảm thiểu bề mặt tấn công)** trong Chương 6 tập trung vào các chiến lược để hạ thấp các chỉ số rủi ro sau khi đã thiết lập được thông tin cơ sở (baseline) từ bước đo lường. Dưới đây là nội dung chi tiết từ tài liệu:

1. Nguyên tắc cốt lõi

- **Mối liên hệ rủi ro:** Bề mặt tấn công được xem là sự đại diện cho **bề mặt lỗ hổng tiềm tàng** của phần mềm.
- **Mục tiêu:** Khi đã biết thông tin cơ sở, mục tiêu tiếp theo là tìm cách hạ thấp số lượng các điểm truy cập này.
- **Áp dụng đặc quyền tối thiểu:** Việc giảm thiểu này thực chất là một hình thức áp dụng **nguyên tắc đặc quyền tối thiểu (least privilege)**.

2. Các phương pháp giảm thiểu

Tài liệu liệt kê các cách cụ thể để thu hẹp bề mặt tấn công:

- **Tắt các thành phần không thiết yếu:** Vô hiệu hóa các tính năng hoặc thành phần không cần thiết đối với đa số người dùng để giảm bề mặt cấu hình mặc định.
- **Quản lý dịch vụ:**

    - Chuyển trạng thái các dịch vụ sang "tắt" theo mặc định hoặc chỉ chạy khi cần thiết (on demand).
    - Hạn chế các dịch vụ chạy với quyền hệ thống (SYSTEM); thay vào đó, chạy với mức đặc quyền thấp nhất có thể.
- **Kiểm soát truy cập mạng:** Đối với các ứng dụng nhận biết mạng (network aware), việc giới hạn truy cập vào một số lượng **endpoint nhất định** hoặc một **dải IP hạn chế** sẽ giúp giảm bề mặt tấn công.
- **Tùy biến quyền truy cập:** Không nhất thiết là "bật hoặc tắt" hoàn toàn; một số thành phần có thể tắt đối với đa số nhưng lại bật cho những người dùng cụ thể trong các trường hợp phù hợp.

3. Quy trình thực hiện trong SDL

- **Tính toán liên tục:** Bề mặt tấn công nên được tính toán xuyên suốt toàn bộ quá trình phát triển (SDL).
- **Lập tài liệu:** Mọi công việc nhằm giảm thiểu bề mặt tấn công cần được ghi lại đầy đủ. Việc này giúp hạ thấp mức bề mặt mặc định khi triển khai và cung cấp thông tin cho khách hàng để họ đưa ra quyết định triển khai an toàn.
- **Sự can thiệp sớm:** Thay đổi càng sớm trong quy trình phát triển thì tác động tiêu cực đến các thành phần xung quanh càng ít.
- **Tích hợp vào thiết kế:** Việc giảm thiểu cần được xem xét ngay từ nỗ lực thiết kế ban đầu. Tài liệu nhấn mạnh rằng bề mặt tấn công là một chỉ số thay thế (surrogate) vững chắc để đo lường hiệu quả an ninh.

Việc nắm giữ tài liệu hiện tại về bề mặt tấn công giúp toàn đội hiểu rõ các hệ quả an ninh từ các quyết định cụ thể của họ.

Threat Intelligence

Phần **Threat Intelligence (Tình báo mối đe dọa)** trong Chương 6 tập trung vào việc sử dụng các thông tin thực tế về bối cảnh đe dọa để hỗ trợ quá trình bảo mật phần mềm. Dưới đây là nội dung chi tiết:

1. Định nghĩa và Mục tiêu

- **Định nghĩa:** Tình báo mối đe dọa là **thông tin có thể hành động (actionable information)** về các tác nhân độc hại, công cụ, hạ tầng và các phương pháp của chúng.
- **Mục tiêu:** Đây là công cụ quan trọng cho những người phòng thủ trong việc săn tìm kẻ tấn công. Vì không có tổ chức nào đủ nguồn lực để bảo vệ mọi thứ hoặc điều tra mọi hành động thù địch, tình báo mối đe dọa giúp **quản lý nguồn lực** hiệu quả bằng cách tập trung vào những gì quan trọng nhất.

2. Ưu tiên hành động với khái niệm Kill Chain

- Khi kết hợp tình báo mối đe dọa với khái niệm **kill chain** (con đường mà kẻ tấn công có khả năng nhất sẽ thực hiện), tổ chức có thể **ưu tiên các hành động** chống lại những mối đe dọa có ý nghĩa và nguy cơ cao nhất.

3. Vai trò đối với Nhà phát triển phần mềm

- Mặc dù nhà phát triển không nhất thiết cần nhận thức tình huống theo thời gian thực (như đội ứng phó sự cố), nhưng những rủi ro mà tình báo mối đe dọa liệt kê thường là **hệ quả từ hành động của chính nhà phát triển**.
- **Tập trung nỗ lực:** Việc biết kẻ tấn công đang làm gì và tấn công vào đâu giúp đội ngũ phát triển tập trung nguồn lực vào đúng chỗ.
- **Thống kê thực tế:** Phần lớn các cuộc tấn công chỉ dựa trên một số ít lỗ hổng. Theo báo cáo VDBIR, chỉ có 10 lỗ hổng chiếm đến 97% tổng số các cuộc tấn công vào doanh nghiệp năm 2014, và hơn 99% trong số đó khai thác các lỗ hổng đã cũ (hơn một năm tuổi).

4. Quản lý linh kiện từ bên thứ ba

- Khi các nhà phát triển tích hợp các **thư viện bên thứ ba, mã nguồn mở hoặc thương mại** vào dự án, họ phải theo dõi các nguồn này để phát hiện các lỗ hổng mới được công bố.
- Chức năng này của tình báo mối đe dọa cung cấp dữ liệu đầu vào quan trọng cho đội ngũ phát triển để kịp thời cập nhật hoặc vá lỗi cho các thành phần thuê ngoài này.

Tình báo mối đe dọa giúp kết nối giữa thực tế tấn công đang diễn ra ngoài thị trường với quy trình phát triển SDL, đảm bảo các nỗ lực bảo mật không bị lãng phí vào những rủi ro không thực tế.

Threat Hunting

Phần **Threat Hunting (Săn tìm mối đe dọa)** trong Chương 6 mô tả một quy trình chủ động và lặp đi lặp lại để phát hiện các mối đe dọa tiềm tàng trong mạng. Dưới đây là nội dung chi tiết:

1. Định nghĩa và Phương pháp tiếp cận

- **Định nghĩa:** Săn tìm mối đe dọa là một **quy trình lặp đi lặp lại** của việc chủ động tìm kiếm các mối đe dọa bên trong mạng nội bộ.
- **Phương pháp dựa trên giả thuyết:** Một trong những mô hình hiệu quả nhất là dựa trên việc **tạo ra một giả thuyết** và sau đó tiến hành kiểm tra giả thuyết đó. Việc này giúp xác định phạm vi cuộc săn tìm, giúp người thực hiện tập trung vào các mục cụ thể thay vì tìm kiếm không mục đích trong một biển dữ liệu thông thường.

2. Ví dụ về Giả thuyết và Kiểm thử

- **Ví dụ giả thuyết:** "Một kẻ thù đang sử dụng các thông tin đăng nhập bị đánh cắp để giả danh người dùng hợp lệ trong giờ không làm việc".
- **Cách kiểm thử:** Giả thuyết này có thể được kiểm chứng bằng cách **kiểm tra các tập tệp nhật ký (logs)** để tìm các hoạt động cụ thể diễn ra trong những khung giờ không làm việc đó.

3. Mục tiêu và Lợi ích

- **Mục tiêu chính:** Sử dụng kiến thức hiện tại về những gì các đối thủ đang thực hiện đối với các doanh nghiệp khác để kiểm tra xem liệu điều đó có đang xảy ra trên mạng lưới của tổ chức mình hay không.
- **Lợi ích:** Phương pháp này giúp **tăng khả năng phát hiện các hoạt động độc hại** vượt ra ngoài những cảnh báo hoặc trình kích hoạt sự cố (incident triggers) thông thường.

Tài liệu cũng giới thiệu thêm một nguồn tham khảo chuyên sâu là bài bạch thư (whitepaper) *"A Practical Model for Conducting Cyber Threat Hunting"* của Dan Gunter và Marc Seitz để tìm hiểu thêm về mô hình thực tế.

Define the Security Architecture

Phần giới thiệu của mục **Define the Security Architecture** (Xác định Kiến trúc Bảo mật) trong Chương 6 tập trung vào bản chất tiến hóa của kiến trúc doanh nghiệp và tầm quan trọng của việc lập kế hoạch bảo mật một cách có chủ đích. Dưới đây là nội dung chi tiết:

Một doanh nghiệp hiện đại không bao giờ chỉ có duy nhất một dạng kiến trúc. Các hệ thống Công nghệ Thông tin (IT) phát triển theo thời gian thông qua một quy trình bồi đắp (accretive process), trong đó các hệ thống mới được thiết kế để đáp ứng các yêu cầu hiện tại, sau đó gia nhập cùng với các hệ thống sẵn có khác trong doanh nghiệp. Việc tích hợp chéo giữa các kiến trúc cho phép tái sử dụng dữ liệu và tăng đáng kể tiện ích tổng thể của toàn bộ kiến trúc doanh nghiệp.

Khi các dịch vụ và cơ hội mới được giới thiệu, nhu cầu tích hợp hoàn toàn (thay vì xây dựng lại các dịch vụ dữ liệu hiện có) là một phương án giúp giảm thiểu cả chi phí lẫn rủi ro. Trong tương lai, sự bồi đắp của doanh nghiệp sẽ tiếp tục diễn ra với việc bổ sung các năng lực mới và loại bỏ các năng lực không còn được sử dụng hoặc không cần thiết.

Kiến trúc sẽ luôn hình thành dù có được lập kế hoạch hay không. Tuy nhiên, để đạt được lợi ích thực sự của một kiến trúc bảo mật, nó phải là một **yếu tố được lập kế hoạch** nhằm hỗ trợ các kết quả bảo mật mong muốn. Điều này quan trọng đối với cả các dự án xây dựng mới hoàn toàn (greenfield projects) lẫn các đợt cập nhật, mở rộng hệ thống hiện tại.

Việc xem xét kiến trúc dựa trên các mục tiêu bảo mật và truyền đạt thiết kế đó cho toàn đội ngũ sẽ cung cấp hướng dẫn cho các nhà phát triển khi họ cần đưa ra lựa chọn về các tùy chọn và công nghệ sử dụng. Các yếu tố then chốt ở đây chính là một **thiết kế chu đáo, vững chắc** và việc **truyền đạt thông tin** đó đến toàn đội ngũ.

Security Control Identification and Prioritization

Phần **Security Control Identification and Prioritization** (Xác định và Ưu tiên các Kiểm soát Bảo mật) thuộc Chương 6 của tài liệu cung cấp các nội dung chi tiết sau:

1. Định nghĩa về Kiểm soát Bảo mật (Security Controls)

- **Kiểm soát bảo mật** là các cơ chế được sử dụng để đạt được các mục tiêu bảo mật trong hệ thống.
- Chúng được định nghĩa là các biện pháp được thực hiện để **phát hiện (detect), ngăn chặn (prevent) hoặc giảm thiểu (mitigate)** các rủi ro liên quan đến các mối đe dọa mà hệ thống phải đối mặt.
- Các kiểm soát này còn được gọi là các **biện pháp đối phó (countermeasures)** hoặc **biện pháp bảo vệ (safeguards)**.

2. Phân loại Kiểm soát theo Hành động

Các kiểm soát có thể được phân loại dựa trên loại hành động mà chúng thực hiện thành ba nhóm chính:

- **Administrative (Hành chính):** Các chính sách, quy trình và hướng dẫn quản lý.
- **Technical (Kỹ thuật):** Các giải pháp phần cứng hoặc phần mềm (ví dụ: quyền truy cập, mã hóa).
- **Physical (Vật lý):** Các biện pháp bảo vệ tài sản vật chất.

3. Bốn loại Kiểm soát trong mỗi nhóm

Trong mỗi nhóm hành chính, kỹ thuật hoặc vật lý, có bốn loại kiểm soát cụ thể:

- **Preventive (Ngăn ngừa - deterrent):**
    - Được sử dụng để ngăn chặn lỗ hổng bị khai thác.
    - Có tính chất chủ động (proactive) và cung cấp mức độ giảm thiểu rủi ro rộng nhất trên mỗi kiểm soát.
    - **Ví dụ:** Phân chia nhiệm vụ (separation of duties), tài liệu đầy đủ, kiểm soát vật lý đối với tài sản và cơ chế ủy quyền.

- **Detective (Phát hiện):**
    - Được sử dụng khi các kiểm soát ngăn ngừa thất bại và một lỗ hổng bị khai thác.
    - Hoạt động sau khi sự việc đã xảy ra để phát hiện sự hiện diện của một cuộc tấn công.
    - **Ví dụ:** Nhật ký hệ thống (logs), kiểm toán (audits) và kiểm kê (inventories).
- **Corrective (Khắc phục - recovery):**
    - Khắc phục hệ thống sau khi lỗ hổng bị khai thác và gây ra tác động.
    - Hoạt động sau sự cố và thường nhắm vào hệ thống bị tấn công hơn là vectơ tấn công để giảm tổng tác động.
    - **Ví dụ:** Sao lưu dữ liệu (backups) giúp phục hồi hiệu quả hơn sau tấn công.
- **Compensating (Bù đắp):**
    - Được thiết kế để hoạt động khi bộ kiểm soát chính đã thất bại.
    - Thường được coi là một hình thức của chiến lược **phòng thủ theo chiều sâu (defense in depth)**.
    - **Ví dụ:** Một cuộc rà soát tài chính đối với các báo cáo kế toán có thể đóng vai trò là kiểm soát bù đắp sau sự cố cho việc phân chia nhiệm vụ ngăn ngừa gian lận.

4. Khung Kiểm soát (Controls Framework)

- Các kiểm soát không hoạt động độc lập mà phối hợp với nhau như một hệ thống hoàn chỉnh để giảm thiểu rủi ro trong toàn doanh nghiệp.
- Việc hiểu rõ môi trường rủi ro và áp dụng các kiểm soát bảo mật để quản lý rủi ro đó là trách nhiệm thuộc phạm vi của nhóm phát triển phần mềm.

Distributed Computing

Phần **Distributed Computing** (Tính toán phân tán) trong Chương 6 của tài liệu tập trung vào các kiến trúc cho phép phân chia việc xử lý và lưu trữ dữ liệu ra nhiều hệ thống khác nhau thay vì tập trung tại một chỗ. Dưới đây là nội dung chi tiết:

1. Tổng quan về Distributed Computing

- Sự ra đời của các giải pháp máy tính nhỏ gọn, giá rẻ nhưng có khả năng xử lý mạnh mẽ đã thúc đẩy việc đưa tính toán ra khỏi trung tâm và đến gần hơn với người dùng trong doanh nghiệp.
- Khả năng kết nối mạng và các tùy chọn lưu trữ mở rộng giúp hỗ trợ việc phân tán xử lý này.

2. Kiến trúc Client-Server (Khách-Chủ)

- **Định nghĩa:** Được xác định bởi hai loại máy phối hợp với nhau: **Server** (Máy chủ) có khả năng xử lý/lưu trữ lớn phục vụ nhiều người dùng và **Client** (Máy trạm) phục vụ công việc của một người dùng duy nhất.
- **Phân loại Client:**
    - **Thin clients (Máy trạm mỏng):** Thực hiện phần lớn việc xử lý trên máy chủ.
    - **Fat clients (Máy trạm béo):** Tự thực hiện một lượng đáng kể việc xử lý.
- **Mô hình N-tier:** Đây là phương pháp mô tả kiến trúc đa máy tính. Ví dụ, mô hình 3 lớp (three-tier) bao gồm máy trạm kết nối với máy chủ trung gian (xử lý logic nghiệp vụ) và sau đó đến máy chủ cơ sở dữ liệu để quản lý lưu trữ.
- **Bảo mật:** Việc phân tán xử lý và lưu trữ trên nhiều máy làm tăng nhu cầu về bảo mật truyền thông giữa chúng. Việc tách kiến trúc thành các lớp (layers) giúp quản lý chức năng nghiệp vụ, tích hợp và bảo mật dễ dàng hơn.
- **Điện toán đám mây (Cloud computing):** Được coi là một trường hợp cực đoan của mô hình client-server, triển khai các yếu tố của mô hình n-tier dưới dạng SaaS, PaaS và IaaS.

3. Kiến trúc Peer-to-Peer (P2P - Ngang hàng)

- Đặc trưng bởi các tập hợp máy tính hoạt động như những người bạn đồng hành (peers).
- Thay vì phân chia nhiệm vụ và quyền lực như client-server, các thiết bị thành viên trong P2P chia sẻ công việc với nhau ở mức độ xử lý tương đương.
- Mô hình này thường được tìm thấy trong việc chuyển giao thông tin và chia sẻ tệp trực tiếp giữa các máy mà không cần một trung tâm lưu trữ trung gian.

4. Message Queuing (Hàng đợi thông báo)

- Đây là công nghệ giải quyết thách thức về quản lý lưu lượng và đảm bảo phân phối dữ liệu khi di chuyển thông tin giữa các hệ thống xử lý khác nhau.
- Nó sử dụng một **máy chủ trung gian** để điều phối việc truyền và nhận thông tin giữa các quy trình.
- Trong các hệ thống doanh nghiệp lớn, hàng đợi thông báo có thể giải quyết các vấn đề về luồng dữ liệu điểm-đến-đa-điểm (point-to-multipoint), hỗ trợ việc ghi nhật ký (logging) và bảo mật luồng dữ liệu.

Service-Oriented Architecture

Phần **Service-Oriented Architecture (SOA - Kiến trúc hướng dịch vụ)** trong Chương 6 mô tả một loại kiến trúc phân tán với các đặc điểm và yêu cầu bảo mật cụ thể như sau:

1. Các đặc điểm cốt lõi của SOA

Kiến trúc hướng dịch vụ được xác định bởi các đặc tính then chốt giúp tối ưu hóa khả năng tích hợp và sử dụng lại trong doanh nghiệp:

- **Platform neutrality (Tính trung lập với nền tảng):** Có khả năng hoạt động trên nhiều nền tảng phần cứng và hệ điều hành khác nhau.
- **Interoperability (Khả năng tương tác):** Cho phép các hệ thống khác nhau giao tiếp và làm việc cùng nhau một cách liền mạch.

- **Modularity and reusability (Tính mô-đun và khả năng tái sử dụng):** Các dịch vụ được xây dựng dưới dạng các đơn vị chức năng độc lập, có thể dùng lại cho nhiều mục đích khác nhau.
- **Abstracted business functionality (Trừu tượng hóa chức năng nghiệp vụ):** Các logic nghiệp vụ phức tạp được đóng gói bên trong các dịch vụ, người dùng chỉ cần biết cách gọi dịch vụ mà không cần biết cách nó thực hiện bên trong.
- **Contract-based interfaces (Giao diện dựa trên hợp đồng):** Các tương tác được xác định rõ ràng thông qua các thỏa thuận hoặc giao thức cụ thể.
- **Discoverability (Khả năng khám phá):** Các dịch vụ có thể được tìm thấy và nhận diện trong hệ thống thông qua các cơ chế đăng ký.

2. Công nghệ triển khai và Giao thức

SOA có thể được triển khai bằng nhiều công nghệ khác nhau:

- **Các mô hình đối tượng:** Bao gồm COM (Common Object Model) và CORBA (Common Object Request Broker Architecture).
- **Web Services (Dịch vụ Web):** Đây là phương thức triển khai phổ biến nhất, thường sử dụng ngôn ngữ **XML** để truyền tin.
- **Giao thức thông điệp:** Hai giao thức phổ biến nhất được sử dụng trong SOA là **SOAP** (Simple Object Access Protocol) và **REST** (Representational State Transfer).

3. Bảo mật trong SOA

Việc sử dụng XML làm phương thức truyền tin chính mang lại sự linh hoạt nhưng cũng tạo ra các thách thức về bảo mật:

- Các thông điệp XML có thể được bảo vệ thông qua **mã hóa XML (XML encryption)**.
- Dữ liệu cũng có thể được truyền qua các **kênh truyền tải an toàn** như SSL/TLS để đảm bảo tính bảo mật và toàn vẹn.

4. Enterprise Service Bus (ESB - Trục dịch vụ doanh nghiệp)

ESB là một dạng cụ thể của kiến trúc SOA, đóng vai trò là lớp trung gian điều phối:

- **Chức năng:** ESB được thiết kế để giám sát và kiểm soát việc định tuyến thông điệp giữa các dịch vụ trong hệ thống.
- **Khả năng xử lý:** Nó có thể cấu hình để thực hiện chuyển đổi giao thức (protocol conversions), xử lý các sự kiện được xác định, và quản lý hàng đợi thông điệp (message queuing).
- **Kết nối đa phương thức:** ESB hoạt động như một ống dẫn cho phép các giao thức khác nhau như XML, EDI, WSDL, REST, DCOM và CORBA giao tiếp với nhau một cách thông suốt.

Web Services

Phần **Web Services** (Dịch vụ Web) trong Chương 6 của tài liệu mô tả các phương thức giao tiếp giữa các thành phần qua Internet với các nội dung chi tiết sau:

1. Định nghĩa và Đặc điểm cốt lõi

- **Web Services** là các hệ thống phần mềm được thiết kế để hỗ trợ khả năng **tương tác giữa máy với máy (machine-to-machine)** qua mạng.
- Đặc điểm nhận dạng chính của dịch vụ web là có **mô tả giao diện ở định dạng máy có thể đọc được**.

2. Ngôn ngữ mô tả dịch vụ (WSDL)

- Định dạng máy đọc được này được gọi là **Web Services Description Language (WSDL)**.
- **WSDL** là một ngôn ngữ dựa trên **XML**, được sử dụng để mô tả các chức năng mà dịch vụ web cung cấp, bao gồm:
    - Cách thức gọi dịch vụ.
    - Các tham số mà dịch vụ yêu cầu.
    - Các cấu trúc dữ liệu mà dịch vụ trả về.

3. Giao thức truyền tin: SOAP và REST

- **SOAP (Simple Object Access Protocol):** Ban đầu, các dịch vụ web được thiết kế với SOAP, sử dụng XML làm nền tảng truyền tin. Các thông điệp SOAP thường được chuyển tải qua HTTP với sự hỗ trợ của các tiêu chuẩn web khác như JSON.
- **REST (Representational State Transfer):** Hiện nay, có một xu hướng chuyển sang ưa chuộng REST vì nó giúp định nghĩa các dịch vụ web có khả năng **hoạt động ở quy mô Internet**.
- **Lý do chuyển đổi:** Có những lo ngại về hiệu suất khi sử dụng XML, cũng như sự phức tạp trong việc triển khai SOAP. Hệ thống tuân thủ REST cho phép truy cập tài nguyên web dưới dạng văn bản (text-based) theo cách **phi trạng thái (stateless)**.

4. Các loại Dịch vụ Web theo W3C

Tổ chức W3C xác định hai lớp dịch vụ web chính:

- **Dịch vụ web tuân thủ REST (REST-compliant):** Mục đích chính là thao tác các biểu diễn XML của tài nguyên web bằng một tập hợp các hoạt động "phi trạng thái" đồng nhất.
- **Dịch vụ web tùy ý (Arbitrary):** Dịch vụ có thể cung cấp một tập hợp các hoạt động tùy ý.

5. Khám phá và Giao diện (UDDI)

- **UDDI (Universal Description, Discovery, and Interface):** Đây là một phương pháp độc lập với nền tảng, giúp doanh nghiệp **khám phá và gọi các dịch vụ web một cách linh hoạt**.
- Sử dụng XML, UDDI được thiết kế như một **sổ đăng ký (registry)** dựa trên giao thức để các dịch vụ có thể niêm yết trên Internet. Tuy nhiên, UDDI chưa bao giờ được áp dụng rộng rãi trên toàn bộ Internet như kế hoạch ban đầu mà chủ yếu chỉ tìm thấy trong nội bộ các tổ chức.

6. Định dạng trao đổi dữ liệu JSON

- **JSON (JavaScript Object Notation):** Là một phương thức trao đổi dữ liệu phổ biến khác trong dịch vụ web.
- Mặc dù dựa trên JavaScript, JSON **độc lập với ngôn ngữ** và có thể được sử dụng trong hầu hết các ngôn ngữ lập trình hiện đại nhờ vào hai cấu trúc dữ liệu phổ biến: tập hợp các cặp tên-giá trị (objects) và danh sách các giá trị có thứ tự (arrays).

Rich Internet Applications

Phần **Rich Internet Applications (RIAs)** trong Chương 6 của tài liệu tập trung vào các kiến trúc ứng dụng hiện đại kết hợp giữa tính năng của ứng dụng máy tính để bàn và khả năng phân phối qua Internet. Dưới đây là nội dung chi tiết:

1. Định nghĩa và Mục tiêu

- **Rich Internet Applications (RIAs)** là một hình thức kiến trúc sử dụng Web làm **cơ chế truyền tải** và sử dụng máy trạm (client) làm **thiết bị xử lý**, thường là cho các chức năng kiểm soát định dạng hiển thị.
- **Mục tiêu chính:** Tạo ra một ứng dụng có các đặc điểm và trải nghiệm người dùng tương tự như một **ứng dụng máy tính để bàn (desktop application)** nhưng được phân phối qua môi trường Internet.
- **Ví dụ tiêu biểu:** Các trang mạng xã hội như **Facebook** là minh chứng rõ nhất cho mô hình RIA.

2. Công nghệ triển khai

- RIAs được xây dựng bằng nhiều khung làm việc (frameworks) khác nhau như **Adobe Flash, Java, và Microsoft Silverlight**.
- **Xu hướng hiện tại:** HTML5 cùng với JavaScript đang trở thành công nghệ thống trị cho việc xây dựng các RIAs trong tương lai.
- RIAs có khả năng đáp ứng một dải chức năng rộng, từ các giao diện nghiệp vụ phức tạp đến các trò chơi và nền tảng học tập.

3. Thách thức về Bảo mật

Mặc dù RIAs thực hiện phần lớn việc xử lý trên máy chủ hoặc nền tảng phía sau, vấn đề bảo mật không thể bị xem nhẹ:

- **Khai thác phía máy trạm (Client-side exploits):** Các ứng dụng này đặc biệt dễ bị tổn thương trước các mối đe dọa từ phía máy trạm do kiến trúc dựa trên máy trạm của chúng.
- **Thực thi mã từ xa (Remote Code Execution - RCE):** Đây là một cuộc tấn công nghiêm trọng, nơi kẻ tấn công kích hoạt việc thực thi mã tùy ý trên một máy tính từ một máy tính khác thông qua kết nối mạng. Mã độc này sẽ hoạt động dưới **quyền hạn bảo mật (security credentials)** của quy trình đang chạy.

4. Nguyên tắc bảo vệ cốt lõi

- **Xác thực đầu vào (Input Validation):** Trong mọi hoạt động của kiến trúc khách-chủ, một sự thật phổ biến là **không bao giờ tin tưởng đầu vào mà không được xác thực**.
- **Rủi ro từ máy trạm:** Ngay cả khi tổ chức có quyền kiểm soát máy trạm, vẫn luôn có rủi ro máy trạm bị hỏng do phần mềm độc hại, người dùng không hài lòng hoặc đơn giản là do cấu hình sai.
- **Sự nhầm lẫn giữa mã và dữ liệu:** Các cuộc tấn công RCE thành công thường là hậu quả của quyết định kiến trúc không phân biệt rõ ràng giữa **mã (code)** và **dữ liệu (data)**, cho phép đầu vào độc hại được thực thi như mã chương trình.

Pervasive/Ubiquitous Computing

Phần **Pervasive/Ubiquitous Computing** (Tính toán phổ biến/khắp nơi) trong Chương 6 của tài liệu mô tả một kỷ nguyên công nghệ nơi máy tính được tích hợp vào mọi thiết bị và hệ thống để cải thiện khả năng kiểm soát. Dưới đây là các nội dung chi tiết:

1. Tổng quan và Internet of Things (IoT)

- **Sự trỗi dậy của vi xử lý:** Sự ra đời của các bộ vi xử lý giá rẻ đã thúc đẩy xu hướng kết nối các thiết bị, chia sẻ thông tin và cho phép điều khiển từ xa.

- **Tính kết nối siêu độ (Hyper-connectedness):** Việc kết nối mọi thứ với nhau mang lại sự tiện lợi và hiệu quả, nhưng cũng tạo ra các hệ thống cực kỳ phức tạp với các vấn đề mới về an toàn, bảo mật và ổn định.
- **Internet vạn vật (IoT):** Đây là một tập hợp các thiết bị kết nối mạng hoạt động ở quy mô Internet. Các hệ thống này thường bao gồm số lượng lớn các thiết bị giá rẻ và hiếm khi có khả năng bảo mật mạnh mẽ một cách độc lập.
- **Yêu cầu bảo mật:** Các nguyên tắc như **phòng thủ theo chiều sâu (defense in depth)**, **thất bại ở trạng thái an toàn (fail to a secure state)** và **kiểm soát toàn diện (complete mediation)** trở nên quan trọng hơn bao giờ hết. Mỗi thành phần cần có khả năng tự túc và tự dựa vào chính mình về khía cạnh an ninh và ổn định.

2. Truyền thông không dây (Wireless)

- **Tính phổ biến:** Các giao thức như di động (cellular), Wi-Fi (802.11), Zigbee, Bluetooth và Wi-Max đang trở nên phổ biến, giúp giải phóng thiết bị khỏi dây dẫn và hỗ trợ tính di động.
- **Rủi ro bảo mật:** Trong mạng không dây, bất kỳ ai trong phạm vi tín hiệu đều có thể giám sát thông tin liên lạc. Do đó, các nhà phát triển phần mềm phải tự chịu trách nhiệm bảo mật dữ liệu khi truyền tải, thay vì kỳ vọng mạng lưới sẽ lo liệu việc đó.

3. Dịch vụ dựa trên vị trí (Location-Based) và Geofencing

- **Ứng dụng:** Dữ liệu vị trí được sử dụng cho tiếp thị hoặc bảo mật (ví dụ: chỉ cho phép thực hiện một chức năng khi người dùng ở một vị trí cụ thể).
- **Geofencing (Hàng rào địa lý):** Là việc kích hoạt một hành động khi thiết bị di động vượt qua một ranh giới địa lý được xác định bởi GPS hoặc khoảng cách từ một điểm nhất định.
- **Bảo mật:** Dữ liệu vị trí là dữ liệu nhạy cảm và cần được bảo vệ để tránh bị lạm dụng trong ứng dụng và các quy trình hạ nguồn.

4. Nhận dạng qua tần số vô tuyến (RFID) và NFC

- **RFID:** Là phương tiện truyền dữ liệu không tiếp xúc giữa thẻ (tag) và bộ đọc. Thẻ có thể là **chủ động (active)** (có nguồn điện, phạm vi xa hơn) hoặc **thụ động (passive)**. Thẻ RFID có thể rất nhỏ và rẻ, được ứng dụng rộng rãi trong quản lý kho hàng và hộ chiếu.
- **NFC (Near-Field Communication):** Là giao thức truyền thông ở khoảng cách cực ngắn (vài inch), thường được dùng trong thanh toán di động. NFC cũng được dùng để "mồi" (bootstrap) các kết nối băng thông cao hơn như Bluetooth.

5. Mạng cảm biến (Sensor Networks)

- **Định nghĩa:** Là các mạng gồm các cảm biến tự trị phân tán được thiết kế để theo dõi các điều kiện vật lý hoặc môi trường (như lượng mưa, thời tiết).
- **Cấu trúc:** Các thành phần riêng lẻ được gọi là các **nút (nodes)**, đóng vai trò vừa là thiết bị đo lường vừa là nền tảng truyền thông. Phần lớn các mạng này hiện nay được triển khai qua kết nối không dây.

Embedded

Phần **Embedded** (Hệ thống nhúng) trong Chương 6 của tài liệu mô tả các hệ thống chuyên dụng với các đặc điểm chính như sau:

- **Định nghĩa:** Hệ thống nhúng là những hệ thống chuyên dụng, nơi **phần cứng và phần mềm được kết hợp chặt chẽ với nhau** để thực hiện một mục đích cụ thể.
- **Sự khác biệt với máy tính đa năng:** Khác với các máy tính đa năng như máy chủ (servers) hay PC (có thể thực hiện nhiều loại hoạt động khác nhau), hệ thống nhúng được thiết kế để **giải quyết một vấn đề cụ thể**.
- **Đặc điểm vận hành:** Chúng được tạo ra để thực hiện một nhiệm vụ riêng biệt, trong đó các **ràng buộc về thời gian (time-sensitive constraints)** là rất phổ biến.
- **Sự phổ biến:** Hệ thống nhúng có mặt ở khắp mọi nơi, xuất hiện trong nhiều loại thiết bị điện tử từ đồng hồ, máy nghe nhạc/xem video đến các hệ thống điều khiển trong nhà máy, cơ sở hạ tầng và phương tiện giao thông.

- **Mối liên hệ với hệ thống điều khiển:** Các thiết bị thuộc hệ thống điều khiển (như SCADA, PLC, RTU) cũng có thể được coi là một dạng hệ thống nhúng vì chúng được tích hợp vào môi trường vật lý với mục đích duy nhất là cung cấp khả năng điều khiển bằng máy tính trong môi trường đó.

Cloud Architectures

Phần **Cloud Architectures** (Kiến trúc Đám mây) trong Chương 6 của tài liệu mô tả điện toán đám mây là một hình thức kiến trúc phân tán hiện đại với các nội dung chi tiết sau:

1. Định nghĩa và Đặc điểm cốt lõi

- **Định nghĩa:** Điện toán đám mây được coi là một trường hợp cực đoan của mô hình **khách-chủ (client-server)**. Đây là kiến trúc của các dịch vụ có khả năng mở rộng, được cung cấp tự động dựa trên nhu cầu sử dụng.
- **5 Đặc điểm then chốt (theo NIST):**
    - **On-demand self-service (Tự phục vụ theo nhu cầu):** Khách hàng có thể tự thiết lập hoặc hủy dịch vụ mà không cần can thiệp thủ công từ nhà cung cấp.
    - **Broad network access (Truy cập mạng rộng rãi):** Dịch vụ có sẵn qua mạng và truy cập được bằng nhiều thiết bị khác nhau.
    - **Resource pooling (Hợp nhất tài nguyên):** Tài nguyên tính toán được dùng chung cho nhiều khách hàng, giúp tối ưu hóa chi phí.
    - **Rapid elasticity (Co giãn nhanh chóng):** Khả năng tăng hoặc giảm tài nguyên ngay lập tức theo nhu cầu thực tế.
    - **Measured service (Dịch vụ định lượng):** Việc sử dụng tài nguyên được giám sát và đo lường để tính phí chính xác.

2. Các mô hình triển khai (Deployment Models)

Tài liệu trích dẫn chuẩn **NIST SP 800-145** xác định 4 mô hình triển khai chính:

- **Private cloud (Đám mây riêng):** Phục vụ cho một tổ chức duy nhất.

- **Public cloud (Đám mây công cộng):** Phục vụ cho nhiều thực thể khác nhau dùng chung hạ tầng.
- **Community cloud (Đám mây cộng đồng):** Dành cho một nhóm các tổ chức có chung mối quan tâm hoặc mục tiêu.
- **Hybrid cloud (Đám mây hỗn hợp):** Kết hợp từ hai loại đám mây trở lên, liên kết với nhau bằng công nghệ cho phép chia sẻ dữ liệu và ứng dụng.

3. Các mô hình dịch vụ (Service Models)

- **Software as a Service (SaaS - Phần mềm như một dịch vụ):** Phần mềm chạy trên hạ tầng đám mây của nhà cung cấp, người dùng truy cập qua trình duyệt. Khách hàng không quản lý hạ tầng bên dưới, nhưng **bảo mật dữ liệu** là vấn đề cực kỳ quan trọng cần được lên kế hoạch kỹ lưỡng.
- **Platform as a Service (PaaS - Nền tảng như một dịch vụ):** Cung cấp một nền tảng hoàn chỉnh (bao gồm cả IaaS và SaaS) để hỗ trợ quá trình phát triển, thử nghiệm và triển khai phần mềm. Người dùng chỉ cấu hình các cài đặt ứng dụng cụ thể.
- **Infrastructure as a Service (IaaS - Hạ tầng như một dịch vụ):** Cung cấp tài nguyên thô như mạng, lưu trữ và xử lý. Khách hàng có quyền quản lý việc cung cấp các yếu tố này và triển khai các phần mềm tùy ý.

4. Mô hình trách nhiệm chung (Shared Responsibility Model)

Đây là một khái niệm bảo mật quan trọng trong đám mây:

- Cả nhà cung cấp dịch vụ và khách hàng đều có vai trò và trách nhiệm trong các hoạt động bảo mật.
- **Phân định:** Ranh giới trách nhiệm phụ thuộc vào mô hình dịch vụ (SaaS, PaaS hay IaaS). Ví dụ, trong IaaS, khách hàng chịu trách nhiệm nhiều hơn (như quản lý hệ điều hành, ứng dụng), trong khi ở SaaS, nhà cung cấp chịu trách nhiệm phần lớn hạ tầng và nền tảng.
- Khách hàng cần hiểu rõ các điều khoản dịch vụ để biết những gì được nhà cung cấp bảo vệ và những gì mình phải tự thực hiện.

Mobile Applications

Phần **Mobile Applications** (Ứng dụng di động) trong Chương 6 của tài liệu mô tả một loại kiến trúc điện toán độc đáo với các đặc điểm và rủi ro bảo mật cụ thể như sau:

1. Định nghĩa và Đặc tính vận hành

- **Định nghĩa:** Ứng dụng di động là các phần mềm được thiết kế để chạy trên các thiết bị di động như điện thoại thông minh và máy tính bảng.
- **Môi trường độc đáo:** Các ứng dụng này được phát triển dựa trên những ràng buộc bản địa của thiết bị bao gồm **sức mạnh xử lý hạn chế, bộ nhớ hạn chế và khả năng nhập/xuất (I/O) hạn chế**.
- **Sự tiện lợi:** Dù bị hạn chế về phần cứng, chúng lại có lợi thế là **luôn bật (always on)** và **luôn ở bên người dùng**, tạo ra một môi trường tính toán mang tính cá nhân và kiên trì cao.

2. Hệ sinh thái Cửa hàng ứng dụng (App Stores)

Môi trường ứng dụng di động điển hình bao gồm các kho lưu trữ được thiết kế để điều phối việc phân phối phần mềm, gọi là các **App Stores**. Có nhiều hình thức khác nhau:

- **Cửa hàng riêng (Private app stores):** Được thiết lập nội bộ trong các doanh nghiệp.
- **Cửa hàng thương mại của nhà sản xuất:** Ví dụ như Apple, Google, BlackBerry, Nokia.
- **Cửa hàng thương mại bên thứ ba:** Ví dụ như Amazon.

3. Thách thức về Bảo mật

Việc phát triển ứng dụng di động an toàn là một vấn đề đáng quan tâm do các rủi ro đặc thù:

- **Rủi ro phía máy trạm:** Các thiết bị di động đang trở thành giao diện phổ biến để kết nối vào mạng, do đó chúng có thể gây ra rủi ro về kết nối hoặc **rủi ro phía máy trạm (client-side risk)**.
- **Truy cập dữ liệu:** Các ứng dụng di động thường có tiềm năng truy cập vào một lượng lớn thông tin nhạy cảm được lưu trữ trực tiếp trên thiết bị.

- **Mối liên kết công nghệ:** Ứng dụng di động thường được tăng cường bởi các công nghệ khác trong hệ sinh thái tính toán phổ biến như **dịch vụ dựa trên vị trí (location-based), RFID, NFC và mạng cảm biến**, điều này mở rộng thêm phạm vi cần bảo vệ.

Hardware Platform Concerns

Dưới đây là nội dung chi tiết cho phần **Hardware Platform Concerns** (Các vấn đề về nền tảng phần cứng) thuộc Chương 6 của tài liệu:

Các thành phần phần cứng có thể mang lại những mối lo ngại riêng về bảo mật phần mềm. Khi tương tác với các yếu tố như TPM hoặc HSM, việc hiểu rõ công nghệ, ưu điểm và hạn chế của chúng là rất quan trọng.

1. Trusted Platform Module (TPM - Mô-đun nền tảng đáng tin cậy)

- **Định nghĩa:** TPM là một giải pháp phần cứng nằm trực tiếp trên bo mạch chủ (motherboard).
- **Chức năng chính:** Nó hỗ trợ việc **tạo khóa**, **lưu trữ khóa** và **tạo số ngẫu nhiên**. TPM hoạt động như một bộ vi xử lý mật mã an toàn.
- **Lợi ích bảo mật:** Khi các khóa mã hóa được lưu trữ trong TPM, chúng không thể bị truy cập thông qua các kênh phần mềm thông thường và được tách biệt vật lý khỏi ổ cứng hoặc các vị trí lưu trữ dữ liệu mã hóa khác. Điều này làm cho TPM trở thành giải pháp an toàn hơn so với việc lưu khóa trong bộ nhớ thông thường của máy tính.

2. Hardware Security Module (HSM - Mô-đun bảo mật phần cứng)

- **Định nghĩa:** HSM là một thiết bị vật lý được sử dụng để quản lý hoặc lưu trữ các khóa mã hóa.
- **Hỗ trợ tính toán:** Ngoài lưu trữ, HSM còn hỗ trợ các hoạt động mật mã như **mã hóa/giải mã, băm (hashing) và thực hiện chữ ký số**.
- **Kết nối:** HSM thường là các thiết bị ngoại vi kết nối qua cổng USB hoặc kết nối mạng.
- **Ưu điểm:**

- **Chống giả mạo:** Có cơ chế bảo vệ chống giả mạo (tamper-protection) để ngăn chặn việc truy cập vật lý vào các bí mật bên trong.
- **Hiệu suất:** Cung cấp lợi thế đáng kể về tốc độ xử lý và hiệu suất thông lượng cho các doanh nghiệp có khối lượng lớn hoạt động mã hóa.
- **Cách ly rủi ro:** Cho phép sử dụng các khóa mà không làm lộ chúng trước các mối đe dọa từ máy chủ (host-based threats).

3. Side Channel (Tấn công kênh kề)

Đây là các cuộc tấn công khai thác "tác dụng phụ" (byproduct) của hệ thống hoặc cách triển khai hệ thống mật mã, thay vì đánh vào độ mạnh của thuật toán. Các loại chính bao gồm:

- **Timing attacks (Tấn công thời gian):** Phân tích thời gian cần thiết để thực hiện một chức năng nhằm thu thập thông tin về những gì đang diễn ra bên trong chương trình.
- **Power attacks (Tấn công năng lượng):** Phân tích mức tiêu thụ năng lượng của thiết bị khi xử lý dữ liệu.
- **Electromagnetic attacks (Tấn công điện từ):** Đọc các mẫu từ tính hoặc phát xạ điện từ từ thiết bị (như màn hình CRT cũ) để tái tạo dữ liệu.
- **Acoustic attacks (Tấn công âm thanh):** Sử dụng micrô của máy tính để ghi lại âm thanh bàn phím và giải mã dựa trên âm thanh khác nhau của mỗi phím.
- **Data remanence attacks (Tấn công dữ liệu dư thừa):** Khai thác dữ liệu còn sót lại trong bộ nhớ sau khi tắt nguồn. Một ví dụ điển hình là nghiên cứu làm lạnh RAM để kéo dài thời gian lấy các giá trị khóa sau khi mất điện.

Cognitive Computing

Phần **Cognitive Computing (Tính toán nhận thức)** trong Chương 6 mô tả việc ứng dụng các công nghệ tiên tiến để giải quyết các thách thức trong phát triển và bảo mật phần mềm hiện đại. Dưới đây là nội dung chi tiết:

1. Bản chất và Mục tiêu

- **Thành phần cốt lõi:** Tính toán nhận thức dựa trên **trí tuệ nhân tạo (AI)** và **máy học (ML)**.
- **Mục tiêu:** Giải quyết các vấn đề mà các phương pháp thuật toán tiêu chuẩn trước đây không thể xử lý được.

2. Bối cảnh về độ phức tạp của phần mềm

- **Tỷ lệ lỗi:** Ước tính có khoảng **một lỗ hổng trên mỗi 10.000 dòng mã**. Với những chương trình lên tới 10 triệu dòng mã, số lượng vấn đề tiềm ẩn là rất lớn.
- **Sự gia tăng quy mô:** Các chương trình đang ngày càng lớn và phức tạp hơn; ví dụ, một chiếc xe điện cao cấp với tính năng tự lái có thể lên tới **100 triệu dòng mã hoặc hơn**. Điều này đòi hỏi những giải pháp mới để quản lý bảo mật.

3. Những tiến bộ và Thử nghiệm thực tế

- **Thử thách của DARPA:** DARPA đã tạo ra một thử thách lớn (grand challenge) yêu cầu một chương trình máy tính tự tạo ra một chương trình máy tính khác. Đây không chỉ là chương trình đơn giản mà là một giải pháp tấn công và phòng thủ phức tạp cho sự kiện "Capture the Flag".
- **Kết quả:** Dù chưa hoàn hảo, cuộc thi này đã thúc đẩy đáng kể việc sử dụng AI và ML trong cả lập trình và thiết kế hệ thống.

4. Ứng dụng thực tế trong bảo mật phần mềm

Hiện nay, AI đã bắt đầu được tích hợp vào các công cụ phát triển để nâng cao tính an toàn:

- **Phân tích mã nguồn (Source code analysis):** AI giúp **giảm thiểu các cảnh báo giả (false alarms)** và cải thiện khả năng nhận diện các lỗi nghiêm trọng.
- **Kiểm thử Fuzzing:** AI giúp cải thiện các công cụ fuzzer bằng cách **tạo ra các tập dữ liệu thử nghiệm (test data sets) có tính liên quan cao hơn**, giúp tìm ra lỗ hổng hiệu quả hơn.

Control Systems

Phần **Control Systems (Hệ thống điều khiển)** trong Chương 6 của tài liệu mô tả các hệ thống chuyên dụng với các đặc điểm chi tiết sau:

- **Định nghĩa và mục đích:** Hệ thống điều khiển là các hệ thống máy tính chuyên dụng được sử dụng để **điều khiển thiết bị một cách tự động**.
- **Tên gọi phổ biến:** Những hệ thống này thường được gọi bằng nhiều tên khác nhau, bao gồm **hệ thống điều khiển công nghiệp (Industrial Control Systems)** và **công nghệ vận hành (Operational Technology - OT)**.
- **Các loại thiết bị và phần mềm:** Một dải rộng các loại thiết bị và phần mềm hỗ trợ nằm trong danh mục này, tiêu biểu là các **bộ điều khiển logic có thể lập trình (PLC)** và các **đơn vị đầu cuối từ xa (RTU)**.
- **Hệ thống SCADA:** Khi các thiết bị này được sử dụng dưới dạng tập hợp phối hợp, chúng thường được gọi chung là hệ thống **điều khiển giám sát và thu thập dữ liệu (SCADA)**.
- **Mối liên hệ với hệ thống nhúng:** Thiết bị hệ thống điều khiển có thể được coi là một dạng của **hệ thống nhúng (embedded system)** vì chúng được tích hợp vào một môi trường vật lý với mục đích duy nhất là cung cấp khả năng điều khiển bằng máy tính trong môi trường đó.

Chapter 7 Secure Software Design

Introduction

Nội dung phần giới thiệu của **Chương 7: Secure Software Design (Thiết kế phần mềm an toàn)**, trước khi đi vào các phần chi tiết, tập trung vào việc thiết lập nền tảng cho giai đoạn thiết kế trong vòng đời phát triển phần mềm an toàn (SDL) với các điểm chính sau:

1. Mục tiêu và Vai trò của Thiết kế An toàn

- **Xây dựng từ gốc:** Việc triển khai bảo mật bắt đầu từ các yêu cầu và thực sự được tích hợp vào hệ thống nếu được đưa vào ngay trong giai đoạn thiết kế của SDL.

- **Tạo tiền đề cho lập trình:** Việc đưa các yêu cầu bảo mật vào thiết kế giúp các giai đoạn lập trình và triển khai sau đó tạo ra được sản phẩm an toàn hơn.
- **Mục tiêu kép:**
    - Mục tiêu hàng đầu là **giảm thiểu tối đa các lỗ hổng bảo mật** (minimization of vulnerabilities).
    - Mục tiêu thứ hai là **phát triển các lớp phòng thủ** (layered defenses) cho những lỗ hổng vẫn còn tồn tại.
- **Bản thiết kế (Blueprint):** Thiết kế ứng dụng chính là việc tạo ra bản thiết kế mà các lập trình viên sẽ dựa vào đó để xây dựng sản phẩm cuối cùng; đây là lúc các yếu tố nền tảng cho chức năng bảo mật được khởi xướng.

2. Cách tiếp cận trong giai đoạn Thiết kế

Để xác định các yếu tố bảo mật cần thiết cho ứng dụng, các nhà thiết kế sử dụng thông tin từ các giai đoạn trước:

- **Xác định "Cái gì" và "Ở đâu":** Sử dụng dữ liệu từ **phân tích bề mặt tấn công** (attack surface analysis) và **mô hình hóa đe dọa** (threat model) đã được thực hiện ở giai đoạn yêu cầu.
- **Xác định "Làm thế nào":** Áp dụng kiến thức về các **nguyên tắc thiết kế an toàn** (secure design principles).
- **Kết quả:** Một kế hoạch toàn diện sẽ cung cấp cho đội ngũ phát triển một nền tảng vững chắc để tạo ra một ứng dụng an toàn.

3. Các mục tiêu học tập chính của chương (Objectives)

Trước khi bắt đầu phần thiết kế giao diện, chương này liệt kê các nội dung sẽ bao gồm:

- Thực hiện thiết kế giao diện an toàn (Secure interface design).
- Thực hiện đánh giá rủi ro kiến trúc (Architectural risk assessments).
- Mô hình hóa các thuộc tính và ràng buộc bảo mật (phi chức năng).

- Mô hình hóa và phân loại dữ liệu.
- Đánh giá và lựa chọn các thiết kế an toàn có thể tái sử dụng.
- Thực hiện xem xét kiến trúc và thiết kế bảo mật.
- Định nghĩa kiến trúc vận hành an toàn.
- Sử dụng các nguyên tắc, mô hình và công cụ thiết kế/kiến trúc an toàn.

Performing Secure Interface Design

Phần **Performing Secure Interface Design** (Thực hiện thiết kế giao diện an toàn) trong Chương 7 tập trung vào việc bảo vệ các ranh giới nơi dữ liệu đi vào và đi ra khỏi các mô-đun phần mềm. Dưới đây là nội dung chi tiết:

1. Tầm quan trọng của Thiết kế Giao diện An toàn

- **Ranh giới nhập/xuất:** Giao diện được định nghĩa là ranh giới giữa đầu vào (inputs) và đầu ra (outputs). Việc đảm bảo đầu vào hợp lệ là yếu tố then chốt để mọi mô-đun hoạt động chính xác.
- **Nguyên nhân lỗ hổng:** Một trong những nguyên nhân hàng đầu gây ra các lỗ hổng bảo mật là **xác thực đầu vào không đúng cách** (improper input validation), và trách nhiệm này thuộc về khâu thiết kế giao diện.
- **Kiểm soát luồng dữ liệu:** Mọi sự di chuyển dữ liệu qua hệ thống đều có thể bị sửa đổi trái phép hoặc không chính xác, dẫn đến lỗi hệ thống. Thiết kế giao diện phải bảo vệ mô-đun khỏi các vấn đề này.

Logging

Việc thiết kế cách thức truy cập và quản lý dữ liệu nhật ký là một phần của thiết kế giao diện.

- **Các phương thức quản lý:**
    - **In-band management (Quản lý trong băng):** Đơn giản hơn về mặt giao tiếp nhưng có rủi ro là bị lộ diện trước cùng các mối đe dọa như kênh giao tiếp chính (ví dụ: tấn công từ chối dịch vụ hoặc đánh hơi lưu lượng).

    - **Out-of-band management (Quản lý ngoài băng):** Sử dụng một kênh giao tiếp riêng biệt. Ưu điểm là cho phép quản lý hệ thống ngay cả khi ứng dụng chính đang bị tấn công, nhưng nhược điểm là đòi hỏi phải thiết kế thêm các kênh giao tiếp bảo mật riêng.

- **Tích hợp hệ thống doanh nghiệp:** Thay vì thiết kế giao diện báo cáo riêng lẻ, việc gửi thông tin đến hệ thống quản lý an ninh tập trung của doanh nghiệp (như **SIEM**) giúp nhân viên an ninh dễ dàng đối soát và phân tích dữ liệu.
- **Thách thức thiết kế nhật ký:** Cần xác định **cái gì cần ghi nhật ký** (dựa trên tuân thủ như HIPAA, SOX, PCI DSS và nhu cầu ứng phó sự cố) và **lưu trữ ở đâu/trong bao lâu**.

Protocol Design Choices

- **API là Hợp đồng:** Các giao diện lập trình ứng dụng (APIs) được coi là các "hợp đồng" quy định cách các mô-đun phần mềm giao tiếp với nhau.
- **Quản lý rủi ro qua API:** Việc lựa chọn API là một vấn đề kiến trúc và thiết kế, vì các kết nối giữa các mô-đun có thể làm tăng **bề mặt tấn công** (attack surface area).
- **Tính linh hoạt:** API cho phép thay đổi các chức năng bên ngoài mà không gây gián đoạn lớn cho dự án phần mềm gốc, hỗ trợ phát triển và triển khai nhanh chóng. Tuy nhiên, mọi quyết định lựa chọn giao diện phải dựa trên rủi ro tổng thể của hệ thống.

4. Các yêu cầu bổ sung

- **Xác thực và Kiểm toán:** Mọi đầu vào giao diện đi vào ứng dụng đều cần có mức độ xác thực phù hợp. Ngoài ra, cần kiểm toán (audit) các tương tác bên ngoài đối với bất kỳ hoạt động đặc quyền nào được thực hiện qua giao diện.

Performing Architectural Risk Assessment (Thực hiện đánh giá rủi ro kiến trúc)

- **Bản chất:** Về cốt lõi, đây là một bài tập **quản trị rủi ro**.

- **Mục tiêu:** Hoạt động này nhằm xác định các lỗi (flaws) trong kiến trúc phần mềm và các rủi ro liên quan đến hệ thống thông tin của tổ chức.
- **Quy trình:** Sử dụng các phương pháp đánh giá rủi ro tiêu chuẩn để làm điểm bắt đầu cho việc đánh giá phần mềm sắp được phát triển. Các lỗi phát hiện được sẽ được **ưu tiên dựa trên mức độ tác động** của chúng đối với hệ thống.
- **Biện pháp khắc phục:** Đối với những rủi ro vượt quá ngưỡng cho phép, các kế hoạch giảm thiểu sẽ được xây dựng và triển khai như một phần của quá trình phát triển.
- **Tầm quan trọng của thời điểm:** Việc hiểu rõ rủi ro ở giai đoạn sớm giúp đưa ra các giải pháp khắc phục ngay từ thiết kế, tránh ảnh hưởng đến tiến độ giao hàng. Ngược lại, nếu phát hiện rủi ro muộn, việc vá lỗi thường dẫn đến tăng chi phí và chậm trễ lịch trình.

Model (Nonfunctional) Security Properties and Constraints (Mô hình hóa các thuộc tính và ràng buộc bảo mật phi chức năng)

- **Định nghĩa:** Bên cạnh các yêu cầu chức năng (functional) xác định phần mềm phải làm gì với dữ liệu, còn có các **yêu cầu phi chức năng (nonfunctional)** xác định các thuộc tính hệ thống.
- **Các thuộc tính chính:** Bao gồm bảo mật (security), độ tin cậy (reliability), khả năng phục hồi (resilience), hiệu năng (performance), khả năng bảo trì (maintainability), khả năng mở rộng (scalability) và khả năng sử dụng (usability).
- **Vai trò:** Các yêu cầu này đóng vai trò là các **ràng buộc hoặc hạn chế** đối với việc thiết kế hệ thống.
- **Tầm quan trọng:** Yêu cầu phi chức năng quan trọng ngang hàng với yêu cầu chức năng. Đặc biệt, những thất bại liên quan đến các yêu cầu tuân thủ (compliance-related requirements) có thể gây ra các vấn đề pháp lý nghiêm trọng cho tổ chức.

Model and Classify Data (Mô hình hóa và phân loại dữ liệu)

Dữ liệu là lý do tồn tại của phần mềm; đó là nguyên liệu mà phần mềm xử lý, hiển thị và lưu trữ. Việc hiểu rõ các loại dữ liệu và hạn chế đi kèm sẽ ảnh hưởng trực tiếp đến thiết kế chương trình. Dữ liệu được chia thành hai loại chính:

Structured Data (Dữ liệu có cấu trúc):

- Có cấu trúc xác định và được quản lý thông qua chính các cấu trúc đó.
- Hình thức phổ biến nhất là dữ liệu trong **cơ sở dữ liệu quan hệ (relational databases)** được sắp xếp theo các bảng.
- Các dạng khác bao gồm: file được định dạng sẵn, dữ liệu XML, JSON và các file văn bản nhất định như file nhật ký (log files).
- Cấu trúc này cho phép các trình phân tách (parsers) dễ dàng duyệt qua để sắp xếp và tìm kiếm dữ liệu.
- **Lưu ý:** Dữ liệu có cấu trúc được xác định dựa trên cách tổ chức (mối quan hệ giữa các phần tử dữ liệu), chứ không phải dựa trên cơ chế lưu trữ (vì dữ liệu phi cấu trúc vẫn có thể được lưu trong cơ sở dữ liệu).

Unstructured Data (Dữ liệu phi cấu trúc):

- Bao gồm tất cả các dữ liệu còn lại trong hệ thống như tài liệu văn phòng (Word, PDF), bảng tính (Spreadsheets) và email.
- Cấu trúc của chúng thường không đều, khó phân tích và tìm kiếm nếu không ở trong ứng dụng gốc.
- Mặc dù chiếm đại đa số dữ liệu trong hầu hết các công ty, loại dữ liệu này rất **khó điều hướng và quản lý** nếu không có các công cụ lưu trữ và khám phá (discovery tools) chuyên dụng của doanh nghiệp.

Evaluate and Select Reusable Secure Design

Nội dung phần giới thiệu của mục **Evaluate and Select Reusable Secure Design** (Đánh giá và lựa chọn thiết kế an toàn có thể tái sử dụng) trong Chương 7 mô tả việc sử dụng lại

các thành phần phần mềm là một phương pháp mang lại nhiều lợi ích nhưng cũng đi kèm rủi ro đáng kể, cụ thể như sau:

1. Khái niệm và Lịch sử

- **Mô hình sản xuất:** Khái niệm tổng quát về tính tái sử dụng (reusability) đã tồn tại từ rất lâu trong sản xuất, nhưng việc tái sử dụng mã nguồn (code reuse) mới thực sự phổ biến trong ngành phần mềm khoảng 20 năm trở lại đây.
- **Cấp độ máy tính:** Ở cấp độ máy, việc tái sử dụng mã là nguyên tắc nền tảng của các trình biên dịch từ những năm 1950; ví dụ điển hình là các thư viện mẫu chuẩn (Standard Template Libraries - STLs) trong các ngôn ngữ như C++.
- **Định nghĩa cấp độ ứng dụng:** Tái sử dụng mã đơn giản là việc xây dựng phần mềm mới từ các thành phần hiện có, có thể là từ các dự án trước đó hoặc từ các thư viện chức năng dùng chung.

2. Lợi ích của việc tái sử dụng

- **Tính kinh tế:** Đây là một cách tiếp cận logic giúp tiết kiệm thời gian và giảm chi phí phát triển.
- **Kiểm soát chất lượng:** Việc sử dụng các hàm, bản mẫu hoặc thủ tục đã được viết sẵn giúp đảm bảo kiểm soát chất lượng và thiết lập một mức năng lực tiêu chuẩn cho mã nguồn được tạo ra.

3. Rủi ro và Thách thức

- **Rủi ro lan truyền:** Vấn đề lớn nhất là nếu một mô đun tái sử dụng bị xâm nhập hoặc có lỗi, nó có thể gây ra rủi ro cho hàng loạt ứng dụng sử dụng mô đun đó sau này.
- **Quản trị rủi ro chuỗi cung ứng:** Theo quan điểm quản lý rủi ro chuỗi cung ứng truyền thống, việc xác định rõ ràng các tình huống và cách dùng cụ thể mà một mô đun tái sử dụng có thể được áp dụng một cách an toàn và hợp pháp là cực kỳ quan trọng.

Creating a Practical Reuse Plan

Phần **Creating a Practical Reuse Plan** (Tạo một kế hoạch tái sử dụng thực tế) trong Chương 7 tập trung vào việc thiết lập các quy trình để quản lý rủi ro khi sử dụng lại các thành phần phần mềm. Dưới đây là nội dung chi tiết:

1. Mục tiêu và Nguyên tắc cốt lõi

- **Mục tiêu chính:** Phát triển một **thư viện an toàn gồm các thành phần có thể tái sử dụng**, tương tự như kho phụ tùng vật lý, nhằm tận dụng năng lực sản xuất và đảm bảo chất lượng.
- **Đảm bảo tính chính xác:** Mã nguồn được tái sử dụng phải được đảm bảo là chính xác để tránh việc lặp lại các lỗi logic và cú pháp trước đó.
- **Các nguyên tắc hỗ trợ:** Việc tái sử dụng được hỗ trợ bởi các nguyên tắc thiết kế phần mềm cơ bản như **tính mô-đun (modularity), che giấu thông tin (information hiding) và giảm thiểu sự phụ thuộc (decoupling)**. Các nguyên tắc này giúp mã nguồn có thể được xác nhận là hoạt động đúng đắn và an toàn.

2. Quản trị Rủi ro và Hợp đồng

- **Sự cần thiết của Quản trị rủi ro:** Vì mã tái sử dụng thường có nguồn gốc từ bên thứ ba hoặc các nguồn chưa biết, việc sử dụng chúng tiềm ẩn rủi ro cao. Do đó, các hoạt động quản lý rủi ro thực tế phải được áp dụng chặt chẽ.
- **Điều khoản hợp đồng:** Các điều khoản sử dụng mã tái sử dụng phải được nêu rõ trong hợp đồng, bao gồm cả các thỏa thuận về **trách nhiệm pháp lý đối với tổn thất hoặc thiệt hại** nếu có vấn đề phát sinh.

3. Quy hoạch và Chiến lược

- **Lập kế hoạch chiến lược:** Việc tái sử dụng mã phải được dẫn dắt bởi một hoạt động lập kế hoạch chiến lược, thường là một phần của **kế hoạch quản lý rủi ro tổng thể**.
- **Các yêu cầu trong kế hoạch:** Kế hoạch cần nêu rõ việc tái sử dụng có liên quan đến **mã nguồn mở**, mã tái sử dụng khác, hay các sản phẩm/dịch vụ giá trị gia tăng hay không.

- **Hỗ trợ ra quyết định:** Quy trình lập kế hoạch giúp những người ra quyết định đánh giá được lợi thế của mã tái sử dụng so với việc tự phát triển mới, cũng như cách bảo vệ phần mềm trước các rủi ro.

4. Quy trình Kiểm thử và Xác thực

- **Kiểm thử bảo mật:** Sau khi các rủi ro đã được ưu tiên và lập hồ sơ, tổ chức cần thực hiện quy trình **kiểm thử bảo mật quản lý rủi ro** với sự tham gia của các bên liên quan (lập trình viên, người quản lý tài sản, chuyên gia lĩnh vực và người dùng).
- **Chuyển giao và Thực hiện:** Khi kiểm thử hoàn tất và kết quả được chấp nhận, mô tả về các rủi ro được xác định cùng các biện pháp giảm thiểu sẽ được chuyển cho các quản lý phát triển để áp dụng vào quy trình tái sử dụng.
- **Tính lặp lại:** Vì các rủi ro mới xuất hiện liên tục, đây là một **quy trình lặp lại** theo chu kỳ phù hợp với môi trường rủi ro.

Credential Management

Phần **Credential Management** (Quản lý thông tin xác thực) trong Chương 7 tập trung vào việc bảo vệ và điều hành các thông tin nhạy cảm dùng để xác minh danh tính người dùng. Dưới đây là nội dung chi tiết dựa trên tài liệu:

1. Định nghĩa và Tầm quan trọng

- **Khái niệm:** Thông tin xác thực (credentials) là dữ liệu định danh mà người dùng cung cấp để khẳng định mình là người dùng hợp lệ.
- **Tính nhạy cảm:** Đây là dữ liệu nhạy cảm đòi hỏi sự bảo vệ nghiêm ngặt.
- **Các hình thức phổ biến:** Credentials có thể tồn tại dưới dạng bí mật được truyền đi (thường là mật khẩu), chuỗi kỹ thuật số trong thẻ bài phần cứng (hardware tokens), thiết bị, dữ liệu sinh trắc học hoặc chứng chỉ kỹ thuật số.

2. Các nhiệm vụ Quản lý

Việc quản lý các bộ thông tin xác thực này, bất kể nguồn gốc từ đâu, đều yêu cầu sự lưu giữ an toàn từ phía thực thể tiếp nhận và bao gồm các nhiệm vụ sau:

- **Tạo mới (Generation):** Thiết lập thông tin xác thực ban đầu.

- **Lưu trữ (Storage):** Đảm bảo an toàn cho dữ liệu khi không sử dụng.
- **Đồng bộ hóa (Synchronization):** Giữ cho thông tin thống nhất giữa các hệ thống.
- **Đặt lại (Reset) và Thu hồi (Revocation):** Xử lý khi thông tin bị quên, bị mất hoặc không còn quyền truy cập.
- **Ghi nhật ký (Logging):** Do tính chất nhạy cảm của việc thao tác với thông tin xác thực, tất cả các hoạt động quản lý trên đều phải được ghi lại trong nhật ký hệ thống.

3. Thông tin xác thực X.509 (X.509 Credentials)

X.509 là một chuỗi các tiêu chuẩn liên quan đến việc thao tác với các chứng chỉ kỹ thuật số dùng để chuyển giao các khóa bất đối xứng một cách có thể xác minh được.

- **Chức năng:** Chứng chỉ kỹ thuật số liên kết danh tính của một cá nhân với một khóa công khai (public key).
- **Cấu trúc hạ tầng khóa công khai (PKI):**
    - **Registration Authority (RA):** Cơ quan đăng ký thực hiện xác minh danh tính cá nhân.
    - **Certificate Authority (CA):** Cơ quan chứng nhận tạo ra chứng chỉ kỹ thuật số sau khi danh tính đã được RA xác minh.
    - **Cơ chế thu hồi:** Sử dụng Danh sách thu hồi chứng chỉ (CRLs) hoặc Giao thức trạng thái chứng chỉ trực tuyến (OCSP) để kiểm tra tính hợp lệ của chứng chỉ trước khi hết hạn.
- **Các trường dữ liệu chính trong chứng chỉ X.509:** Bao gồm số phiên bản, số sê-ri duy nhất, thuật toán chữ ký, thông tin thực thể cấp phát (Issuer), thời hạn hiệu lực, chủ sở hữu chứng chỉ (Subject), khóa công khai và các phần mở rộng (Extensions).

4. Đăng nhập một lần (Single Sign-On - SSO)

SSO cho phép người dùng, sau khi được xác thực lần đầu, có thể tái sử dụng thông tin xác thực cho các ứng dụng khác mà không cần nhập lại bí mật.

- **Cơ chế:** Thông tin xác thực được lưu trữ bên ngoài ứng dụng và được tái sử dụng để đối soát với các hệ thống khác. Các giao thức phổ biến bao gồm Kerberos, SAML và OpenID.
- **Liên minh (Federation):** Đây là khái niệm then chốt của SSO, cho phép người dùng đăng nhập vào một trang web và truy cập vào các trang liên kết khác mà không cần nhập lại thông tin.
- **Mục tiêu và Rủi ro:**
    - Mục tiêu chính là sự tiện lợi cho người dùng.
    - Rủi ro: Việc thiết lập niềm tin liên minh rất khó khăn; các hệ thống dựa trên SSO có thể tạo ra kịch bản **"điểm yếu duy nhất" (single-point-of-failure)**, do đó không được khuyến nghị cho một số triển khai có rủi ro cao.

Data Loss Prevention

Phần **Data Loss Prevention (DLP - Ngăn ngừa mất dữ liệu)** trong Chương 7 mô tả một lớp bảo mật quan trọng tập trung trực tiếp vào việc bảo vệ tài sản quý giá nhất của tổ chức: dữ liệu. Dưới đây là nội dung chi tiết:

1. Vai trò và Vị trí

- **Hàng phòng thủ cuối cùng:** DLP được coi là hàng phòng thủ cuối cùng (last line of defense) trong chiến lược bảo mật.
- **Mục tiêu bảo vệ:** Trong khi các tài sản khác như thiết bị, ứng dụng và các biện pháp kiểm soát đều quan trọng, mục tiêu cuối cùng của chúng là để bảo vệ dữ liệu của tổ chức.

2. Cơ chế hoạt động

Các giải pháp DLP hoạt động bằng cách **sàng lọc lưu lượng truy cập (screening traffic)** dựa trên các thông số hồ sơ (profile parameters) đã được thiết lập trước. Các thông số này bao gồm:

- **Quy mô chuyển giao (Size of transfer):** Theo dõi các lần truyền dữ liệu có dung lượng bất thường.

- **Đích đến (Destination):** Kiểm tra dữ liệu đang được gửi tới đâu.
- **Thành phần dữ liệu cụ thể (Specific data elements):** Nhận diện các mẫu dữ liệu nhạy cảm cần được bảo vệ.

Hành động xử lý: Nếu hệ thống phát hiện lưu lượng khớp với các thông số rủi ro, nó sẽ xác định đó là một sự kiện rò rỉ dữ liệu (exfiltration event) đang diễn ra và thực hiện **chấm dứt kết nối** ngay lập tức.

3. Các thách thức trong triển khai

Mặc dù đơn giản về mặt lý thuyết, việc triển khai DLP rất phức tạp, đặc biệt là trong môi trường phòng thủ theo chiều sâu (defense-in-depth):

- **Vị trí phát hiện (Detection location):** Công nghệ DLP cần phải nằm trực tiếp trên đường dẫn lưu lượng mạng (netflow path) thực tế của quá trình chuyển giao dữ liệu. Việc này dễ dàng trong các mạng đơn giản nhưng cực kỳ thách thức và tốn kém trong các doanh nghiệp lớn với vô số kết nối bên ngoài.
- **Khả năng hiển thị dữ liệu (Visibility):** Những kẻ tấn công thường sử dụng mã hóa để ẩn giấu các luồng dữ liệu, khiến hệ thống DLP khó phát hiện.
- **Điện toán đám mây:** Việc di chuyển các dịch vụ và dữ liệu lên đám mây khiến quá trình kiểm soát và ngăn ngừa mất dữ liệu trở nên phức tạp hơn nhiều.

Virtualization

Phần **Virtualization (Ảo hóa)** trong Chương 7 của tài liệu cung cấp các thông tin chi tiết về vai trò, lợi ích và tầm quan trọng của công nghệ này trong thiết kế phần mềm an toàn:

- **Vị trí và Chức năng:** Ảo hóa thiết lập một **lớp trung gian (virtualization layer)** nằm giữa phần cứng và hệ điều hành. Lớp này cho phép nhiều hệ điều hành chạy đồng thời trên cùng một thiết bị phần cứng vật lý.
- **Lợi ích Bảo mật:**

    - Cung cấp khả năng **cô lập (isolation)**: Nếu người dùng tải xuống nội dung độc hại khi lướt web, máy ảo đó có thể bị xóa ngay sau phiên làm việc, giúp ngăn chặn sự lây lan của mã độc sang các hệ điều hành khác.
    - Tăng cường tính linh hoạt trong vận hành.
- **Các nhà cung cấp chính:** Các phần mềm ảo hóa phổ biến bao gồm **VMware, Microsoft, Oracle và Xen**.
- **Lợi ích đối với Tổ chức:**
    - **Giảm chi phí:** Nhờ vào việc hợp nhất máy chủ (server consolidation).
    - **Hiệu quả vận hành:** Cải thiện hiệu suất quản trị cho các tác vụ nhất định.
    - **Tính di động và Cô lập:** Cải thiện khả năng di chuyển và cô lập ứng dụng, dữ liệu cũng như nền tảng.
    - **Sự linh hoạt (Agility):** Giúp mở rộng môi trường nhanh chóng, ví dụ như trong điện toán đám mây.
- **Tầm quan trọng đối với đội ngũ phát triển:** Vì máy ảo (VM) đã trở thành nền tảng chủ đạo trong nhiều doanh nghiệp, đội ngũ phát triển cần hiểu rõ các tác động của môi trường ảo hóa đối với ứng dụng của mình, đặc biệt nếu ứng dụng đó có khả năng được triển khai trên máy ảo.

Trusted Computing

Phần **Trusted Computing (Tính toán tin cậy)** trong Chương 7 tập trung vào các công nghệ phần cứng được thiết kế để đảm bảo máy tính hoạt động một cách nhất quán và đúng như mong đợi. Dưới đây là nội dung chi tiết:

1. Khái niệm Trusted Computing (TC)

- **Định nghĩa:** Đây là thuật ngữ dùng để mô tả các công nghệ được phát triển và thúc đẩy bởi **Trusted Computing Group**.
- **Mục tiêu:** Đảm bảo hệ thống máy tính hành xử theo cách thức ổn định, có thể dự đoán và không bị thay đổi ngoài ý muốn.

- **Thành phần cốt lõi:** Một yếu tố then chốt trong nỗ lực này là **Trusted Platform Module (TPM)**, đóng vai trò là giao diện phần cứng cho các hoạt động bảo mật.

2. Trusted Computing Base (TCB - Cơ sở tính toán tin cậy)

- **Thành phần:** TCB là tập hợp tất cả các thành phần phần cứng, firmware và phần mềm có vai trò thiết yếu đối với tính bảo mật của hệ thống.
- **Vấn đề chính:** Mối quan tâm hàng đầu của TCB là ngăn chặn **leo thang đặc quyền (privilege escalation)**. Nếu bất kỳ phần nào của hệ thống có khả năng tự tăng đặc quyền mà không được phép, nó sẽ vi phạm chính sách bảo mật và phải được coi là một phần của TCB.
- **Tầm quan trọng:** Khái niệm TCB tạo nền tảng cho các nguyên tắc bảo mật quan trọng, chẳng hạn như **trung gian hoàn toàn (complete mediation)**.

3. Trusted Platform Module (TPM)

TPM là một bảng mạch phần cứng tích hợp các chức năng mã hóa trực tiếp trên bo mạch chủ của máy tính.

- **Chức năng:** Cung cấp mức độ bảo mật sâu hơn hệ điều hành và gần như không thể bị can thiệp từ phía phần mềm.
- **Khả năng bảo mật:** TPM có thể lưu giữ các khóa mã hóa mà hệ thống không thể truy cập được ngoại trừ thông qua chính chip TPM, giúp bảo vệ các bí mật ngay cả khi hệ điều hành bị xâm nhập.
- **Các tính năng phần cứng (Hình 7-2):**
    - **Bộ xử lý mật mã:** Bao gồm bộ tạo số ngẫu nhiên (RNG), bộ tạo khóa RSA, bộ tạo băm SHA1 và công cụ ký mã hóa/giải mã.
    - **Bộ nhớ linh hoạt (Versatile memory):** Chứa các thanh ghi cấu hình nền tảng, khóa định danh xác thực và các khóa lưu trữ.
    - **Bộ nhớ bền vững (Persistent memory):** Chứa khóa gốc lưu trữ (storage root key) và khóa chứng thực (endorsement key) của nhà sản xuất.

- **Tranh luận:** Công nghệ này từng gây tranh cãi vì lo ngại nó có thể được dùng để kiểm soát quyền sở hữu máy tính hoặc hạn chế loại phần mềm được phép chạy trên hệ thống.

Database Security

Phần **Database Security (An ninh Cơ sở dữ liệu)** trong Chương 7 tập trung vào các cơ chế bảo vệ dữ liệu khi lưu trữ và thao tác trong hệ quản trị cơ sở dữ liệu (DBMS). Dưới đây là nội dung chi tiết:

1. Tổng quan về An ninh Cơ sở dữ liệu

Cơ sở dữ liệu là công nghệ dùng để lưu trữ và thao tác dữ liệu, thường được tổ chức dưới dạng bảng trong các cơ sở dữ liệu quan hệ. An ninh cơ sở dữ liệu bao gồm các thành phần đảm bảo tính bảo mật (confidentiality), tính toàn vẹn (integrity) và tính sẵn sàng (availability). Mặc dù việc thiết kế chi tiết môi trường cơ sở dữ liệu có thể nằm ngoài phạm vi chuyên môn của một CSSLP, nhưng việc hiểu rõ các khả năng của nó là rất quan trọng.

2. Mã hóa (Encryption)

Dữ liệu lưu trữ trong cơ sở dữ liệu là mục tiêu béo bở cho những kẻ tấn công, giống như kho tiền trong ngân hàng.

- **Cơ chế:** Mã hóa dữ liệu tĩnh (at rest) là biện pháp kiểm soát ngăn chặn có thể được thực hiện thông qua các chức năng bản địa của DBMS hoặc sử dụng tài nguyên mật mã bên ngoài.
- **Lưu ý quan trọng về Khóa chính (Primary keys):** Các khóa chính được sử dụng để lập chỉ mục và liên kết các bảng, do đó **không thể bị che giấu hoặc mã hóa**. Đây là lý do không nên sử dụng thông tin định danh cá nhân (PII) hoặc thông tin sức khỏe cá nhân (PHI) làm khóa chính.
- **Chiến lược mã hóa:** Cần xem xét các yếu tố như mức độ phân loại rủi ro, mô hình sử dụng dữ liệu (trong quá trình truyền tải và sử dụng), tiêu chuẩn doanh nghiệp và các tùy chọn mã hóa có sẵn cho đội ngũ phát triển.

- **Phân tách dữ liệu:** Nếu chỉ một vài cột là nhạy cảm, việc phân tách dữ liệu có thể giúp bảo vệ các phần nhỏ dữ liệu một cách hiệu quả hơn.

3. Trình kích hoạt (Triggers)

Triggers là các hoạt động cơ sở dữ liệu cụ thể được **tự động thực thi** để phản ứng với các sự kiện nhất định (ví dụ: thêm hoặc thay đổi một bản ghi). Chúng cung cấp sự linh hoạt để tự động hóa bất kỳ tác vụ nào, chẳng hạn như ghi nhật ký (logging) hoặc tích hợp các quy tắc nghiệp vụ.

4. Khung nhìn (Views)

Views là các bản trích xuất dữ liệu được thiết kế theo chương trình từ một loạt các bảng.

- **Lợi ích bảo mật:** View cho phép hiển thị một tập hợp con dữ liệu dựa trên các quy tắc nghiệp vụ. Ví dụ, một view có thể chỉ cung cấp các cột địa chỉ giao hàng cho quy trình vận chuyển mà không tiết lộ thông tin thẻ tín dụng hoặc PII.
- **Nguyên tắc:** "Không thể tiết lộ những gì không có ở đó".

5. Quản lý đặc quyền (Privilege Management)

Cơ sở dữ liệu có các cơ chế kiểm soát truy cập nội bộ tương tự như danh sách kiểm soát truy cập (ACL) của hệ thống tệp. Các cơ chế này có thể được liên kết với hệ thống ủy quyền của doanh nghiệp thông qua các **vai trò (roles)** được định nghĩa trong hệ thống cơ sở dữ liệu.

6. Quản lý đồng thời (Concurrency Control)

Khi có hàng ngàn khách hàng truy cập đồng thời, DBMS cần quản lý để đảm bảo an toàn và dự đoán được kết quả thông qua các kiểm soát đồng thời. Các vấn đề cần tránh bao gồm:

- **Lost updates (Mất cập nhật):** Xảy ra khi nhiều giao dịch cùng chọn một hàng và cập nhật, nhưng chỉ có một giao dịch được ghi nhận (committed).
- **Dirty read (Đọc dữ liệu "rác"):** Một giao dịch chọn một hàng đã được cập nhật bởi giao dịch khác nhưng chưa được ghi nhận.
- **Nonrepeatable reads (Đọc không thể lặp lại):** Một giao dịch truy cập cùng một hàng nhiều lần và nhận được dữ liệu khác nhau mỗi lần.

- **Incorrect summary (Tóm tắt sai):** Xảy ra khi một giao dịch đang tính toán tóm tắt các giá trị thì một giao dịch khác cập nhật một vài giá trị trong số đó, dẫn đến kết quả tóm tắt không còn chính xác.

Programming Language Environment

Phần **Programming Language Environment** (Môi trường ngôn ngữ lập trình) trong Chương 7 tập trung vào việc hiểu cách các ngôn ngữ lập trình khác nhau ảnh hưởng đến tính bảo mật và cách mã nguồn được chuyển đổi để thực thi trên hệ thống. Dưới đây là nội dung chi tiết:

1. Tổng quan về việc thực thi mã

Các nhà phát triển sử dụng ngôn ngữ lập trình để viết mã nguồn, nhưng mã này hiếm khi là thứ thực sự chạy trên máy tính mục tiêu. Mã nguồn thường được chuyển đổi thành mã vận hành thông qua các trình biên dịch (**compilers**), trình thông dịch (**interpreters**) hoặc sự kết hợp của cả hai. Việc lựa chọn ngôn ngữ phát triển thường dựa trên các yêu cầu cụ thể của ứng dụng và kỹ năng của đội ngũ phát triển.

2. Trình biên dịch và Liên kết (Compiling & Linking)

Quá trình biên dịch bao gồm hai bước nhỏ: biên dịch (chuyển mã nguồn thành mã đặc thù của bộ xử lý) và liên kết (kết nối các phần tử chương trình như thư viện và tệp phụ thuộc). Việc liên kết có hai hình thức chính:

- **Liên kết tĩnh (Static linking):** Sao chép tất cả các thành phần cần thiết vào tệp thực thi cuối cùng. Điều này giúp thực thi nhanh hơn và dễ phân phối hơn nhưng làm tăng kích thước tệp.
- **Liên kết động (Dynamic linking):** Chỉ đặt tên và vị trí của các thành phần phụ thuộc vào mã; chúng sẽ được giải quyết khi chương trình được nạp vào bộ nhớ lúc chạy (runtime). Cách này tạo ra tệp nhỏ hơn nhưng có rủi ro nếu các chương trình phụ thuộc bị chiếm quyền điều khiển.

3. Tính an toàn về kiểu (Type Safety)

Đây là khả năng của ngôn ngữ lập trình trong việc ngăn chặn các lỗi do xung đột kiểu dữ liệu. Các ngôn ngữ thực thi **Type Safety** mạnh mẽ sẽ bắt buộc các kiểu dữ liệu phải khớp

nhau, nếu không sẽ phát sinh lỗi. Việc lập trình an toàn về kiểu giúp tự động giải quyết nhiều vấn đề liên quan đến quản lý bộ nhớ vì nó xác định rõ độ dài của các biến.

4. Trình thông dịch và Giải pháp hỗn hợp

- **Trình thông dịch:** Sử dụng một chương trình trung gian để thực hiện mã nguồn theo từng dòng tại thời điểm chạy. Cách này cho phép thay đổi mã nhanh chóng (không cần biên dịch lại) nhưng thực thi chậm hơn.
- **Giải pháp hỗn hợp (Hybrid):** Mã nguồn được biên dịch thành một giai đoạn trung gian, sau đó được thông dịch tại thời điểm chạy. Hai hệ thống phổ biến nhất là **Java** và **Microsoft .NET**.

5. Managed Code và Unmanaged Code

- **Managed Code (Mã được quản lý):** Chạy trong một môi trường trung gian (như JVM của Java hoặc CLR của .NET) cung cấp nhiều kiểm soát bảo mật như: sandboxing, dọn rác (garbage collection), kiểm tra chỉ mục và quản lý bộ nhớ tự động. Điều này giúp giảm thiểu các rủi ro như tràn bộ đệm (buffer overflow).
    - **CLR (.NET):** Biên dịch mã thành ngôn ngữ trung gian chung (CIL/MSIL) và thực thi thông qua trình biên dịch Just-In-Time.
    - **JVM (Java):** Biên dịch mã thành "byte code" và chạy trên máy ảo Java.
- **Unmanaged Code (Mã không được quản lý):** Thực thi trực tiếp trên hệ điều hành mục tiêu. Nó có ưu thế lớn về hiệu suất nhưng lập trình viên phải tự chịu trách nhiệm về quản lý bộ nhớ, cấp phát tài nguyên và an toàn kiểu. Điều này khiến nó dễ gặp lỗi rò rỉ bộ nhớ hoặc ghi đè con trỏ.

6. Các cơ chế bảo vệ bổ sung

- **Compiler Switches (Cờ biên dịch):** Cho phép đội ngũ phát triển kiểm soát cách trình biên dịch xử lý các khía cạnh như bảo vệ ngăn xếp (stack protection) và xử lý ngoại lệ. Ví dụ: cờ `/GS` giúp ngăn chặn các cuộc tấn công tràn ngăn xếp. Đội ngũ bảo mật nên xác định các tùy chọn switch này như một yêu cầu bảo mật bắt buộc.

- **Sandboxing:** Thực thi mã trong một môi trường cách ly để ngăn mã tiếp xúc trực tiếp với hệ thống mục tiêu. Kỹ thuật này thường dùng cho các mã không đáng tin cậy hoặc chưa được xác minh.
- **Locality (Nguyên lý cục bộ):** Các truy cập bộ nhớ thường có xu hướng gần với các tham chiếu trước đó. Các cuộc tấn công bộ nhớ thường lợi dụng nguyên lý này, và các biện pháp bảo vệ như **ASLR** (Address Space Layout Randomization) được thiết kế để chống lại chúng bằng cách xáo trộn vị trí bộ nhớ.

Operating System Controls and Services

Phần **Operating System Controls and Services** (Kiểm soát và Dịch vụ Hệ điều hành) trong Chương 7 của tài liệu cung cấp cái nhìn tổng quan về vai trò của hệ điều hành (OS) trong việc hỗ trợ và bảo vệ ứng dụng. Dưới đây là nội dung chi tiết:

1. Định nghĩa và Vai trò cơ bản

- **Tập hợp phần mềm trung gian:** Hệ điều hành là một tập hợp các phần mềm đóng vai trò trung gian giữa các chương trình ứng dụng và tài nguyên phần cứng của máy tính.
- **Giao diện chức năng:** OS cung cấp một giao diện chức năng cho tất cả các dịch vụ được kích hoạt bởi phần cứng.
- **Môi trường thực thi:** Nó tạo ra môi trường nơi các ứng dụng thực thi, cung cấp cho chúng các tài nguyên cần thiết để hoạt động.

2. Sự đa dạng của Hệ điều hành

- **Đa nền tảng:** Hệ điều hành tồn tại cho tất cả các nền tảng, từ máy tính lớn (mainframes), máy tính cá nhân (PCs) cho đến các thiết bị di động.
- **Phân loại theo mục đích:** Có nhiều loại hệ điều hành khác nhau, mỗi loại được hướng tới một mục đích cụ thể:
    - **Hệ thống đa người dùng (Multi-user):** Được thiết kế để quản lý nhiều tiến trình của người dùng, giữ cho chúng tách biệt và quản lý các quyền ưu tiên.

- **Hệ thống thời gian thực (Real-time) và Hệ thống nhúng (Embedded):** Được thiết kế đơn giản và tinh gọn hơn để phù hợp với môi trường đặc thù của chúng.

3. Tầm quan trọng đối với Bảo mật Thiết kế

Mặc dù phần này trong tài liệu khá ngắn gọn, nó nhấn mạnh rằng việc hiểu các dịch vụ và cơ chế kiểm soát của OS là một phần thiết yếu trong thiết kế phần mềm an toàn. Ứng dụng phải dựa vào các dịch vụ của OS để tương tác với phần cứng, và OS chịu trách nhiệm cô lập các tiến trình để đảm bảo an ninh cho toàn bộ hệ thống.

Secure Backup and Restoration Planning

Phần **Secure Backup and Restoration Planning** (Lập kế hoạch Sao lưu và Phục hồi An toàn) trong Chương 7 nhấn mạnh tầm quan trọng của việc bảo vệ khả năng phục hồi của hệ thống. Dưới đây là nội dung chi tiết dựa trên tài liệu:

1. Vai trò thiết yếu của Sao lưu

Sao lưu là một phần không thể thiếu của phần mềm hiện đại. Để một kế hoạch sao lưu thực sự có chức năng và hiệu quả, nó không chỉ đơn thuần là sao chép dữ liệu mà còn phải bao gồm:

- **Mã nguồn (Code base):** Duy trì bản sao của mã thực thi.
- **Các thiết lập cấu hình (Configuration settings):** Đảm bảo các tham số vận hành được lưu trữ an toàn.
- **Dữ liệu (Data):** Những dữ liệu mà hệ thống thao tác và xử lý.

2. Quản lý và Phối hợp

- **Hệ thống hỗ trợ:** Trong một số trường hợp, việc lập kế hoạch sao lưu và phục hồi dữ liệu có thể được đảm nhiệm bởi một hệ thống khác, ví dụ như hệ quản trị cơ sở dữ liệu.
- **Bảo vệ thiết lập cấu hình:** Việc bảo mật các thiết lập cấu hình khi triển khai phần mềm là một vấn đề nghiêm trọng. Quy trình lập kế hoạch cần xác định rõ cách thức các thiết lập này được bảo vệ, sao lưu và phục hồi khi cần thiết.

3. Quy trình Thiết kế Hệ thống An toàn

- **Sự hiểu biết và Giao tiếp:** Thiết kế hệ thống an toàn đòi hỏi sự hiểu biết thấu đáo thông qua việc lập kế hoạch và giao tiếp hiệu quả trong toàn bộ đội ngũ thiết kế.
- **Các yếu tố cần thiết:** Đội ngũ cần nắm rõ mọi thành phần liên quan đến việc sao lưu và phục hồi cho cả phần mềm lẫn dữ liệu.

Tài liệu cũng lưu ý rằng các thông tin chi tiết hơn về các cơ chế này sẽ được trình bày cụ thể trong Chương 17.

Secure Data Retention, Retrieval, and Destruction

Phần **Secure Data Retention, Retrieval, and Destruction** (Lưu trữ, Truy xuất và Hủy dữ liệu An toàn) trong Chương 7 tập trung vào việc quản lý toàn bộ vòng đời của dữ liệu sau khi nó được lưu trữ trong hệ thống. Dưới đây là nội dung chi tiết:

1. Các câu hỏi then chốt khi lưu trữ dữ liệu

Dữ liệu dự định lưu trữ lâu dài (persistent) trong hệ thống cần phải được xác định rõ ràng qua một loạt các yếu tố:

- **Chủ sở hữu dữ liệu (Data owner) là ai?**
- **Mục đích của việc lưu trữ dữ liệu này là gì?**
- **Mức độ bảo vệ nào là cần thiết?**
- **Dữ liệu cần được lưu trữ trong bao lâu?**

2. Trách nhiệm quản lý

Dữ liệu được lưu trữ trong doanh nghiệp phải tuân thủ đầy đủ các trách nhiệm của **chủ sở hữu dữ liệu (data owner)** và **người giám hộ dữ liệu (data custodian)**:

- **Thiết kế kế hoạch bảo vệ:** Kế hoạch này không chỉ dành cho kho lưu trữ chính mà còn phải bao gồm các hình thức lưu trữ thay thế như bản sao lưu (backups), bản sao tại các địa điểm phục hồi sau thảm họa (DR sites) và kho lưu trữ phục vụ mục đích pháp lý (legal hold archives).
- **Bảo mật nhật ký hệ thống (System logs):** Nhật ký hệ thống có thể chứa thông tin nhạy cảm, do đó cần được bảo vệ và xây dựng kế hoạch lưu trữ phù hợp để hỗ trợ

việc thực thi pháp luật (legal hold), tìm chứng cứ điện tử (e-discovery) và tuân thủ quy định.

3. Mục đích của việc hủy dữ liệu (Data Destruction)

Việc hủy dữ liệu an toàn phục vụ hai mục đích chính trong doanh nghiệp:

- **Tiết kiệm tài nguyên:** Ngừng tiêu tốn tài nguyên vào việc duy trì các dữ liệu không còn giá trị kinh doanh.
- **Giới hạn trách nhiệm pháp lý:** Giảm thiểu rủi ro bị liên đới về mặt pháp lý liên quan đến các dữ liệu cũ.

4. Quy trình kết thúc vòng đời dữ liệu

- **Xác định thời hạn lưu trữ:** Thời hạn này được quyết định bởi hai yếu tố là **mục đích kinh doanh** và **yêu cầu tuân thủ**.
- **Thực hiện hủy dữ liệu:** Khi dữ liệu hết hạn sử dụng, **người giám hộ dữ liệu (data custodian)** có nhiệm vụ đảm bảo dữ liệu được xử lý thích hợp trên tất cả các nguồn (bản sao lưu, kho lịch sử, v.v.).
- **Trường hợp ngoại lệ:** Dữ liệu đang trong tình trạng "giữ lại để phục vụ pháp lý" (legal hold) phải được quản lý riêng và không thuộc phạm vi của các quy trình hủy dữ liệu thông thường.

Việc lập kế hoạch vòng đời dữ liệu một cách an toàn là trách nhiệm quan trọng của các chuyên gia CSSLP trong suốt quá trình phát triển phần mềm.

Perform Security Architecture and Design Review

Phần **Perform Security Architecture and Design Review** (Thực hiện Đánh giá Kiến trúc và Thiết kế Bảo mật) trong Chương 7 nhấn mạnh rằng các hệ thống an toàn không tự nhiên mà có; chúng phải được kiến trúc và kỹ thuật hóa một cách chủ đích. Dưới đây là nội dung chi tiết:

1. Mục tiêu cốt lõi

- **Bảo vệ mọi trạng thái:** Mục tiêu của việc đánh giá là nhằm đảm bảo an ninh cho cả **trạng thái hiện tại** và các **trạng thái phát triển trong tương lai**.

- **Nỗ lực ban đầu:** Điều này đòi hỏi các nỗ lực thiết kế và thực hiện các đợt đánh giá ngay từ giai đoạn đầu của dự án.

2. Các nguồn hướng dẫn và Phương pháp luận

Tùy thuộc vào phạm vi của hệ thống và các thành phần cấu thành, có rất nhiều phương pháp luận và nguồn hướng dẫn bảo mật có sẵn để đội ngũ thiết kế tham khảo:

- **NIST:** Các hướng dẫn được phát triển và công bố bởi Viện Tiêu chuẩn và Công nghệ Quốc gia.
- **20 Common Security Controls:** Bộ 20 kiểm soát bảo mật phổ biến.
- **OWASP:** Các hướng dẫn của dự án bảo mật ứng dụng web mở để bảo vệ các ứng dụng web.
- **Nguồn dữ liệu chuyên biệt:** Đối với các hệ thống xử lý dữ liệu đặc thù như tài chính, y tế hoặc dữ liệu chính phủ, cần tham khảo thêm các hướng dẫn bổ sung dành riêng cho các lĩnh vực này.

3. Quy trình thực hiện

- **Lựa chọn nguồn phù hợp:** Việc sử dụng đúng các nguồn hướng dẫn phù hợp với đặc thù dự án là rất quan trọng.
- **Thiết lập kiến trúc:** Một kiến trúc bảo mật cần được thiết lập dựa trên các hướng dẫn và yêu cầu đã xác định.
- **Đánh giá định kỳ:** Hệ thống phải được **đánh giá thường xuyên** để đảm bảo rằng các quy chuẩn của kiến trúc bảo mật đang được tuân thủ nghiêm ngặt xuyên suốt quá trình phát triển.

Việc thực hiện đánh giá này là một bước thiết yếu trong quy trình thiết kế bảo mật, giúp xác định các lỗi thiết kế trước khi chúng trở nên quá tốn kém để sửa chữa.

Define Secure Operational Architecture

Phần **Define Secure Operational Architecture** (Xác định Kiến trúc Vận hành An toàn) trong Chương 7 tập trung vào việc xây dựng một cách tiếp cận có hệ thống để quản lý rủi ro khi phần mềm được triển khai thực tế. Dưới đây là nội dung chi tiết:

1. Mục tiêu và Phạm vi

- **Mục tiêu cốt lõi:** Mục tiêu duy nhất của việc xác định kiến trúc này là tạo ra một cách tiếp cận mang tính hệ thống đối với phần mềm nhằm mục đích **quản lý các rủi ro**.
- **Thành phần cấu thành:** Kiến trúc bảo mật không chỉ có công nghệ mà còn bao gồm các **hệ thống, quy trình, con người và công cụ** được sử dụng để nhận diện, ngăn chặn và giảm thiểu các điều kiện dẫn đến rủi ro.
- **Căn cứ thiết kế:** Toàn bộ tiền đề này được thiết kế và xây dựng dựa trên việc triển khai vận hành thực tế của hệ thống, có tính đến:
    - **Sơ đồ hệ thống (System topologies)**.
    - **Các phương thức truyền thông**.
    - **Các giao diện** cung cấp thông tin về các tình trạng bình thường hoặc bất thường.

2. Các Khung kiến trúc Bảo mật Doanh nghiệp phổ biến

Tài liệu liệt kê ba khung kiến trúc thường được sử dụng nhất, chúng có thể hỗ trợ lẫn nhau vì mỗi khung có các mức độ tập trung khác nhau:

- **SABSA:** Là một loạt các khung, mô hình, phương pháp và quy trình tích hợp, được sử dụng độc lập hoặc như một giải pháp doanh nghiệp tích hợp toàn diện.
- **Open Group Library:** Cung cấp một phạm vi rộng lớn các tiêu chuẩn, hướng dẫn và thông tin kiến trúc để thiết lập một kiến trúc doanh nghiệp an toàn.
- **Open Security Architecture (OSA):** Một bộ công cụ, kiểm soát và các mẫu (patterns) nhằm hỗ trợ việc tạo ra một môi trường kiến trúc an toàn.

3. Xu hướng DevSecOps

- Tài liệu giới thiệu **DevSecOps** như một dạng môi trường vận hành hiện đại, là phiên bản bảo mật của các phương pháp Agile DevOps.
- Một nguồn tham khảo quan trọng được đề xuất để tìm hiểu cách triển khai phần mềm ở "tốc độ vận hành" trong khi vẫn duy trì an ninh doanh nghiệp là tài liệu *US DoD Enterprise DevSecOps Reference Design v1.0*.
- Việc sử dụng DevOps hoặc DevSecOps **không loại trừ** việc sử dụng các tài liệu hướng dẫn kiến trúc đã nêu ở trên.

Tóm lại, việc xác định kiến trúc vận hành an toàn giúp đảm bảo rằng các mục tiêu bảo mật từ giai đoạn thiết kế được duy trì xuyên suốt quá trình phần mềm hoạt động trong môi trường thực tế.

Use Secure Architecture and Design Principles, Patterns, and Tools

Phần **Use Secure Architecture and Design Principles, Patterns, and Tools** (Sử dụng các Nguyên tắc, Mẫu và Công cụ Thiết kế và Kiến trúc An toàn) trong Chương 7 nhấn mạnh việc biến các lý thuyết kiến trúc thành thực tế vận hành. Dưới đây là nội dung chi tiết:

- **Tính thực tiễn của kiến trúc bảo mật:** Việc sử dụng kiến trúc bảo mật không phải là một bài tập mang tính học thuật mà là một phần thường xuyên của các hoạt động triển khai và vận hành giải pháp phần mềm.
- **Từ thiết kế đến thực thi:** Thiết kế kiến trúc bảo mật chỉ là bước đầu tiên; thiết kế này phải được hiện thực hóa bằng các **chính sách và quy trình** sử dụng các công cụ và phương pháp để biến kiến trúc đó thành một "thực thể sống" trong doanh nghiệp.
- **Tích hợp vào quy trình phát triển:** Quá trình thiết kế kiến trúc phải cân nhắc đến các chính sách, mẫu (patterns), công cụ và quy trình cần thiết, đồng thời **tích hợp chúng trực tiếp vào quy trình phát triển**.
- **Hướng dẫn từ OWASP:** OWASP đã xuất bản một hướng dẫn phát triển để hỗ trợ các nhà phát triển trong việc hiện thực hóa các nguyên tắc bảo mật. Mặc dù các

Nguyên tắc Thiết kế Bảo mật của OWASP ban đầu được tạo ra cho các ứng dụng web, chúng vẫn có thể **áp dụng cho tất cả các dự án phần mềm khác**.

- **Các nguyên tắc thiết kế chính:** Các nguyên tắc này bao gồm việc xác định rõ tài sản, hiểu rõ kẻ tấn công, chú trọng vào bộ ba CIA (Tính bảo mật, Tính toàn vẹn, Tính sẵn sàng) và các nguyên tắc bảo mật khác.
- **Hệ thống hỗ trợ:** Việc triển khai một kiến trúc an toàn đòi hỏi phải có sự hỗ trợ từ cả các **thành phần vận hành và các thành phần quy trình**.

Part IV Secure Software Implementation

Chapter 8 Secure Coding Practices

Declarative vs. Imperative Security

Nội dung phần giới thiệu của mục **Declarative vs. Imperative Security** (Bảo mật khai báo so với Bảo mật mệnh lệnh) trong Chương 8 tập trung vào hai cách thức khác nhau để hiện thực hóa bảo mật trong mã nguồn. Dưới đây là chi tiết:

Bảo mật có thể được khởi tạo theo hai cách khác nhau trong mã nguồn: trong chính **vỏ bọc (container)** hoặc trong **nội dung** của vỏ bọc đó.

1. Declarative Security (Bảo mật khai báo)

- **Định nghĩa:** Lập trình khai báo là khi việc lập trình xác định "cái gì" cần thực hiện, chứ không phải "làm thế nào" để thực hiện các nhiệm vụ đó.
- **Ví dụ:** SQL là một ví dụ điển hình, nơi "cái gì" được mô tả và công cụ SQL sẽ quản lý cách thức thực hiện ("how").
- **Đặc điểm:** Bảo mật khai báo đề cập đến việc xác định các quan hệ bảo mật đối với container. Việc sử dụng cách tiếp cận dựa trên container để khởi tạo bảo mật tạo ra một giải pháp **linh hoạt hơn**, với các quy tắc bảo mật được cấu hình như một phần của quá trình **triển khai (deployment)** chứ không phải nằm trong chính mã nguồn.
- **Quản lý:** Bảo mật được quản lý bởi **nhân viên vận hành**, không phải đội ngũ phát triển.

2. Imperative Security (Bảo mật mệnh lệnh)

- **Định nghĩa:** Còn được gọi là **programmatic security** (bảo mật lập trình), đây là trường hợp ngược lại, nơi việc thực thi bảo mật được **nhúng trực tiếp vào chính mã nguồn**.
- **Đặc điểm:** Cách tiếp cận này cho phép đạt được **độ chi tiết (granularity) cao hơn** nhiều trong phương pháp bảo mật. Kiểu bảo mật chi tiết này, dưới sự kiểm soát của chương trình, có thể được sử dụng để thực thi các quy tắc nghiệp vụ phức tạp mà cách tiếp cận container (vốn mang tính chất "được tất cả hoặc không có gì") không thể thực hiện được.
- **Hạn chế:** Mặc dù là một lợi thế cho các điều kiện cụ thể, nhưng nó có xu hướng làm cho mã nguồn **ít có khả năng di động (portable) hoặc tái sử dụng (reusable)** hơn do logic nghiệp vụ cụ thể đã được xây dựng cứng vào chương trình.

3. Tầm quan trọng của việc lựa chọn

Việc lựa chọn giữa chức năng bảo mật khai báo, bảo mật mệnh lệnh, hoặc thậm chí là sự kết hợp của cả hai, là một **quyết định ở mức thiết kế (design-level decision)**. Một khi hệ thống được thiết kế với một phương pháp cụ thể, quy trình Vòng đời Phát triển Phần mềm An toàn (SDL) có thể xây dựng các biện pháp bảo vệ phù hợp dựa trên thiết kế đó. Đây là một trong những yếu tố yêu cầu phải có quyết định thiết kế sớm, vì nhiều yếu tố khác sẽ phụ thuộc vào nó.

Bootstrapping

Phần **Bootstrapping** trong Chương 8 (và được bổ sung thêm ở Chương 16) tập trung vào việc bảo mật quá trình khởi động và thiết lập ban đầu của một chương trình hoặc hệ thống. Dưới đây là nội dung chi tiết:

1. Định nghĩa về Bootstrapping

- **Quá trình khởi động tự duy trì:** Bootstrapping đề cập đến quá trình khởi động tự duy trì diễn ra khi máy tính bắt đầu hoạt động hoặc khi một chương trình được khởi tạo,.

- **Nguồn gốc thuật ngữ:** Thuật ngữ này bắt nguồn từ một ẩn dụ ở thế kỷ 19 về việc "tự nhấc mình lên bằng dây giày" (pulling oneself up by one's bootstraps), ám chỉ các hành động diễn ra mà không cần sự trợ giúp từ bên ngoài.
- **Tên gọi khác:** Trong ngữ cảnh máy tính cá nhân (PC), quá trình này còn được gọi đơn giản là **booting**.

2. Các giai đoạn của quá trình Bootstrapping

Khi một hệ thống máy tính khởi động, một tập hợp các hoạt động được sắp xếp sẵn sẽ bắt đầu, bao gồm:

- **POST (Power-on self-test):** Các quy trình tự kiểm tra khi bật nguồn,.
- **Boot loaders:** Các bộ nạp khởi động.
- **Khởi tạo hệ điều hành:** Các hoạt động thiết lập ban đầu của hệ điều hành.
- **IPL (Initial Program Load):** Nạp chương trình ban đầu sau bước POST.

3. Thách thức và Rủi ro Bảo mật

- **Sự can thiệp của mã độc:** Việc bảo mật trình tự khởi động là một thách thức vì phần mềm độc hại thường tìm cách ngắt quãng quá trình bootstrapping để chèn các "móc" (hooks) của riêng chúng vào hệ điều hành,.
- **Mất lòng tin hệ thống:** Tính toàn vẹn của quá trình khởi động là bắt buộc; nếu các bước này bị xâm phạm, tất cả các quy trình diễn ra sau đó trên máy tính đều không thể tin cậy được.

4. Các nguyên tắc lập trình và triển khai an toàn

Để đảm bảo an toàn cho quá trình bootstrapping, các nhà phát triển và chuyên gia bảo mật cần lưu ý:

- **Kiểm soát biến môi trường:** Khi viết mã ứng dụng dựa trên các yếu tố hệ thống (như biến môi trường PATH), cần hết sức cẩn thận để đảm bảo các giá trị này không bị thay đổi ngoài tầm kiểm soát của ứng dụng,.

- **Sử dụng tệp cấu hình:** Nên sử dụng các tệp cấu hình để quản lý các yếu tố khởi động và giữ chúng dưới sự kiểm soát trực tiếp của ứng dụng,.
- **Thiết lập tham số chính xác:** Quá trình bootstrapping phải đảm bảo tính chính xác của cấu hình ban đầu, bao gồm việc đặt các giá trị mặc định, tham số thực thi phù hợp và các tính năng bảo mật (ví dụ: cài đặt trình giám sát tham chiếu - reference monitor),.
- **Kiểm tra định kỳ:** Việc triển khai các quy trình kiểm tra và giám sát nội bộ để thu thập dữ liệu về hiệu suất và tính toàn vẹn của quá trình khởi động là rất quan trọng.

Cryptographic Agility

Phần **Cryptographic Agility** (Tính linh hoạt mật mã) trong Chương 8 và Chương 10 tập trung vào khả năng thích ứng của phần mềm trước sự lỗi thời của các thuật toán mật mã. Dưới đây là nội dung chi tiết:

1. Định nghĩa và Mục tiêu

- **Định nghĩa:** Cryptographic agility là khả năng quản lý các đặc điểm cụ thể của các hàm mật mã được thể hiện trong mã nguồn mà **không cần phải biên dịch lại (recompiling)**, thường là thông qua một **tệp cấu hình (configuration file)**.
- **Mục tiêu:** Tạo ra phần mềm có thể thay đổi cấu hình thuật toán ngay lập tức ("on the fly") để thay thế các thuật toán đã bị phát hiện là có điểm yếu hoặc bị lỗi thời.

2. Tại sao cần tính linh hoạt mật mã?

- **Sự lỗi thời của thuật toán:** Lịch sử cho thấy các thuật toán chúng ta tin dùng hôm nay (như DES, MD5, RC2) đều sẽ bị lỗi thời hoặc bị phá vỡ trong tương lai.
- **Thay thế hiệu quả:** Khi một thuật toán bị coi là không an toàn, cần có một cơ chế để thay thế nó bằng thuật toán mạnh hơn một cách hiệu quả mà không cần can thiệp vào mã nguồn.

3. Phương thức triển khai

- **Trừu tượng hóa lời gọi thư viện:** Các nhà phát triển sử dụng các lời gọi thư viện mật mã, sau đó trừu tượng hóa chúng sao cho việc chỉ định thuật toán cụ thể được quản lý thông qua tệp cấu hình.
- **Cập nhật cấu hình:** Điều này cho phép thay đổi thuật toán mật mã chỉ bằng một thay đổi nhỏ trong tệp cấu hình và khởi động lại chương trình.

4. Lợi ích trong phân phối quốc tế

- **Tuân thủ quy định:** Một số quốc gia có các quy định hạn chế việc xuất khẩu hoặc sử dụng các thuật toán mật mã nhất định.
- **Giải pháp duy nhất:** Thay vì tạo ra nhiều phiên bản mã nguồn khác nhau cho từng quốc gia, tính linh hoạt mật mã cho phép quản lý các biến thể này thông qua cấu hình.

5. Tầm quan trọng trong Thiết kế

- **Quyết định sớm:** Cryptographic agility là một **quyết định ở mức thiết kế (design-level decision)**.
- **Sự phụ thuộc của SDL:** Một khi quyết định này được đưa ra, quy trình Vòng đời Phát triển Phần mềm An toàn (SDL) sẽ xây dựng các biện pháp bảo vệ phù hợp dựa trên thiết kế đó, vì nhiều thành phần khác sẽ phụ thuộc vào nó.

Handling Configuration Parameters

Phần **Handling Configuration Parameters** (Xử lý các tham số cấu hình) trong Chương 8 tập trung vào việc bảo vệ các thiết lập có khả năng thay đổi hành vi của ứng dụng. Dưới đây là nội dung chi tiết:

1. Tầm quan trọng của việc bảo mật cấu hình

- **Thay đổi hành vi ứng dụng:** Các tham số cấu hình có thể làm thay đổi cách thức hoạt động của chương trình.
- **Rủi ro từ cấu hình:** Việc quản lý bảo mật cho các tham số này là tối quan trọng, đặc biệt khi chúng có thể thay đổi các hành vi lập trình.

2. Phân tích mức độ quan trọng (Criticality)

- Để xác định mức độ quan trọng, cần phân tích xem **chức năng ứng dụng nào sẽ bị thay đổi** bởi tham số đó.
- **Mức độ rủi ro:** Có thể dao động từ **gần như bằng không** (đối với các tham số không quan trọng) đến **cực kỳ cao** nếu các chức năng quan trọng như hàm mật mã có thể bị thay đổi hoặc vô hiệu hóa thông qua cấu hình.

3. Các biện pháp bảo vệ

Mức độ bảo vệ phải **tương xứng với rủi ro tiếp xúc**. Các phương pháp bao gồm:

- **Mức độ đơn giản:** Lưu trữ tệp cấu hình trong một thư mục được bảo vệ bởi **Danh sách kiểm soát truy cập (ACL)**.
- **Mức độ cao nhất:** Thực hiện **mã hóa dữ liệu nhạy cảm** ngay bên trong tệp cấu hình.

4. Truyền dữ liệu cấu hình

Dữ liệu cấu hình có thể được truyền từ một ứng dụng gọi (calling application) đến ứng dụng mục tiêu theo nhiều cách:

- Thông qua **chuỗi URL**.
- Thông qua **tiêm trực tiếp vào bộ nhớ (direct memory injection)** dựa trên thông tin ứng dụng mục tiêu cung cấp.

5. Yêu cầu kiểm thử (Testing)

Quá trình kiểm thử cần kiểm tra kỹ việc xử lý dữ liệu cấu hình thông qua các yếu tố như:

- Các trình định vị tài nguyên thống nhất (**URLs**).
- **Cookies**.
- Các **tệp tạm thời** (temp files) và các thiết lập khác để xác nhận dữ liệu được xử lý chính xác và an toàn.

Memory Management

Phần **Memory Management** (Quản lý Bộ nhớ) trong Chương 8 tập trung vào cách thức phần mềm kiểm soát việc phân bổ và sử dụng bộ nhớ để đảm bảo tính an toàn và ổn định. Dưới đây là nội dung chi tiết:

1. Tầm quan trọng của Quản lý Bộ nhớ

- **Chức năng:** Bộ nhớ là nơi lưu trữ mã vận hành, dữ liệu, biến và không gian làm việc của chương trình.
- **Độ phức tạp:** Việc quản lý rất phức tạp do tính chất động của việc sử dụng bộ nhớ giữa ứng dụng đơn lẻ, nhiều chương trình chạy song song và hệ điều hành.
- **Trách nhiệm:** Việc phân bổ bộ nhớ là trách nhiệm chung của cả hệ điều hành và ứng dụng.

2. Managed Code (Mã được quản lý) và Unmanaged Code (Mã không được quản lý)

- **Managed Code:** Một trong những điểm mạnh chính là khả năng tự động quản lý bộ nhớ thông qua các công cụ thực thi mã trung gian (như CLR hoặc JVM). Nó cung cấp khả năng **kiểm soát vòng đời tài nguyên tự động** và chạy trong môi trường sandbox, giúp runtime engine duy trì quyền kiểm soát đối với mọi tài nguyên.
- **Unmanaged Code:** Nhà phát triển phải chịu trách nhiệm hoàn toàn về tài nguyên và bộ nhớ, bao gồm thu gom rác (garbage collection), gộp luồng (thread pooling) và ngăn chặn tràn bộ nhớ. Do đó, unmanaged code dễ gặp các lỗi rò rỉ bộ nhớ như **tràn bộ đệm (buffer overruns)** và **ghi đè con trỏ (pointer overrides)**.

3. Thực hành An toàn Kiểu (Type-Safe Practice)

- **Định nghĩa:** Type safety là mức độ mà một ngôn ngữ lập trình ngăn chặn các lỗi phát sinh do sự khác biệt giữa các kiểu dữ liệu trong chương trình.
- **Mối liên hệ với bộ nhớ:** An toàn kiểu liên kết chặt chẽ với an toàn bộ nhớ. Mã an toàn kiểu sẽ **không vô tình truy cập vào các vị trí bộ nhớ tùy ý** nằm ngoài phạm vi dự kiến.

- **Lợi ích:** Việc xác định rõ các biến và độ dài bộ nhớ tương ứng giúp lập trình an toàn kiểu tự động giải quyết nhiều vấn đề liên quan đến bộ nhớ.

4. Nguyên lý Cục bộ (Locality)

- **Định nghĩa:** Locality là nguyên lý cho rằng khi một chương trình tham chiếu đến một vùng bộ nhớ, các lần truy cập bộ nhớ tiếp theo thường có thể dự đoán được và nằm gần các tham chiếu trước đó.
- **Rủi ro:** Các cuộc tấn công như tràn bộ đệm thường lợi dụng nguyên lý này để chèn mã độc.
- **Biện pháp phòng vệ: Address Space Layout Randomization (ASLR)** là kỹ thuật quản lý bộ nhớ cụ thể được thiết kế để chống lại các cuộc tấn công dựa trên tính cục bộ bằng cách xáo trộn các vị trí trong không gian địa chỉ bộ nhớ.

Error Handling

Phần **Error Handling** (Xử lý lỗi) trong Chương 8, còn được gọi là **Exception Management** (Quản lý ngoại lệ), tập trung vào việc đảm bảo hệ thống phản ứng một cách an toàn và có thể dự đoán được khi gặp phải các tình huống bất thường. Dưới đây là nội dung chi tiết:

1. Mục tiêu và Định nghĩa

- **Định nghĩa:** Error handling là cách một ứng dụng phát hiện và xử lý các thất bại, có thể do người dùng gây ra, các hậu quả không mong muốn hoặc lỗi lập trình.
- **Mục tiêu cốt lõi:** Ngăn chặn hệ thống rơi vào **trạng thái không an toàn** (insecure state) khi có lỗi xảy ra.

2. Các tiêu chí cho việc Quản lý Ngoại lệ An toàn

Để quản lý ngoại lệ một cách an toàn, hệ thống cần đáp ứng các tiêu chí sau:

- **Phát hiện và Xử lý:** Tất cả các ngoại lệ phải được phát hiện và xử lý thay vì để hệ điều hành (OS) tự xử lý. OS thường thiếu kiến thức nghiệp vụ để xử lý lỗi một cách an toàn, dễ dẫn đến các vấn đề như leo thang đặc quyền.

- **Thất bại an toàn (Fail-safe):** Hệ thống phải được thiết kế để không vô tình cấp quyền truy cập hoặc để lộ dữ liệu khi gặp lỗi.
- **Không rò rỉ thông tin:** Các thông báo lỗi không được tiết lộ thông tin nhạy cảm như chuỗi kết nối cơ sở dữ liệu, thông tin đăng nhập, đường dẫn tệp hoặc mật mã.

3. Thực hành lập trình an toàn

- **Xử lý gần nguồn lỗi:** Việc bắt lỗi (catching errors) ngay tại nơi phát sinh giúp việc ghi nhật ký (logging) và quản lý trở nên chi tiết và ít mơ hồ hơn về nguồn gốc hoặc nguyên nhân gây lỗi.
- **Ngăn chặn lỗi lan truyền (Cascading):** Mỗi hàm nên thực hiện giảm thiểu lỗi hoàn toàn (bao gồm bẫy lỗi và xử lý triệt để) trước khi trả kết quả về quy trình gọi (calling routine) để tránh lỗi lan rộng ra toàn hệ thống.
- **Xử lý các ngoại lệ phổ biến:** Các lỗi như **tràn số học (arithmetic overflows)** nên được bẫy và xử lý bằng logic nghiệp vụ bên trong phần mềm thay vì để OS can thiệp.

4. Ghi nhật ký lỗi

- Việc ghi lại chi tiết các ngoại lệ là rất quan trọng cho việc gỡ lỗi (debugging) và phân tích sau sự cố.
- Tuy nhiên, các mục nhật ký lỗi phải được bảo vệ và không được chứa dữ liệu định danh cá nhân (PII) hoặc các bí mật hệ thống.

Interface Coding

Phần **Interface Coding** (Lập trình Giao diện) trong Chương 8 tập trung vào việc bảo mật các điểm kết nối giữa các thành phần phần mềm. Dưới đây là nội dung chi tiết:

1. Vai trò của Giao diện lập trình ứng dụng (API)

- **Định nghĩa:** API xác định cách thức các thành phần phần mềm được kết nối và tương tác với nhau.
- **Tính mô-đun:** Phát triển phần mềm hiện đại được thực hiện theo phương thức mô-đun, sử dụng các API để kết nối các chức năng của các mô-đun khác nhau.

2. Giao diện là Điểm yếu Bảo mật (Entry Points)

- **Điểm xâm nhập:** API có ý nghĩa quan trọng vì chúng đại diện cho các điểm xâm nhập (entry points) vào phần mềm.
- **Phân tích rủi ro:** Phân tích bề mặt tấn công (attack surface analysis) và mô hình hóa mối đe dọa (threat modeling) phải xác định được các API có thể bị tấn công.
- **Kế hoạch giảm thiểu:** Đội ngũ phát triển cần có các kế hoạch giảm thiểu rủi ro để giới hạn các nguy cơ liên quan đến những điểm xâm nhập này.

3. Quản lý API bên thứ ba và API lỗi thời

- **Kiểm tra API bên thứ ba:** Các API của bên thứ ba được đưa vào ứng dụng phải được kiểm tra kỹ lưỡng, và các sai sót hoặc vấn đề bảo mật phải được giảm thiểu như một phần của quy trình Vòng đời Phát triển Phần mềm An toàn (SDL).
- **Loại bỏ API yếu:** Các API cũ, yếu và không còn được hỗ trợ (deprecated) cần phải được xác định rõ và không được phép đưa vào ứng dụng cuối cùng.

4. Các yêu cầu bảo mật bắt buộc cho đầu vào giao diện

- **Xác thực:** Điều quan trọng là phải có mức độ xác thực (authentication) phù hợp đối với tất cả các đầu vào giao diện vào ứng dụng của bạn.
- **Kiểm toán (Auditing):** Cần phải thực hiện kiểm toán các tương tác bên ngoài đối với bất kỳ hoạt động đặc quyền (privileged operations) nào được thực hiện thông qua một giao diện.

Primary Mitigations

Phần **Primary Mitigations** (Các biện pháp giảm thiểu rủi ro chính) trong Chương 8 trình bày một bộ quy tắc được coi là các thực hành tốt nhất (best practices) đã được kiểm chứng qua thời gian. Dưới đây là nội dung chi tiết:

1. Danh sách các biện pháp giảm thiểu cốt lõi

Đây là những công cụ tiêu chuẩn mà một chuyên gia CSSLP cần nắm vững và áp dụng thông qua các báo cáo mối đe dọa (threat reports):

- **Thắt chặt môi trường vận hành (Lock down your environment):** Đảm bảo môi trường chạy mã nguồn được bảo vệ tối đa.
- **Kiểm soát hoàn toàn đầu vào (Establish and maintain control over all inputs):** Luôn xác thực và kiểm soát mọi dữ liệu đi vào hệ thống.
- **Kiểm soát hoàn toàn đầu ra (Establish and maintain control over all outputs):** Đảm bảo dữ liệu đầu ra không bị rò rỉ hoặc sai lệch.
- **Nguyên tắc "Giả định bị xâm nhập":** Luôn giả định rằng các thành phần bên ngoài có thể bị phá hoại và mã nguồn của bạn có thể bị bất kỳ ai đọc được.
- **Sử dụng thư viện và khung công nghệ (frameworks) an toàn:** Ưu tiên các công cụ giúp lập trình viên tránh tự gây ra các lỗ hổng bảo mật.
- **Sử dụng các tính năng bảo mật tiêu chuẩn ngành:** Tuyệt đối không tự phát minh ra các thuật toán hoặc tính năng bảo mật riêng.
- **Tích hợp bảo mật vào toàn bộ vòng đời phát triển (SDL):** Bảo mật không phải là bước cuối cùng mà là một phần xuyên suốt.
- **Sử dụng đa dạng các phương pháp:** Kết hợp nhiều kỹ thuật khác nhau để tìm kiếm và ngăn chặn điểm yếu một cách toàn diện.

2. Bản chất của Lập trình phòng thủ (Defensive Coding)

- Lập trình phòng thủ không phải là một "nghệ thuật huyền bí" mà là việc áp dụng một cách có hệ thống các tài liệu từ báo cáo mối đe dọa.
- **Các yếu tố nền tảng:** Giảm bề mặt tấn công (attack surface reduction), hiểu rõ các lỗ hổng phổ biến và áp dụng các biện pháp giảm thiểu tiêu chuẩn.
- **Bộ công cụ bổ trợ:** Bao gồm phân tích mã (code analysis), đánh giá mã (code review), quản lý phiên bản, tính linh hoạt mật mã, quản lý bộ nhớ, xử lý ngoại lệ và sử dụng mã được quản lý (managed code).

Learning from Past Mistakes

Phần **Learning from Past Mistakes** (Học hỏi từ những sai lầm trong quá khứ) trong Chương 8 nhấn mạnh tầm quan trọng của việc rút kinh nghiệm từ thực tế để cải thiện an ninh phần mềm. Dưới đây là nội dung chi tiết:

- **Nguồn thông tin thực tế:** Một trong những nguồn thông tin tốt nhất về các thất bại bảo mật đến từ các lỗi thực thi trong thế giới thực của các doanh nghiệp trong ngành.
- **Lời cảnh báo cho các đội ngũ:** Khi một công ty phải khắc phục một sự cố bảo mật (chẳng hạn như một "cửa sau" - back door bị bỏ lại trong sản phẩm), đây được coi là một lời cảnh tỉnh cho tất cả các đội ngũ phát triển ở các công ty khác.
- **Hậu quả của việc lặp lại sai lầm:** Mặc dù sai sót là điều có thể xảy ra, nhưng việc lặp lại các vấn đề đã được biết đến là điều rất khó giải trình với khách hàng và ban quản lý. Điều này đặc biệt đúng khi các lỗi này có tác động lớn và chi phí khắc phục rất tốn kém cho cả công ty phần mềm lẫn khách hàng.
- **Thực hành kinh doanh tốt:** Học hỏi từ thất bại của người khác và đưa chúng vào danh sách "những sai lầm cần tránh" của chính mình là một thực hành kinh doanh tốt.
- **Cập nhật yêu cầu bảo mật:** Một phần vai trò của đội ngũ bảo mật là duy trì và cập nhật danh sách các yêu cầu bảo mật cho các dự án.
- **Lợi ích về chi phí và thời gian:** Việc kiểm tra lỗi từ các công ty khác và cập nhật bộ yêu cầu bảo mật để ngăn chặn các cạm bẫy đã biết sẽ giúp tiết kiệm thời gian và tiền bạc về lâu dài,.

Secure Design Principles

Phần **Secure Design Principles** (Các nguyên tắc Thiết kế An toàn) trong Chương 8 tập trung vào việc áp dụng các nguyên tắc thiết kế kinh điển vào quá trình lập trình thực tế để đạt được mục tiêu bảo mật ứng dụng.

Dưới đây là nội dung chi tiết của các nguyên tắc này:

1. Bản chất của Thiết kế An toàn

- **Không phải ngẫu nhiên:** Các thiết kế an toàn là sản phẩm của kiến trúc và kế hoạch có chủ đích, dựa trên nền tảng là các nguyên tắc thiết kế an toàn đã được kiểm chứng qua thời gian.
- **Dựa trên rủi ro:** Nhà thiết kế sử dụng thông tin từ **phân tích bề mặt tấn công** và **mô hình hóa mối đe dọa** để xác định nơi cần áp dụng các phần tử thiết kế cụ thể nhằm đạt được mục tiêu bảo mật.

2. Các Nguyên tắc Thiết kế Cốt lõi

- **Good Enough Security (Bảo mật vừa đủ):** Tránh thiết kế quá mức gây lãng phí tài nguyên hoặc bảo mật dưới mức làm tăng rủi ro. Cần xác định mức độ chức năng bảo mật phù hợp dựa trên giá trị của tài sản cần bảo vệ.
- **Least Privilege (Đặc quyền tối thiểu):** Một đối tượng (subject) chỉ nên có các quyền cần thiết để thực hiện tác vụ hiện tại và không có thêm bất kỳ quyền nào khác. Việc chạy phần mềm với quyền quản trị (admin/root) là một thực hành xấu vì nó loại bỏ các rào cản bảo mật của hệ điều hành.
- **Separation of Duties (Phân tách nhiệm vụ):** Đảm bảo một nhiệm vụ quan trọng đòi hỏi nhiều hơn một cá nhân hoặc điều kiện để hoàn thành, nhằm ngăn chặn việc lạm dụng hệ thống.
- **Defense in Depth (Phòng thủ chiều sâu):** Sử dụng nhiều lớp bảo vệ chồng lên nhau và đa dạng về tính chất (như kết hợp mã hóa và kiểm soát truy cập) để nếu một lớp bị phá vỡ, các lớp khác vẫn có thể duy trì an ninh.
- **Fail Safe (Thất bại an toàn):** Khi một hệ thống gặp lỗi hoặc thất bại, nó phải chuyển về **trạng thái an toàn** (ví dụ: mặc định từ chối truy cập - *explicit deny*) thay vì vô tình cấp quyền trái phép.
- **Economy of Mechanism (Kinh tế cơ chế):** Ưu tiên sự đơn giản và thanh lịch trong thiết kế. Hệ thống càng phức tạp thì càng khó hiểu, khó khắc phục sự cố và càng có nhiều lỗ hổng tiềm ẩn.

- **Complete Mediation (Giám sát toàn diện):** Mọi yêu cầu truy cập của đối tượng vào tài nguyên phải được kiểm tra quyền hạn mỗi khi yêu cầu đó xảy ra, đảm bảo hệ thống xác thực không bao giờ bị bỏ qua.
- **Open Design (Thiết kế mở):** Bảo mật của hệ thống không được phụ thuộc vào việc giữ bí mật thiết kế hay thuật toán (bảo mật thông qua sự mơ hồ - *security through obscurity*) mà nên phụ thuộc vào những yếu tố như khóa mật mã.
- **Least Common Mechanism (Cơ chế dùng chung ít nhất):** Hạn chế việc các tiến trình hoặc người dùng khác nhau chia sẻ chung một cơ chế truy cập để tránh tạo ra các đường dẫn thông tin vô tình giữa chúng.
- **Psychological Acceptability (Sự chấp nhận về tâm lý):** Các biện pháp bảo mật phải dễ sử dụng và không gây cản trở người dùng. Nếu quá phức tạp, người dùng sẽ tìm cách lách luật, làm vô hiệu hóa các nỗ lực bảo mật.
- **Weakest Link (Mắt xích yếu nhất):** Một hệ thống chỉ mạnh bằng mắt xích yếu nhất của nó. Kẻ tấn công sẽ luôn tìm điểm yếu nhất thay vì tấn công vào nơi có hàng phòng thủ mạnh nhất.
- **Leverage Existing Components (Tận dụng các thành phần hiện có):** Việc tái sử dụng các thành phần đã được kiểm chứng (như thư viện mật mã tiêu chuẩn) giúp giảm thiểu sai sót và thu hẹp bề mặt tấn công so với việc tự viết mã mới. Tuy nhiên, mã cũ (legacy code) vẫn cần được rà soát bảo mật nghiêm ngặt.
- **Single Point of Failure (Điểm lỗi đơn lẻ):** Thiết kế phải đảm bảo không có bất kỳ điểm nào mà nếu nó thất bại sẽ khiến toàn bộ hệ thống sụp đổ.

Interconnectivity

Phần **Interconnectivity** (Sự kết nối lẫn nhau) trong Chương 8 tập trung vào việc bảo mật các con đường truyền thông tin giữa các thành phần của ứng dụng. Khi các thành phần giao tiếp với nhau, chúng tạo ra các lộ trình cần được bảo vệ nghiêm ngặt.

Dưới đây là nội dung chi tiết của 3 hoạt động cốt lõi trong việc quản lý sự kết nối này:

1. Session Management (Quản lý Phiên)

- **Mục tiêu:** Kiểm soát kênh truyền thông trên cơ sở từng cuộc hội thoại (conversation-by-conversation) để ngăn chặn việc bị chiếm quyền điều khiển (hijacking) bởi bên thứ ba không được phép.
- **Lợi ích:** Cho phép nhiều bên sử dụng cùng một phương thức truyền thông mà không can thiệp lẫn nhau hoặc làm rò rỉ thông tin chéo giữa các bên.
- **Triển khai:** Có thể đơn giản bằng cách sử dụng giao thức **HTTPS** hoặc mã hóa riêng biệt cho từng luồng giao tiếp cá nhân. Nhà thiết kế cần cân bằng giữa việc bảo mật và độ phức tạp của hệ thống.

2. Exception Management (Quản lý Ngoại lệ)

- **Định nghĩa:** Là phản hồi mang tính lập trình đối với các ngoại lệ phát sinh trong quá trình vận hành chương trình, thông qua các hàm đặc biệt gọi là **exception handlers**.
- **Nguyên tắc:** Ứng dụng nên tự bẫy và xử lý lỗi thay vì để hệ điều hành (OS) làm việc đó, vì OS thiếu kiến thức nghiệp vụ để xử lý an toàn.
- **Rủi ro:** Các ngoại lệ không được xử lý (unhandled exceptions) thường là nơi xảy ra các lỗi **leo thang đặc quyền (privilege escalation)**.
- **Lưu ý khi ghi nhật ký:** Khi xử lý ngoại lệ, cần cẩn trọng không làm lộ thông tin nhạy cảm (như mật khẩu, đường dẫn tệp, PII) trong các mục nhật ký (log entries).

3. Configuration Management (Quản lý Cấu hình)

- **Phạm vi:** Quản lý các yếu tố cấu hình chức năng kết nối như tham số khởi tạo, chuỗi kết nối (connection strings), đường dẫn (paths) và các khóa (keys).
- **Tầm quan trọng:** Các yếu tố này điều khiển cách phần mềm hoạt động và tương tác với môi trường bên ngoài, do đó chúng phải được xác định và bảo vệ như một phần của quy trình bảo mật.

Cryptographic Failures

Phần **Cryptographic Failures** (Thất bại về Mật mã) trong Chương 8 tập trung vào những lỗi phổ biến khi triển khai các biện pháp bảo vệ bằng mật mã, dẫn đến việc dữ liệu nhạy cảm và phần mềm bị xâm phạm. Các chuyên gia CSSLP cần tránh tự xây dựng thuật toán mật mã riêng mà nên sử dụng các thư viện đã được ngành công nghiệp chấp nhận.

Dưới đây là chi tiết các loại thất bại mật mã chính:

1. Hard-Coded Credentials (Thông tin đăng nhập bị mã hóa cứng)

- **Vấn đề:** Việc ghi trực tiếp mật khẩu, khóa hoặc các dữ liệu nhạy cảm vào mã nguồn làm cho chúng rất khó thay đổi (cần phải cập nhật lại toàn bộ chương trình).
- **Rủi ro:** Các thông tin này sẽ không thể giữ bí mật vì kẻ tấn công có thể sử dụng các kỹ thuật đảo ngược mã nguồn (reverse-engineer) để tìm ra vị trí và giá trị của khóa bí mật.
- **Giải pháp:** Không bao giờ cho phép mã hóa cứng trong mã nguồn. Lỗi này có thể dễ dàng được phát hiện thông qua các buổi rà soát mã (code walk-throughs).

2. Missing Encryption of Sensitive Data (Thiếu mã hóa dữ liệu nhạy cảm)

- **Nguyên nhân:** Thường do đội ngũ phát triển thiếu hiểu biết về mức độ nhạy cảm của dữ liệu. Chủ sở hữu dữ liệu (data owner) có trách nhiệm xác định và tài liệu hóa các yêu cầu bảo vệ này.
- **Phạm vi bảo vệ:** Không chỉ dữ liệu chính, mà các bản sao lưu (backups), tệp nhật ký (log files) và cơ chế báo cáo lỗi cũng cần được mã hóa vì chúng thường là nơi để lộ bí mật hệ thống.
- **Kỹ thuật bổ trợ: Tokenization** (Thay thế dữ liệu nhạy cảm bằng một giá trị ngẫu nhiên không có liên kết bên ngoài) là một thực hành tốt để bảo vệ thông tin như số thẻ tín dụng mà không để lộ dữ liệu thực.

3. Use of a Broken or Risky Cryptographic Algorithm (Sử dụng thuật toán mật mã yếu hoặc rủi ro)

- **Sự lỗi thời:** Các thuật toán từng được coi là tiêu chuẩn như DES, MD5, RC2 hiện đã lỗi thời do sự gia tăng của năng lực tính toán.

- **Sai lầm nghiêm trọng:** Tự tạo ra thuật toán mật mã riêng ("rolling your own crypto") gần như luôn dẫn đến thất bại vì chúng sẽ bị tin tặc phá vỡ nhanh chóng.
- **Số ngẫu nhiên:** Các hàm tạo số ngẫu nhiên thông thường trong thư viện lập trình (pseudo-random) không đủ độ ngẫu nhiên cho mật mã. Phải sử dụng các hàm tạo số ngẫu nhiên mạnh về mặt mật mã (cryptographically strong) từ các thư viện chuẩn.
- **Độ dài khóa:** Cần chú trọng đến độ dài khóa (ví dụ: khóa RSA hiện nay phải lớn hơn 2.048 bits).

4. Download of Code Without Integrity Check (Tải mã nguồn mà không kiểm tra tính toàn vẹn)

- **Rủi ro:** Mã độc có thể được đính kèm vào các bản cập nhật hoặc phần mềm tải từ Internet.
- **Biện pháp:** Tất cả mã tải về phải được xác minh tính toàn vẹn bằng các giá trị băm (hash values) trước khi cài đặt. Tuy nhiên, việc chỉ gửi kèm mã băm là không đủ (vì tin tặc cũng có thể sửa mã băm); mã băm phải đến từ một nguồn đáng tin cậy hoặc thông qua chữ ký số.

5. Use of a One-Way Hash Without a Salt (Sử dụng hàm băm một chiều không có muối)

- Mặc dù tài liệu không đi sâu vào chi tiết đoạn văn cho mục này trong chương 8, nó được liệt kê là một trong những thất bại mật mã nghiêm trọng và nằm trong danh sách CWE/SANS Top 25 (CWE-759). Việc thiếu "muối" (salt) cho phép kẻ tấn công sử dụng các bảng tra cứu trước (rainbow tables) để bẻ khóa hàm băm dễ dàng hơn.

Input Validation Failures

Phần **Input Validation Failures** (Thất bại trong xác thực đầu vào) trong Chương 8 được coi là cơ chế phòng thủ quan trọng nhất mà một lập trình viên có thể triển khai. Việc coi tất cả các dữ liệu nhập vào là "thù địch" cho đến khi được xác thực đúng cách có thể giảm thiểu nhiều cuộc tấn công phổ biến.

Dưới đây là các nội dung chi tiết:

1. Bản chất của Xác thực đầu vào

- **Nguyên tắc vàng:** Luôn giả định mọi dữ liệu đầu vào từ bên ngoài (người dùng, tệp tin, mạng) là không an toàn,.
- **Thời điểm xác thực:** Việc xác thực phải diễn ra **sau khi tất cả các trình phân tích cú pháp (parsers) đã hoàn tất** việc xử lý luồng dữ liệu. Điều này đặc biệt quan trọng đối với các ứng dụng web sử dụng Unicode hoặc các bộ ký tự quốc tế để tránh việc dữ liệu bị thay đổi sau khi đã kiểm tra.
- **Các lỗ hổng có thể giảm thiểu:** Xác thực đầu vào giúp chống lại lỗi tràn bộ đệm, tấn công kịch bản chéo trang (XSS), giả mạo yêu cầu chéo trang (CSRF), duyệt thư mục (path traversal) và tính toán sai kích thước bộ đệm.

2. Buffer Overflow (Tràn bộ đệm)

- **Định nghĩa:** Xảy ra khi dữ liệu đầu vào lớn hơn không gian bộ nhớ được cấp phát, dẫn đến việc ghi đè lên các phần tử quan trọng khác của chương trình.
- **Tầm quan trọng:** CERT/CC ước tính khoảng **50% các vụ khai thác phần mềm** trong lịch sử bắt nguồn từ một dạng của tràn bộ đệm.
- **Nguyên nhân:** Sự kết hợp giữa thực hành lập trình kém và điểm yếu của ngôn ngữ (ví dụ: ngôn ngữ C cho phép truy cập bộ nhớ trực tiếp qua con trỏ và có các hàm không an toàn như `gets()`).
- **Phòng thủ:** Sử dụng các hàm an toàn hơn (ví dụ: `strncpy()` thay vì `strcpy()`) và luôn kiểm tra độ dài dữ liệu trước khi nạp vào bộ đệm,.

3. Canonical Form (Dạng chuẩn hóa)

- **Vấn đề:** Một dữ liệu đầu vào có thể có nhiều cách biểu diễn khác nhau (Unicode, mã hóa Hex, UTF-8). Ví dụ, ký tự `o` có thể được mã hóa thành `%6f`.
- **Canonicalization:** Là quá trình đưa các chuỗi về **dạng tiêu chuẩn hoặc đơn giản nhất** trước khi thực hiện các kiểm tra bảo mật.
- **Rủi ro:** Kẻ tấn công thường sử dụng mã hóa kép (double encoding) để che giấu các ký tự nguy hiểm như `../` (dùng trong tấn công duyệt thư mục) nhằm vượt qua các

bộ lọc bảo mật,. Nếu việc kiểm tra lỗi xảy ra trước khi đưa về dạng chuẩn, các cuộc tấn công sẽ bị bỏ sót.

4. Missing Defense Functions (Thiếu các hàm phòng vệ)

- **Thực thi đồng nhất:** Các cơ chế phòng thủ cơ bản như xác thực (authentication) và ủy quyền (authorization) phải được áp dụng đồng bộ cho mọi hoạt động đi qua **ranh giới tin cậy (trust boundary)**.
- **Ví dụ:** Việc có vé vào sân vận động (xác thực) không có nghĩa là bạn được ngồi ở bất kỳ chỗ nào; bạn phải xuất trình vé lần nữa để vào khu vực ghế ngồi cao cấp (kiểm tra lại tại ranh giới tin cậy).

5. Output Validation Failures (Thất bại trong xác thực đầu ra)

- **Tầm quan trọng:** Xác thực đầu ra cũng quan trọng không kém xác thực đầu vào. Đầu ra của quy trình này thường là đầu vào của một hệ thống khác.
- **Kiểm tra tính hợp lý (Sanity Check):** Cần có các bước kiểm tra logic trước khi gửi dữ liệu đi. Ví dụ: Nếu một truy vấn đăng nhập trả về nhiều hơn một bản ghi người dùng, hệ thống phải xử lý đó là một lỗi thay vì cho phép đăng nhập,. Hoặc nếu một hóa đơn tiền điện tăng vọt gấp 100 lần mức bình thường, hệ thống cần gắn cờ cảnh báo thay vì tự động gửi đi.

General Programming Failures

Phần **General Programming Failures** (Các lỗi lập trình chung) trong Chương 8 nhấn mạnh rằng lập trình hiện đại không còn là một "môn nghệ thuật" ngẫu hứng mà là một quy trình kỹ thuật phức tạp đòi hỏi các quy tắc và hướng dẫn khắt khe để ngăn ngừa sai sót.

Dưới đây là các nội dung chi tiết:

1. Sử dụng các hàm nguy hiểm (Dangerous Functions)

- **Vấn đề:** Nhiều hàm lập trình phổ biến không tự kiểm tra kích thước đầu vào, dẫn đến lỗi tràn bộ đệm.
- **Ví dụ điển hình:** Hàm `strcpy()` trong C/C++ không kiểm tra độ dài chuỗi nguồn trước khi sao chép vào bộ đệm đích.

- **Giải pháp:** Thay thế bằng các phiên bản an toàn hơn như `strncpy()`, hàm này thực hiện kiểm tra độ dài dữ liệu dù có thể tốn thêm một chút thời gian xử lý.

2. Sử dụng mã nguồn không đáng tin cậy hoặc mã cũ (Legacy Code)

- **Rủi ro:** Việc đưa mã nguồn cũ hoặc mã từ bên thứ ba vào dự án mà không chạy qua quy trình **Vòng đời phát triển phần mềm an toàn (SDLC)** sẽ để lại các lỗ hổng chưa được kiểm tra.
- **Hệ quả:** Nhiều lỗi nghiêm trọng bắt nguồn từ việc sử dụng lại các đoạn mã cũ (legacy code) hoặc mã nội bộ đã tồn tại từ trước mà không có sự rà soát kỹ lưỡng.

3. Tầm quan trọng của rà soát mã nguồn (Static Code Analysis)

- Tất cả mã nguồn cần được kiểm tra bằng các công cụ phân tích tĩnh (static scanners) để tìm kiếm:
    - Các thư viện hoặc hàm đã lỗi thời (obsolete).
    - Các mẫu điểm yếu phổ biến (common weakness patterns).
    - Các lỗi logic như **off-by-one** (sai số một đơn vị) hoặc lỗi **không khởi tạo biến đúng cách**.

4. Lỗi về trình tự và thời gian (Sequencing and Timing)

- **Race Conditions:** Xảy ra khi nhiều luồng (threads) hoặc hệ thống cùng truy cập một đối tượng tại một thời điểm, tạo ra sự phụ thuộc lẫn nhau dẫn đến lỗi logic.
- **TOC/TOU Attack:** Kẻ tấn công lợi dụng khoảng cách thời gian giữa lúc chương trình kiểm tra một giá trị (Time of Check) và lúc nó sử dụng giá trị đó (Time of Use) để thao túng kết quả.
- **Infinite Loops (Vòng lặp vô hạn):** Kết quả của các logic điều kiện phức tạp với các trạng thái không được xử lý (unhandled states). Ví dụ: Thiết bị Microsoft Zune từng bị treo hàng loạt do lỗi xử lý ngày năm nhuận.

Technology Solutions

Phần **Technology Solutions** (Giải pháp Công nghệ) trong Chương 8 tập trung vào các đổi mới ở cấp độ phần cứng nhằm bảo vệ mã nguồn và dữ liệu khỏi các cuộc tấn công khai thác lỗi vi kiến trúc. Dưới đây là nội dung chi tiết:

1. Bối cảnh và Thách thức

- **Vấn đề:** Ngành công nghiệp vi xử lý đã đối mặt với các lỗ hổng bảo mật nghiêm trọng như **Spectre** và **Meltdown** trên các CPU Intel, cũng như các khiếm khuyết trong chipset của AMD.
- **Giải pháp:** Các nhà sản xuất đã giới thiệu các tiện ích mở rộng bảo mật vi kiến trúc để cung cấp các lớp bảo vệ mạnh mẽ hơn ngay từ cấp độ phần cứng.

2. Các Công nghệ Bảo mật Phần cứng Chính

- **Intel Software Guard Extensions (Intel SGX):**
    - Cung cấp tính năng **mã hóa bộ nhớ dựa trên phần cứng** để cô lập mã ứng dụng và dữ liệu cụ thể.
    - Cho phép mã ở cấp người dùng (user-level) cấp phát các vùng bộ nhớ riêng tư gọi là **enclaves**.
    - Các enclave này được thiết kế để bảo vệ dữ liệu khỏi các tiến trình chạy ở mức đặc quyền cao hơn (như hệ điều hành hoặc hypervisor).
- **AMD Secure Memory Encryption (SME) / Secure Encrypted Virtualization (SEV):**
    - Cung cấp khả năng mã hóa các vị trí bộ nhớ.
    - Giúp bảo vệ mã và dữ liệu, chỉ cho phép tiến trình gốc (originating process) truy cập được.
- **ARM TrustZone:** Một giải pháp công nghệ cung cấp sự bảo vệ dựa trên phần cứng cho dữ liệu và mã trong bộ nhớ.

3. Tầm quan trọng và Tác động

- **Mục tiêu:** Các tính năng này được sử dụng để ngăn chặn hàng loạt các cuộc tấn công nhắm vào mã nguồn quan trọng và ngăn kẻ tấn công can thiệp vào luồng thực thi mã.
- **Đối tượng áp dụng:** Mặc dù không nhất thiết phải áp dụng cho mọi dự án, các kỹ thuật này cực kỳ quan trọng đối với các **chức năng trọng yếu hoạt động ở mức đặc quyền cao**.
- **Sự đánh đổi:** Việc áp dụng các giải pháp này làm tăng độ phức tạp và có thể giảm hiệu suất nhẹ, nhưng các tác động này được đánh giá là ở mức có thể quản lý được.

Chapter 9 Analyze Code for Security Risks

Code Analysis (Static and Dynamic)

Nội dung phần **Code Analysis (Static and Dynamic)** trong Chương 9 tập trung vào các quy trình kiểm tra mã nguồn để tìm kiếm điểm yếu và lỗ hổng bảo mật. Dưới đây là thông tin chi tiết từ các nguồn tài liệu:

1. Tổng quan về Phân tích mã nguồn (Code Analysis)

- **Định nghĩa:** Đây là thuật ngữ mô tả các quy trình kiểm tra mã nguồn nhằm phát hiện điểm yếu và lỗ hổng bảo mật.
- **Phân loại:** Được chia thành hai hình thức chính là **phân tích tĩnh (static)** và **phân tích động (dynamic)**.
- **Nguyên tắc chi phí:** Việc phân tích nên được thực hiện ở mọi giai đoạn phát triển vì lỗ hổng được phát hiện càng sớm thì chi phí khắc phục càng rẻ.

2. Kiểm thử Bảo mật Ứng dụng Tĩnh (SAST - Static Application Security Testing)

- **Đặc điểm:** Kiểm tra mã nguồn mà **không thực thi chương trình**. Có thể thực hiện trên cả mã nguồn (source code) và mã đối tượng (object code).
- **Công cụ:** Thường sử dụng các công cụ tự động gọi là trình phân tích mã nguồn (source code analyzers) hoặc công cụ "linting".
- **Ưu điểm:**

- Độ bao phủ mã nguồn cao hơn so với kiểm tra thủ công.
- Có thể tích hợp vào môi trường phát triển (IDE) và thực hiện nhiều lần trong chu kỳ phát triển.
- Phát hiện được các lỗ hổng phức tạp ngay từ giai đoạn đầu.

- **Hạn chế:**
  - Không kiểm tra được ứng dụng trong môi trường thực tế, dễ bỏ sót các lỗi liên quan đến cấu hình hoặc logic điều kiện.
  - Tỷ lệ **dương tính giả (false positives)** cao khiến lập trình viên đôi khi e ngại sử dụng.
  - Khó thực hiện nếu không có quyền truy cập mã nguồn.

3. Kiểm thử Bảo mật Ứng dụng Động (DAST - Dynamic Application Security Testing)

- **Đặc điểm:** Thực hiện khi **phần mềm đang được thực thi** trên hệ thống thật hoặc hệ thống giả lập. Hệ thống được cung cấp các đầu vào thử nghiệm cụ thể để tạo ra các hành vi nhất định.
- **Ưu điểm:**
  - Ít dương tính giả hơn do hoạt động dựa trên kiến thức toàn diện về ứng dụng đang chạy.
  - Xác định được các vấn đề thời gian chạy (runtime) như **lỗi tranh chấp (race conditions)** và các vấn đề tương tác với môi trường (xác thực, ủy quyền).
  - Ít tốn kém và ít phức tạp hơn để triển khai so với SAST.
- **Hạn chế:**
  - Không thể kiểm tra các tiêu chuẩn lập trình nội bộ (như việc sử dụng các hàm bị cấm).
  - Khó xác định chính xác vị trí dòng mã gây ra lỗi.

- Chỉ có thể thực hiện muộn hơn trong chu kỳ phát triển vì cần mã nguồn đã hoàn thiện để chạy.

4. Kiểm thử Bảo mật Ứng dụng Tương tác (IAST) và Tự bảo vệ (RASP)

- **IAST (Interactive Application Security Testing):** Sử dụng các cảm biến (instrumentation) bên trong IDE để theo dõi luồng dữ liệu và điều khiển, cho phép phát hiện lỗi sớm và chính xác hơn.
- **RASP (Runtime Application Self-Protection):** Được triển khai khi ứng dụng **đang chạy trong thực tế (production)**. Nó hoạt động như một trình giám sát, cảnh báo hoặc ngăn chặn các hành vi bất thường và đầu vào không mong muốn ngay lập tức.

Code/Peer Review

Nội dung phần **Code/Peer Review** (Đánh giá mã nguồn/Đánh giá đồng nghiệp) trong Chương 9 tập trung vào các hoạt động nhóm nhằm kiểm tra mã nguồn một cách thủ công để phát hiện lỗi và nâng cao chất lượng phần mềm.

1. Bản chất và Mục tiêu của Đánh giá Mã nguồn

- **Tiền đề:** Hoạt động này dựa trên nguyên tắc "nhiều mắt cùng nhìn sẽ phát hiện ra những gì một người bỏ lỡ".
- **Mục tiêu chính:**
  - Tìm kiếm các điểm yếu hoặc lỗ hổng bảo mật tiềm ẩn.
  - Đảm bảo mã nguồn **sạch sẽ, dễ hiểu** để có thể bảo trì và phát triển bền vững trong tương lai.
  - Hỗ trợ các lập trình viên sản xuất mã nguồn có khả năng phòng thủ tốt hơn.

2. Quy trình thực hiện (Code Walk-through)

- **Cách thức:** Tác giả của đoạn mã sẽ giải thích cho cả nhóm từng bước một, từng dòng mã một về cách thức hoạt động của nó.
- **Vai trò của nhóm:** Các thành viên khác sẽ tìm kiếm lỗi dựa trên kinh nghiệm cá nhân, quan sát phong cách lập trình và mức độ chú thích của mã nguồn.

- **Lợi ích tâm lý:** Việc phải trình bày mã của mình trước nhóm thúc đẩy lập trình viên viết mã sạch hơn và có tính giải trình cao hơn.

3. Các vấn đề cần kiểm tra (Table 9-1)

Trong các buổi walk-through, nhóm cần tập trung kiểm tra các yếu tố sau:

- **Mã nguồn kém hiệu quả:** Đơn giản hóa nếu mã quá phức tạp hoặc bị che giấu.
- **Danh sách lỗi phổ biến:** Đối chiếu với **SANS Top 25** và **OWASP Top 10**.
- **Quản lý ngoại lệ:** Kiểm tra xem mọi hàm và thành phần đã xử lý lỗi đầy đủ chưa.
- **Lỗi tiêm nhiễm (Injection):** Kiểm tra các bước xác thực đầu vào.
- **Lời gọi mật mã:** Đảm bảo sử dụng thư viện đã được phê duyệt và số ngẫu nhiên tốt.
- **Mức đặc quyền:** Kiểm tra các vi phạm nguyên tắc **đặc quyền tối thiểu**.
- **Ghi nhật ký:** Đảm bảo ghi lại đúng các lỗi và điều kiện hệ thống.
- **Thông tin khóa bảo mật:** Cách xử lý khóa, dữ liệu định danh (PII) và dữ liệu nhạy cảm.

4. Lợi ích đối với Đội ngũ và Tổ chức

- **Phát triển nhân sự:** Các buổi đánh giá là cơ hội học tập tuyệt vời cho các thành viên cấp dưới (juniors).
- **Sở hữu chung:** Giúp các thành viên hiểu được những phần mã mà họ không trực tiếp viết, giúp việc bàn giao hoặc thay thế nhân sự trở nên dễ dàng hơn.
- **Phát hiện mã độc:** Giúp phát hiện các phần tử mã không mong muốn như **logic bombs** (bom logic), **backdoors** (cửa sau) hoặc **Easter eggs** dễ dàng hơn khi có sự giám sát của cả nhóm.
- **Đo lường độ hỗn loạn (Entropy):** Có thể tìm thấy các bí mật bị mã hóa cứng (hard-coded passwords) thông qua việc phát hiện các vùng có độ entropy cao trong mã nguồn.

Code Review Objectives

Dưới đây là nội dung chi tiết cho phần **Code Review Objectives** (Mục tiêu của Đánh giá mã nguồn) và phần tiếp theo là **Additional Sources of Vulnerability Information** (Các nguồn thông tin bổ sung về lỗ hổng) từ Chương 9:

1. Code Review Objectives (Mục tiêu của Đánh giá mã nguồn)

Đánh giá mã nguồn có nhiều mục tiêu khác nhau, từ cơ bản đến chuyên sâu:

- **Kiểm tra cú pháp và phong cách:** Ở mức cơ bản nhất, đây là hình thức kiểm tra cú pháp và phong cách lập trình để loại bỏ các lớp lỗi nhất định.
- **Thực thi quy tắc nghiệp vụ:** Đảm bảo tuân thủ các quy tắc nghiệp vụ về phong cách lập trình, xác định các lời gọi hàm bất hợp pháp và các thành phần bắt buộc (như kiểm tra lỗi).
- **Tìm kiếm vùng có độ hỗn loạn cao (High Entropy):** Đánh giá mã nguồn có thể tìm ra các vùng có độ entropy cao, thường là dấu hiệu của các **bí mật được lưu trữ trực tiếp (hard-coded passwords)** trong mã nguồn hoặc trong bộ nhớ.
- **Xác định mã độc và mã chết:** Giúp phát hiện các đoạn **mã chết** (không bao giờ được thực thi) hoặc **mã độc** như cửa sau (backdoors) và bom logic (logic bombs).
- **Hạn chế của con người:** Do quy mô của các kho mã nguồn hiện nay, việc kiểm tra thủ công bởi con người không thể mở rộng hiệu quả, vì vậy cần kết hợp với các công cụ tự động.

2. Additional Sources of Vulnerability Information (Các nguồn thông tin bổ sung về lỗ hổng)

Để đánh giá mã nguồn hiệu quả, đội ngũ cần tham khảo các danh sách lỗ hổng đã biết:

- **Danh sách lỗi nội bộ:** Đây là danh sách quan trọng nhất. Lỗi không phải là sự kiện ngẫu nhiên mà thường do các **thói quen hoặc mẫu lập trình** của đội ngũ tạo ra. Việc kiểm tra các lỗi đã từng mắc phải trong quá khứ giúp tránh lặp lại sai lầm.
- **SANS Top 25 & OWASP Top 10:** Đây là hai danh sách phổ biến nhất, bao phủ phần lớn các lỗi phổ biến hiện nay.

- **MITRE CWE và CVE:**
    - **CWE (Common Weakness Enumeration):** Danh mục các loại điểm yếu về kiến trúc, thiết kế hoặc mã nguồn.
    - **CVE (Common Vulnerabilities and Exposures):** Danh sách các lỗ hổng bảo mật cụ thể đã được xác nhận trong các sản phẩm thực tế.

Additional Sources of Vulnerability Information

Tiếp nối phần các nguồn thông tin về lỗ hổng, Chương 9 đi sâu vào chi tiết các danh mục lỗ hổng phổ biến nhất và cách sử dụng chúng để cải thiện an ninh phần mềm:

1. Danh mục CWE/SANS Top 25

Đây là kết quả của sự hợp tác giữa MITRE, SANS và nhiều chuyên gia bảo mật trên toàn thế giới.

- **Mục tiêu:** Xác định những lỗi phần mềm phổ biến và nghiêm trọng nhất có thể dẫn đến các lỗ hổng nghiêm trọng.
- **Cách sử dụng:**
    - **Giáo dục:** Nâng cao nhận thức cho đội ngũ phát triển về các loại lỗi thường gặp.
    - **Mua sắm phần mềm:** Sử dụng như một tiêu chuẩn kỹ thuật để yêu cầu các nhà cung cấp giảm thiểu rủi ro.
    - **Lập trình:** Làm bảng kiểm (checklist) để nhắc nhở lập trình viên và xây dựng bộ giải pháp giảm thiểu rủi ro (mitigation).
    - **Kiểm thử:** Giúp kiểm thử viên xây dựng các bộ kịch bản kiểm thử (test suites) để rà soát lỗi trước khi phát hành sản phẩm.
- **Một số lỗi tiêu biểu:** SQL Injection (CWE-89), Buffer Overflow (CWE-120), Hard-Coded Credentials (CWE-798), và Missing Encryption (CWE-311).

2. Danh mục OWASP Top 10

Dành riêng cho các ứng dụng web, được cung cấp bởi cộng đồng Open Web Application Security Project.

- **Tài liệu tham khảo chính:** "A Guide to Building Secure Web Applications and Web Services" cung cấp thông tin chi tiết về các lỗ hổng và cách phòng tránh.
- **Các lỗ hổng hàng đầu:** Bao gồm Injection (Tiêm mã), Broken Authentication (Lỗi xác thực), Cross-Site Scripting (XSS - Kịch bản chéo trang), và Sensitive Data Exposure (Lộ dữ liệu nhạy cảm).

3. Lỗ hổng phổ biến và Các biện pháp đối phó

- **Sự giao thoa:** Có sự chồng chéo đáng kể giữa danh sách Top 25 và Top 10, vì lập trình web là một tập con của lập trình nói chung.
- **Chiến lược đối phó:** Thay vì xử lý từng lỗi riêng lẻ, phương pháp hiệu quả hơn là **nhóm các lỗ hổng có cùng nguyên nhân** lại để áp dụng một chiến lược phòng thủ chung cho cả nhóm.

CWE/SANS Top 25 Vulnerability Categories

Tiếp nối phần nội dung về các nguồn thông tin lỗ hổng, dưới đây là chi tiết về danh mục **CWE/SANS Top 25 Vulnerability Categories** từ Chương 9:

1. Bản chất và Mục tiêu của CWE/SANS Top 25

- **Sự hợp tác toàn cầu:** Đây là kết quả của sự phối hợp giữa MITRE, SANS và nhiều chuyên gia bảo mật hàng đầu thế giới.
- **Đối tượng tập trung:** Danh sách này đại diện cho những lỗi phần mềm phổ biến và nghiêm trọng nhất, có khả năng dẫn đến các lỗ hổng nguy hiểm.
- **Đặc điểm lỗi:** Những lỗi này thường dễ tìm thấy và dễ bị khai thác. Nếu không được giảm thiểu, chúng sẽ trở thành mục tiêu hàng đầu cho kẻ tấn công, gây thiệt hại rộng lớn cho phần mềm, dữ liệu và an ninh doanh nghiệp.

2. Cách ứng dụng trong Vòng đời Phát triển (SDL)

Danh sách Top 25 phục vụ nhiều vai trò quan trọng trong quy trình phát triển phần mềm an toàn:

- **Giáo dục và Nhận thức:** Là công cụ đào tạo giúp đội ngũ phát triển hiểu về các loại lỗ hổng đang gây hại cho ngành công nghiệp phần mềm.
- **Mua sắm phần mềm (Procurement):** Được sử dụng như một bản đặc tả các yêu cầu bảo mật mà nhà cung cấp phải cam kết giảm thiểu trong phần mềm thương mại.
- **Đối với Lập trình viên:**
    - Sử dụng như một bảng kiểm (checklist) để nhắc nhở trong quá trình viết mã.
    - Là nguồn để xây dựng danh sách "Top N" tùy chỉnh cho riêng tổ chức dựa trên dữ liệu lịch sử nội bộ.
    - Xây dựng bộ giải pháp giảm thiểu (mitigation) chuẩn để áp dụng đồng nhất.
- **Đối với Kiểm thử viên (Testers):** Dùng để xây dựng các bộ kịch bản kiểm thử (test suites) nhằm đảm bảo các lỗi này đã được rà soát kỹ trước khi phát hành.

3. Danh sách các Lỗ hổng (Phiên bản hiện tại - 2011)

Mặc dù danh sách này không được cập nhật từ năm 2011, nhưng tài liệu khẳng định nó vẫn **cực kỳ quan trọng và có giá trị thực tiễn cao**. Dưới đây là một số lỗi tiêu biểu nhất:

1. **CWE-89:** SQL Injection (Tiêm mã SQL).
2. **CWE-78:** OS Command Injection (Tiêm lệnh hệ điều hành).
3. **CWE-120:** Buffer Overflow (Tràn bộ đệm).
4. **CWE-79:** Cross-Site Scripting (XSS - Kịch bản chéo trang).
5. **CWE-306:** Missing Authentication for Critical Function (Thiếu xác thực cho hàm quan trọng).
6. **CWE-862:** Missing Authorization (Thiếu ủy quyền).
7. **CWE-798:** Hard-Coded Credentials (Thông tin đăng nhập mã hóa cứng).

8. **CWE-311:** Missing Encryption of Sensitive Data (Thiếu mã hóa dữ liệu nhạy cảm). ... (Danh sách kéo dài đến 25 lỗi, bao gồm cả Integer Overflow và Path Traversal).

OWASP Vulnerability Categories

Phần **OWASP Vulnerability Categories** (Các danh mục lỗ hổng OWASP) trong Chương 9 tập trung vào các rủi ro bảo mật quan trọng nhất đối với các ứng dụng web. Dưới đây là nội dung chi tiết:

1. Cộng đồng OWASP

- **Định nghĩa:** Open Web Application Security Project (OWASP) là một cộng đồng mở, miễn phí và hoạt động trên toàn thế giới với mục tiêu tìm kiếm và chống lại các nguyên nhân gây ra phần mềm ứng dụng web không an toàn.
- **Tài liệu tham khảo:** OWASP cung cấp nhiều công cụ và tài liệu miễn phí, nổi bật nhất là cuốn "A Guide to Building Secure Web Applications and Web Services" cung cấp thông tin chi tiết về các lỗ hổng và cách phòng tránh.

2. Danh mục OWASP Top 10 (Phiên bản 2013 - Hiện hành trong tài liệu)

Danh sách này xác định 10 rủi ro bảo mật ứng dụng web quan trọng nhất:

1. **A1 – Injection:** Lỗi tiêm mã (như SQL, OS, LDAP).
2. **A2 – Broken Authentication and Session Management:** Lỗi xác thực và quản lý phiên.
3. **A3 – Cross-Site Scripting (XSS):** Kịch bản chéo trang.
4. **A4 – Insecure Direct Object References:** Tham chiếu đối tượng trực tiếp không an toàn.
5. **A5 – Security Misconfiguration:** Cấu hình sai bảo mật.
6. **A6 – Sensitive Data Exposure:** Lộ dữ liệu nhạy cảm.
7. **A7 – Missing Function-Level Access Control:** Thiếu kiểm soát truy cập mức chức năng.

8. **A8 – Cross-Site Request Forgery (CSRF):** Giả mạo yêu cầu chéo trang.
9. **A9 – Using Known Vulnerable Components:** Sử dụng các thành phần có lỗ hổng đã biết.
10. **A10 – Unvalidated Redirects and Forwards:** Chuyển hướng và chuyển tiếp không được xác thực.

3. Mối quan hệ với CWE/SANS Top 25

- **Sự giao thoa:** Có sự chồng chéo đáng kể giữa hai danh sách này. **Tất cả 10 mục trong OWASP Top 10 đều nằm trong danh sách CWE/SANS Top 25**.
- **Lý do:** Lập trình ứng dụng web là một tập con của lĩnh vực lập trình nói chung.
- **Chiến lược đối phó:** Thay vì xử lý từng lỗ hổng đơn lẻ, phương pháp hiệu quả nhất là **nhóm các lỗ hổng có cùng nguyên nhân** lại để áp dụng một chiến lược phòng thủ chung có thể giải quyết nhiều vấn đề cùng lúc.

Common Vulnerabilities and Countermeasures

Dưới đây là nội dung chi tiết cho phần **Common Vulnerabilities and Countermeasures** (Các lỗ hổng phổ biến và biện pháp đối phó) từ Chương 9, tập trung vào các cuộc tấn công dạng Injection và các phương pháp phòng vệ tương ứng.

1. Chiến lược Đối phó Tổng quát

- **Sự giao thoa giữa các danh sách:** Có sự chồng chéo đáng kể giữa danh sách SANS Top 25 và OWASP Top 10. Tất cả 10 mục của OWASP đều xuất hiện trong Top 25.
- **Tiếp cận theo nhóm:** Thay vì xử lý từng lỗ hổng đơn lẻ, phương pháp hiệu quả hơn là **nhóm các lỗ hổng có cùng nguyên nhân** để áp dụng một chiến lược phòng thủ chung.
- **Nguyên tắc vàng về đầu vào:** Không bao giờ để người dùng trực tiếp định nghĩa các yếu tố hành vi của chương trình. Việc lọc (cleansing) đầu vào là rất khó; thay vào đó, hãy sử dụng **danh sách trắng (whitelisting)** các tùy chọn đã được phê duyệt để người dùng lựa chọn.

2. Các cuộc tấn công Tiêm mã (Injection Attacks)

Đây là nhóm tấn công phổ biến và nghiêm trọng nhất hiện nay, bao gồm:

SQL Injection (CWE-89)

- **Cơ chế:** Kẻ tấn công nhập các chuỗi ký tự đặc biệt (như dấu `--` để biến phần còn lại của câu lệnh thành chú thích) nhằm thao túng truy vấn cơ sở dữ liệu,.
- **Biện pháp đối phó:**
    - Sử dụng **thủ tục lưu sẵn (stored procedures)** là phương pháp an toàn nhất.
    - Sử dụng **truy vấn có tham số (parameterized queries)**.
    - Nâng cao nhận thức của lập trình viên là yếu tố then chốt.

OS Command Injection (CWE-78)

- **Cơ chế:** Lợi dụng việc chương trình gọi các lệnh hệ điều hành thông qua đầu vào của người dùng không được kiểm soát.
- **Biện pháp đối phó:** Thiết kế để loại bỏ nhu cầu gọi lệnh hệ điều hành từ đầu vào hoặc thực hiện xác thực đầu vào cực kỳ nghiêm ngặt.

Path Traversal (CWE-22)

- **Cơ chế:** Sử dụng các ký tự như `../` (hoặc mã hóa hex của chúng) để truy cập các tệp nằm ngoài thư mục gốc của trang web.
- **Biện pháp đối phó:** Không sử dụng đầu vào người dùng trực tiếp khi truy cập tài nguyên cục bộ. Hãy khớp đầu vào với một danh sách tài nguyên cụ thể và sử dụng **đường dẫn được mã hóa cứng (hard-coded paths)**.

Cross-Site Scripting (XSS - CWE-79)

- **Cơ chế:** Tiêm các kịch bản (scripts) độc hại vào trang web để thực thi trên trình duyệt của người dùng khác.
- **Phân loại:** Bao gồm kịch bản chéo trang không lưu trữ (non-persistent), lưu trữ (persistent) và dựa trên DOM (DOM-based).

- **Biện pháp đối phó:** Sử dụng các **thư viện chống XSS (anti-XSS libraries)** để loại bỏ script khỏi đầu vào, kết hợp với giới hạn loại và kích thước tệp tải lên.

Cross-Site Request Forgery (CSRF - CWE-352)

- **Cơ chế:** Lợi dụng sự tin tưởng của một trang web vào phiên làm việc đã được xác thực của người dùng (vấn đề "người đại diện bị nhầm lẫn" - confused deputy) để thực hiện các yêu cầu trái phép.
- **Biện pháp đối phó:** Phương pháp mạnh nhất là sử dụng **mã thông báo CSRF ngẫu nhiên (random CSRF tokens)** cho mỗi lần gửi biểu mẫu.

Chapter 10 Implement Security Controls

Security Risks

Nội dung phần **Security Risks** (Rủi ro Bảo mật) trong Chương 10 tập trung vào các phương án mà ban quản lý có thể lựa chọn để xử lý rủi ro sau khi chúng đã được xác định. Có bốn phương án chính để đối phó với rủi ro: **Remediation** (Khắc phục), **Mitigation** (Giảm thiểu), **Transfer** (Chuyển giao) và **Accept** (Chấp nhận).

Dưới đây là chi tiết về từng phương án:

1. Remediation (Khắc phục/Sửa lỗi)

- Đây là **phương án được ưu tiên nhất** vì nó giải quyết triệt để vấn đề.
- Khắc phục bao gồm việc hiểu rõ nguyên nhân thực sự của lỗ hổng và sửa đổi mã nguồn hoặc thiết kế để loại bỏ nó.
- Việc sửa lỗi sẽ dễ dàng và ít tốn kém hơn nếu vấn đề được phát hiện sớm trong chu kỳ phát triển.

2. Mitigation (Giảm thiểu)

- Phương án này tập trung vào việc làm yếu đi hoặc ngăn chặn các yếu tố rủi ro liên quan đến lỗ hổng.
- Có thể thực hiện bằng cách **loại bỏ tính năng** gây ra lỗ hổng hoặc áp dụng các **kiểm soát bù đắp (compensating controls)**.

- Ví dụ: Nếu một giao thức truyền tin ở dạng văn bản thuần túy (cleartext) không thể thay thế, việc thêm mã hóa vào kênh truyền thông đó là một hình thức giảm thiểu để loại bỏ nguy cơ lộ lọt thông tin.

3. Transfer (Chuyển giao rủi ro)

- Chuyển giao rủi ro là việc đẩy tác động của rủi ro sang một bên thứ ba.
- Các hình thức phổ biến bao gồm:
    - **Cảnh báo người dùng:** Đẩy trách nhiệm bảo vệ phần mềm sang phía người dùng.
    - **Mua bảo hiểm:** Chuyển tác động tài chính sang công ty bảo hiểm.
    - **Cơ chế ngân hàng:** Trong gian lận thẻ tín dụng, rủi ro thường được chuyển từ người bán sang bên phát hành thẻ (và cuối cùng là phân bổ chi phí lên tất cả khách hàng qua lãi suất).

4. Acceptance (Chấp nhận rủi ro)

- Đây là phương án **không làm gì cả**, đồng nghĩa với việc chấp nhận mọi hậu quả nếu rủi ro xảy ra.
- Việc chấp nhận rủi ro chỉ được coi là hợp lý khi nó là một **quyết định có chủ đích**, dựa trên sự hiểu biết đầy đủ về các hệ quả tiềm tàng.
- Thông thường, rủi ro được chấp nhận khi nó ở mức **rủi ro tồn dư (residual risk)** — tức là phần rủi ro còn lại sau khi các biện pháp tránh, loại bỏ hoặc giảm thiểu đã được thực hiện.

Ưu tiên và Đánh giá rủi ro

Để quyết định xem lỗi nào cần được sửa trước, các đội ngũ phát triển thường sử dụng mô hình **DREAD** để tính toán mức độ rủi ro:

- **Công thức:** `Rủi ro = Tác động (Impact) × Xác suất (Probability).`
- Các lỗi có điểm số rủi ro cao nhất (như lỗi cho phép leo thang đặc quyền hoặc từ chối dịch vụ) sẽ được ưu tiên khắc phục trước.

Implement Security Controls

Nội dung của **Chương 10: Implement Security Controls** (Triển khai các biện pháp kiểm soát bảo mật) tập trung vào việc áp dụng các biện pháp kỹ thuật và quản lý để giảm thiểu rủi ro trong suốt quá trình xây dựng và tích hợp phần mềm. Dưới đây là các phần chi tiết:

1. Quản lý Rủi ro Bảo mật (Security Risks)

Trước khi triển khai kiểm soát, tổ chức cần xác định cách xử lý các rủi ro đã được nhận diện thông qua bốn lựa chọn chính:

- **Remediation (Khắc phục):** Sửa chữa tận gốc lỗ hổng để loại bỏ rủi ro.
- **Mitigation (Giảm thiểu):** Áp dụng các biện pháp kiểm soát bù đắp để giảm bớt tác động hoặc khả năng xảy ra của rủi ro khi không thể sửa chữa hoàn toàn.
- **Transfer (Chuyển giao):** Chuyển rủi ro sang bên thứ ba, ví dụ như thông qua bảo hiểm hoặc cảnh báo người dùng.
- **Accept (Chấp nhận):** Chấp nhận rủi ro hiện hữu sau khi đã cân nhắc các chi phí và lợi ích.

2. Triển khai các biện pháp kiểm soát cụ thể

Các biện pháp kiểm soát bảo mật là các bước bổ sung nhằm đảm bảo mã nguồn chỉ thực hiện đúng chức năng được thiết kế. Một số biện pháp phổ biến bao gồm:

- **Watchdogs (Cơ chế giám sát):** Các tiến trình kiểm tra để đảm bảo hệ thống hoạt động bình thường, không bị kẹt trong vòng lặp hoặc lỗi (ví dụ: chỉ số TTL trong giao thức mạng).
- **Heartbeat (Nhịp đập):** Tín hiệu định kỳ để báo hiệu một tiến trình vẫn đang chạy.
- **File Integrity Monitoring - FIM (Giám sát tính toàn vẹn của tệp):** Sử dụng mã băm (hash) hoặc checksum để xác minh tệp cấu hình hoặc dữ liệu quan trọng không bị giả mạo trước khi ứng dụng sử dụng.

3. Bảo mật trong môi trường Build (Build Environment)

Việc áp dụng bảo mật ngay trong quá trình xây dựng phần mềm giúp phát hiện lỗi sớm và tự động hóa việc thực thi chính sách.

- **Compiler Switches (Tùy chọn trình biên dịch):** Sử dụng các cờ (flags) như `/GS` của Microsoft để bảo vệ chống tràn ngăn xếp (stack overflow) hoặc `/SAFEH` để xử lý ngoại lệ an toàn.
- **Safe Libraries (Thư viện an toàn):** Sử dụng các thư viện đã được kiểm chứng (như OWASP ESAPI hoặc Microsoft Anti-XSS) thay vì tự viết các hàm xử lý rủi ro cao.

4. Kỹ thuật chống giả mạo (Anti-tampering Techniques)

Các kỹ thuật này khiến kẻ tấn công khó có thể thay đổi mã nguồn hoặc hiểu cách thức hoạt động của nó.

- **Code Signing (Ký mã nguồn):** Sử dụng chữ ký số để xác thực nguồn gốc và đảm bảo tính toàn vẹn của phần mềm khi phân phối qua Internet.
- **Code Obfuscation (Làm mờ mã nguồn):** Sắp xếp lại mã thực thi để ngăn chặn việc dịch ngược (reverse-engineering) và phân tích các bí mật như thuật toán hoặc khóa mã hóa bên trong chương trình.

5. Lập trình phòng thủ (Defensive Coding Techniques)

Lập trình phòng thủ yêu cầu ứng dụng phải đứng vững trước các điều kiện thay đổi hoặc các cuộc tấn công.

- **Cryptographic Agility (Tính linh hoạt mật mã):** Khả năng thay đổi thuật toán mã hóa thông qua tệp cấu hình mà không cần biên dịch lại mã nguồn.
- **Interface Coding (Lập trình giao diện):** Xác thực mọi đầu vào giao diện và kiểm toán (audit) các hoạt động đặc quyền.
- **Memory Management (Quản lý bộ nhớ):** Sử dụng mã nguồn được quản lý (Managed code như .NET, Java) để tự động hóa việc dọn rác và kiểm tra biên, giúp tránh lỗi rò rỉ bộ nhớ.
- **ASLR (Address Space Layout Randomization):** Kỹ thuật ngẫu nhiên hóa sơ đồ không gian địa chỉ để bảo vệ chống lại các cuộc tấn công chiếm quyền điều khiển bộ nhớ.

6. Tích hợp và Tái sử dụng an toàn (Secure Integration & Reuse)

- **Tái sử dụng mã nguồn bên thứ ba:** Sử dụng các công cụ **SCA (Software Composition Analysis)** để quét các thư viện nguồn mở nhằm phát hiện lỗ hổng đã biết và quản lý tính tuân thủ giấy phép.
- **Tích hợp hệ thống (System-of-Systems):** Thiết lập các "Hợp đồng tin cậy" (Trust Contracts) để xác thực lại dữ liệu mỗi khi nó đi qua ranh giới tin cậy (trust boundary) giữa các phân hệ.

Applying Security via the Build Environment

Nội dung phần **Applying Security via the Build Environment** (Áp dụng bảo mật thông qua môi trường xây dựng) trong Chương 10 tập trung vào việc tận dụng các công cụ và thiết lập trong quá trình sản xuất phần mềm để đảm bảo an ninh. Dưới đây là các chi tiết cụ thể:

1. Quy trình xây dựng hiện đại

Việc tạo ra phần mềm trong môi trường phát triển hiện đại là một quy trình đa bước, bao gồm: biên dịch (compiling), liên kết (linking), kiểm thử (testing), đóng gói (packaging - bao gồm cả ký mã nguồn) và phân phối (distribution). Mỗi tác vụ này thường có một hoặc một bộ công cụ hỗ trợ riêng.

2. Tầm quan trọng của thiết lập công cụ

- **Cấu hình chính xác:** Việc thiết lập các tùy chọn (options) đúng cho công cụ (như trình biên dịch) là cực kỳ quan trọng vì chúng quyết định các bước kiểm tra lỗi nào sẽ được thực hiện trong quá trình biên dịch.
- **Tuân thủ SDL:** Các tổ chức áp dụng vòng đời phát triển an toàn (SDL) cần có quy trình rõ ràng để đảm bảo các công cụ được sử dụng với cấu hình chuẩn. Điều này giúp loại bỏ các lỗi đáng lẽ phải được phát hiện ngay từ giai đoạn phát triển.

3. Tùy chọn trình biên dịch (Compiler Switches)

Các trình biên dịch cung cấp các "cờ" (flags) để kích hoạt các tính năng bảo mật tích hợp sẵn.

- **Ví dụ tiêu biểu:** Cờ `/GS` của Microsoft giúp kích hoạt tính năng bảo vệ chống tràn ngăn xếp (stack overflow) bằng cách sử dụng một "cookie bảo mật" để kiểm tra trước khi sử dụng địa chỉ trả về của hàm.
- **Lợi ích:** Sử dụng các tùy chọn này giúp loại bỏ các điều kiện tràn ngăn xếp phổ biến và tăng cường độ an toàn cho mã nguồn.
- **Yêu cầu:** Các thiết lập cho trình biên dịch, trình liên kết và công cụ phân tích mã nên được đội ngũ bảo mật xác định và công bố như một phần của yêu cầu bảo mật trong quy trình SDL.

4. Sử dụng các thư viện an toàn (Safe Libraries)

Thay vì tự viết các hàm xử lý cho các tác vụ có rủi ro cao, tổ chức nên xác định và sử dụng các thư viện an toàn đã được kiểm chứng.

- **Mục tiêu:** Giảm thiểu khả năng xảy ra lỗi gọi hàm đối với các vấn đề phổ biến như tràn bộ đệm, tấn công XSS hoặc tấn công tiêm nhiễm (injection).
- **Ví dụ:** Dự án **OWASP Enterprise Security API (ESAPI)** hoặc thư viện **Microsoft Anti-Cross Site Scripting** cho .NET.

5. Môi trường phát triển tích hợp (IDE)

- **Tự động hóa:** Các công cụ tự động có thể được tích hợp trực tiếp vào IDE (như Microsoft Visual Studio), giúp lập trình viên thực hiện các kiểm tra tĩnh và động một cách tự động và thường xuyên.
- **Năng suất:** Việc sử dụng IDE tiên tiến giúp loại bỏ hàng loạt lỗi đơn giản và cho phép theo dõi các số liệu (metrics) quan trọng.
- **Hiệu quả lâu dài:** Mặc dù đội ngũ phát triển cần thời gian để làm quen với các công cụ này, nhưng thời gian tiết kiệm được từ việc giảm thiểu công sức sửa lỗi ở các giai đoạn sau (kiểm thử hoặc vận hành) là rất lớn.

Anti-tampering Techniques

Nội dung phần **Anti-tampering Techniques** (Các kỹ thuật chống giả mạo) trong Chương 10 tập trung vào việc bảo vệ mã nguồn để ngăn chặn những thay đổi trái phép từ kẻ tấn công. Dưới đây là các chi tiết cụ thể từ nguồn tài liệu:

1. Mục tiêu của kỹ thuật chống giả mạo

Các kỹ thuật này được áp dụng nhằm khiến kẻ tấn công khó có thể thay đổi mã nguồn và sau đó giả mạo phần mềm đó là sản phẩm chính hãng. Hai phương pháp chính được sử dụng là **Ký mã nguồn (Code signing)** và **Làm mờ mã nguồn (Obfuscation)**.

2. Ký mã nguồn (Code Signing)

Ký mã nguồn cho phép người dùng cuối xác minh tính toàn vẹn của mã và cung cấp bằng chứng về nguồn gốc của phần mềm.

- **Cơ chế:** Kỹ thuật này dựa trên hạ tầng khóa công khai (PKI). Để sử dụng, nhà phát triển cần một cặp khóa được ký bởi một cơ quan chứng thực (CA) đáng tin cậy.
- **Các bước thực hiện:**
    1. Tác giả tạo một bản băm (hash) một chiều của mã nguồn để tạo ra một bản tóm lược (digest).
    2. Bản tóm lược này được mã hóa bằng khóa bí mật (private key) của người ký.
    3. Mã nguồn cùng với bản tóm lược đã ký được chuyển đến người dùng.
    4. Người dùng tạo bản băm của mã nhận được bằng cùng một hàm băm.
    5. Người dùng giải mã bản tóm lược đã ký bằng khóa công khai (public key) của người gửi.
    6. Nếu hai bản tóm lược khớp nhau, mã nguồn được xác thực và tính toàn vẹn được đảm bảo.
- **Lưu ý:** Ký mã nguồn chỉ chứng minh mã không bị thay đổi kể từ khi ký; nó **không đảm bảo phần mềm không có lỗi hoặc lỗ hổng**.

3. Quản lý cấu hình: Mã nguồn và Phiên bản (Configuration Management: Source Code and Versioning)

Việc quản lý các thay đổi và phiên bản của mã nguồn là rất quan trọng đối với các ứng dụng hiện đại.

- **Mục tiêu:** Đánh dấu và quản lý duy nhất từng bản phát hành khác nhau (thường thông qua số phiên bản).
- **Hệ thống kiểm soát phiên bản:** Giúp theo dõi tất cả các thành phần, cho phép quay lại các phiên bản cũ hơn khi cần và kiểm soát quyền truy cập để tránh việc nhiều lập trình viên ghi đè lên công việc của nhau.
- **Tự động hóa:** Do tính chi tiết và khối lượng công việc lớn, việc quản lý phiên bản nên được thực hiện thông qua các hệ thống tự động để loại bỏ sai sót của con người.

4. Làm mờ mã nguồn (Code Obfuscation)

Kỹ thuật này được sử dụng khi mã thực thi bị lộ ra môi trường bên ngoài, nơi kẻ tấn công có thể sao chép và phân tích thông qua các chương trình gỡ lỗi (debuggers).

- **Tác dụng:** Sử dụng các chương trình làm mờ (obfuscators) để sắp xếp lại mã thực thi, khiến quá trình dịch ngược (decompiling) trở nên cực kỳ khó khăn.
- **Mục tiêu bảo vệ:** Ngăn chặn kẻ tấn công phát hiện ra các bí mật bên trong phần mềm như các thuật toán hoặc khóa mã hóa.
- **Tính chất:** Đây là một cuộc chạy đua vũ trang giữa người bảo vệ và kẻ tấn công; mặc dù không hoàn hảo, nó nâng cao rào cản và buộc kẻ tấn công phải bắt đầu lại từ đầu với mỗi bản phát hành mới.

Defensive Coding Techniques

Nội dung phần **Defensive Coding Techniques** (Các kỹ thuật lập trình phòng thủ) trong Chương 10 tập trung vào việc xây dựng mã nguồn có khả năng chống lại các cuộc tấn công và hoạt động ổn định ngay cả khi điều kiện môi trường thay đổi.

Dưới đây là các kỹ thuật và nguyên tắc cốt lõi được trình bày trong tài liệu:

1. Khái niệm và Thành phần nền tảng

Lập trình phòng thủ không phải là một "nghệ thuật đen tối" mà là việc áp dụng các báo cáo về mối đe dọa vào thực tế. Các yếu tố nền tảng bao gồm:

- **Giảm thiểu bề mặt tấn công (Attack surface reduction)**.
- **Hiểu rõ các lỗ hổng lập trình phổ biến**.
- **Áp dụng các biện pháp giảm thiểu tiêu chuẩn**.

2. Bảo mật Khai báo (Declarative) so với Bảo mật Lập trình (Programmatic)

Đây là quyết định ở mức độ thiết kế về cách thực thi bảo mật:

- **Bảo mật khai báo:** Định nghĩa các quy tắc bảo mật với "vỏ bọc" (container) bên ngoài mã nguồn (ví dụ: cấu hình trong tệp XML). Kỹ thuật này linh hoạt và được quản lý bởi đội ngũ vận hành.
- **Bảo mật lập trình (Imperative):** Các logic bảo mật được nhúng trực tiếp vào mã nguồn. Nó cho phép kiểm soát chi tiết các quy tắc nghiệp vụ phức tạp nhưng khiến mã nguồn khó tái sử dụng hoặc di chuyển hơn.

3. Tự khởi động an toàn (Bootstrapping)

Đề cập đến quy trình khởi động tự duy trì của chương trình. Khi lập trình, cần đảm bảo các biến môi trường (như đường dẫn PATH) không bị thay đổi bởi các tác nhân bên ngoài kiểm soát của ứng dụng. Sử dụng các tệp cấu hình được bảo vệ để quản lý các yếu tố khởi động.

4. Tính linh hoạt mật mã (Cryptographic Agility)

Đây là khả năng thay đổi các thuật toán hoặc cấu hình mật mã mà **không cần biên dịch lại mã nguồn**, thường thông qua các tệp cấu hình. Điều này giúp ứng dụng phản ứng nhanh khi một thuật toán bị coi là lỗi thời (như DES hoặc MD5) hoặc khi cần tuân thủ các luật xuất khẩu mật mã khác nhau giữa các quốc gia.

5. Quản lý tham số cấu hình

Các tham số cấu hình có thể thay đổi hành vi của ứng dụng, vì vậy chúng cần được bảo vệ tương ứng với mức độ rủi ro. Các biện pháp bao gồm đặt tệp trong thư mục được bảo vệ bởi ACL hoặc mã hóa dữ liệu nhạy cảm bên trong tệp cấu hình.

6. Lập trình giao diện (Interface Coding)

Các API là những điểm nhập (entry points) quan trọng vào phần mềm. Kỹ thuật lập trình phòng thủ yêu cầu:

- Xác thực mọi đầu vào tại giao diện.
- Kiểm toán (audit) các tương tác bên ngoài đối với mọi thao tác đặc quyền được thực hiện qua giao diện.
- Loại bỏ các API cũ, yếu hoặc đã lỗi thời (deprecated).

7. Quản lý bộ nhớ (Memory Management)

Đây là khía cạnh cực kỳ quan trọng đối với bảo mật mã nguồn:

- **Mã nguồn được quản lý (Managed code):** Các ngôn ngữ như Java hoặc .NET chạy trong một "hộp cát" (sandbox), tự động hóa việc dọn rác (garbage collection), kiểm tra chỉ số và quản lý bộ nhớ.
- **Mã nguồn không được quản lý (Unmanaged code):** Lập trình viên phải tự quản lý việc cấp phát bộ nhớ, khiến chương trình dễ bị rò rỉ bộ nhớ, tràn bộ đệm hoặc lỗi ghi đè con trỏ.
- **Tính an toàn kiểu (Type-safe):** Ngăn chặn lỗi do sai lệch kiểu dữ liệu, từ đó ngăn việc truy cập trái phép vào các vị trí bộ nhớ ngoài phạm vi dự kiến.
- **Tính cục bộ (Locality):** Sử dụng các kỹ thuật như **ASLR** để ngẫu nhiên hóa sơ đồ không gian địa chỉ, chống lại các cuộc tấn công dựa trên tính dự đoán của các tham chiếu bộ nhớ.

8. Các biện pháp giảm thiểu chính (Primary Mitigations)

Tài liệu liệt kê các "công cụ chuẩn" cho lập trình viên phòng thủ:

- Kiểm soát chặt chẽ mọi đầu vào và đầu ra.
- Giả định rằng các thành phần bên ngoài có thể bị phá hoại và mã nguồn có thể bị đọc bởi bất kỳ ai.

- Sử dụng các thư viện/khung (frameworks) an toàn và các tính năng bảo mật đã được ngành công nghiệp chấp nhận thay vì tự sáng tạo.

Primary Mitigations

Nội dung phần **Primary Mitigations** (Các biện pháp giảm thiểu chính) trong Chương 10 mô tả các công cụ và kỹ thuật chuẩn đã được chứng minh qua thực tế là các biện pháp tốt nhất (best practices) để bảo vệ phần mềm. Dưới đây là các chi tiết cụ thể:

1. Khái niệm và Vai trò

- **Công cụ chuẩn:** Đây là tập hợp các biện pháp giảm thiểu được thiết lập qua thời gian và được coi là những công cụ tiêu chuẩn mà một chuyên gia bảo mật phần mềm (CSSLP) cần phải có trong "hộp dụng cụ" của mình.
- **Kiến thức thiết yếu:** Việc hiểu rõ từng biện pháp, cũng như cách thức và vị trí áp dụng chúng, là kiến thức bắt buộc đối với tất cả các thành viên trong đội ngũ phát triển.
- **Triển khai qua báo cáo mối đe dọa:** Các biện pháp này thường được áp dụng thông qua việc sử dụng **báo cáo mối đe dọa (threat report)**.

2. Nền tảng của Lập trình Phòng thủ

Lập trình phòng thủ không phải là một "nghệ thuật đen tối" mà đơn giản là việc áp dụng các nội dung chi tiết trong báo cáo mối đe dọa. Các yếu tố nền tảng bao gồm:

- **Giảm thiểu bề mặt tấn công (Attack surface reduction)**.
- **Hiểu rõ các lỗ hổng lập trình phổ biến**.
- **Áp dụng các biện pháp giảm thiểu tiêu chuẩn**.

3. Các công cụ bổ sung trong bộ công cụ phòng thủ

Ngoài các biện pháp giảm thiểu chính, bộ công cụ lập trình phòng thủ còn bao gồm các hạng mục sau:

- **Phân tích mã nguồn (Code analysis)** và **Rà soát mã nguồn (Code review)**.
- **Quản lý phiên bản (Versioning)**.

- **Tính linh hoạt mật mã (Cryptographic agility)**.
- **Quản lý bộ nhớ (Memory management)** và **Xử lý ngoại lệ (Exception handling)**
- **Lập trình giao diện (Interface coding)** và **Mã nguồn được quản lý (Managed code)**.

4. Kỹ thuật Tokenization (Mẹo thi)

Một biện pháp giảm thiểu cụ thể được nhấn mạnh để bảo vệ dữ liệu nhạy cảm là **Tokenization (Mã báo hiệu)**:

- **Cơ chế:** Thay thế dữ liệu nhạy cảm bằng một dữ liệu khác không có kết nối bên ngoài với dữ liệu gốc.
- **Ví dụ:** Trong giao dịch thẻ tín dụng, nhà hàng thường chỉ in vài số cuối của thẻ và thay các số còn lại bằng ký tự "XXXX", tạo ra một token cho dữ liệu đó mà không làm lộ dữ liệu thực tế.

Secure Integration of Components

Phần **Secure Integration of Components** (Tích hợp các thành phần an toàn) trong Chương 10 tập trung vào việc kết nối các phân hệ, thư viện và mã nguồn từ nhiều nguồn khác nhau vào một hệ thống thống nhất mà vẫn đảm bảo an ninh. Dưới đây là nội dung chi tiết:

1. Bối cảnh Tích hợp hiện đại

- **Quy trình bồi đắp (Accretive process):** Các hệ thống CNTT hiện đại không tồn tại đơn lẻ mà phát triển qua quá trình bồi đắp, nơi các hệ thống mới được thiết kế để đáp ứng yêu cầu rồi gia nhập vào hệ thống chung của doanh nghiệp.
- **Lợi ích:** Việc tích hợp chéo giữa các kiến trúc cho phép tái sử dụng dữ liệu, tăng hiệu quả sử dụng kiến trúc tổng thể và giảm chi phí so với việc xây dựng lại các dịch vụ dữ liệu hiện có.
- **Yêu cầu an toàn:** Tất cả các hoạt động như mua phần mềm, áp dụng giải pháp mã nguồn mở hay tạo mã nguồn tùy chỉnh đều phải diễn ra dưới một quy trình bảo mật chặt chẽ để đảm bảo mọi rủi ro đều được tính toán.

2. Tái sử dụng an toàn mã nguồn hoặc thư viện bên thứ ba

Việc viết một chương trình dài từ đầu đến cuối hiện nay đã lỗi thời; thay vào đó, phần mềm được cấu thành từ các mô-đun làm việc cùng nhau.

- **Lợi ích của mô-đun:** Giúp bảo trì hiệu quả, cho phép tái sử dụng mã nguồn có sẵn (như các thuật toán băm, mã hóa) giúp tiết kiệm thời gian và tăng tính an toàn vì các lỗi lý thuyết đã được loại bỏ từ trước.
- **Software Composition Analysis (SCA - Phân tích thành phần phần mềm):** Đây là tập hợp các phương pháp dùng để quản lý thông tin liên quan đến các thư viện và phân đoạn mã nguồn bên thứ ba.
- **Công cụ SCA:** Các công cụ này tự động quét mã nguồn để nhận diện:
    - Tất cả các thành phần mã nguồn mở.
    - Dữ liệu tuân thủ giấy phép (license compliance).
    - Các lỗ hổng bảo mật đã biết.
    - Các tạo tác liên quan như container, registry và thư viện nhập khẩu.

3. Tích hợp Hệ thống-trong-Hệ thống (System-of-Systems Integration)

Việc tích hợp bao gồm việc kết nối các phân hệ khác nhau thành một thực thể hoạt động hoàn chỉnh.

- **Ranh giới tin cậy (Trust boundary):** Bất cứ khi nào dữ liệu di chuyển từ hệ thống này sang hệ thống khác, nó đều đi qua một ranh giới tin cậy.
- **Xác thực lại (Revalidation):** Mỗi khi dữ liệu băng qua ranh giới tin cậy, nó phải được xác thực lại là chính xác để ngăn chặn các đầu vào không phù hợp hoặc thù địch vào hệ thống.
- **Hợp đồng tin cậy (Trust contracts):** Đây là các tài liệu mô tả yêu cầu về độ tin cậy khi truyền dữ liệu giữa các thành phần. Ví dụ: Hợp đồng không chỉ quy định dữ liệu là số nguyên (integer), mà còn phải chi tiết là bao nhiêu bit (8, 16, 32 bit), có dấu hay không dấu, phạm vi giá trị hợp lệ là bao nhiêu. Việc này giúp nhà cung cấp dữ

liệu không gửi nhầm dữ liệu bị từ chối và cung cấp thông tin cần thiết cho việc kiểm thử bảo mật hiệu quả.

Part V Secure Software Testing

chapter 11 Security Test Cases

Security Test Cases

Dưới đây là nội dung chi tiết của phần **Security Test Cases** (Các trường hợp kiểm thử bảo mật) trong Chương 11, ngay trước phần Đánh giá Bề mặt Tấn công:

Tổng quan về Security Test Cases

- **Định nghĩa:** Một trường hợp kiểm thử bảo mật (**Security Test Case**) là một tài liệu mô tả một đầu vào (input), hành động hoặc sự kiện được mong đợi sẽ tạo ra một **phản hồi có thể dự đoán được**.
- **Mục tiêu cốt lõi:** Mục đích cơ bản của tất cả các trường hợp kiểm thử là để xác định xem một tính năng cụ thể trong hệ thống hoặc sản phẩm phần mềm có đang **hoạt động đúng chức năng** hay không.
- **Các thành phần chính:** Một trường hợp kiểm thử tiêu chuẩn nên chứa các thông tin cụ thể như:
    - Mã định danh trường hợp kiểm thử (test case identifier).
    - Tên trường hợp kiểm thử.
    - Mục tiêu kiểm thử.
    - Các điều kiện hoặc thiết lập kiểm thử.
    - Yêu cầu về dữ liệu đầu vào.
    - Các bước thực hiện và **kết quả mong đợi**.

Vai trò và Quy trình thực hiện

- **Phát hiện sớm vấn đề:** Quá trình xây dựng các trường hợp kiểm thử có thể giúp phát hiện các vấn đề ngay từ khâu xác định yêu cầu hoặc thiết kế, vì nó bắt buộc đội ngũ phải **suy nghĩ thấu đáo** về cách thức vận hành của ứng dụng. Do đó, việc chuẩn bị các trường hợp kiểm thử càng sớm càng tốt trong quy trình phát triển là một thói quen hữu ích.
- **Kiểm soát điều kiện:** Về mặt chức năng, kiểm thử bao gồm việc xem xét một hệ thống hoặc ứng dụng dưới các **điều kiện được kiểm soát** và sau đó đánh giá kết quả.
- **Tư duy thử sai:** Các điều kiện kiểm soát này nên bao gồm cả điều kiện **bình thường và bất thường**. Kiểm thử nên được thiết kế để **cố tình làm cho mọi thứ đi sai hướng** nhằm xác định những sự kiện không mong muốn có thể xảy ra, hoặc những sự kiện mong muốn lại không xảy ra.

Hai danh mục kiểm thử chính

Các phương pháp kiểm thử thường được chia thành hai loại chính dựa trên mức độ tiếp cận mã nguồn:

1. **White-box testing (Kiểm thử hộp trắng):** Các kỹ thuật cố gắng thực thi càng nhiều mã nguồn càng tốt trong phạm vi các ràng buộc về tài nguyên.
2. **Black-box testing (Kiểm thử hộp đen):** Các kỹ thuật không xem xét đến cấu trúc của mã nguồn khi lựa chọn các trường hợp kiểm thử.

Tùy thuộc vào mức độ toàn vẹn yêu cầu đối với sản phẩm mà các phương pháp này sẽ được triển khai theo những cách thức khác nhau.

Attack Surface Evaluation

Phần **Attack Surface Evaluation (Đánh giá Bề mặt Tấn công)** trong Chương 11 tập trung vào việc xác định và đo lường các điểm mà kẻ tấn công có thể lợi dụng để xâm nhập hoặc trích xuất dữ liệu từ hệ thống. Dưới đây là nội dung chi tiết:

1. Khái niệm về Bề mặt Tấn công

- **Định nghĩa:** Bề mặt tấn công là tập hợp tất cả các điểm nằm trên ranh giới của một hệ thống, thành phần hệ thống hoặc môi trường mà kẻ tấn công có thể cố gắng truy cập để gây ảnh hưởng hoặc lấy dữ liệu.
- **Vị trí trọng yếu:** Trong hầu hết các hệ thống phần mềm, đây chính là những nơi **dữ liệu hoặc thông tin đi vào hệ thống**, dù là thông qua các kênh hợp lệ hay trái phép.

2. Quy trình Đánh giá và Kiểm thử

- **Giai đoạn thiết kế:** Nhóm phát triển cần ước tính rủi ro và các nỗ lực giảm thiểu liên quan đến các điểm nhập (entry points) khác nhau.
- **Trong quá trình phát triển:** Việc kiểm thử mã nguồn để phát hiện các lỗi hiển nhiên ở từng bước giúp xác định xem các mục tiêu thiết kế ban đầu có được đáp ứng hay không.
- **Cập nhật liên tục:** Bề mặt tấn công cần được **tài liệu hóa và cập nhật liên tục** trong suốt vòng đời phát triển phần mềm (SDL) để cung cấp phản hồi cho đội ngũ phát triển,.

3. Các yếu tố kiểm thử then chốt

Việc kiểm thử bề mặt tấn công bao gồm các hoạt động thiết yếu sau:

- Xác định mức độ mã nguồn mà người dùng **không đáng tin cậy** có thể tiếp cận được.
- Kiểm tra số lượng mã nguồn chạy với **quyền ưu tiên cao (elevated privilege)**.
- Xác minh việc triển khai các kế hoạch giảm thiểu rủi ro đã được chi tiết hóa trong mô hình đe dọa (threat model).

4. Công cụ Microsoft Attack Surface Analyzer

Tài liệu nhấn mạnh một công cụ quan trọng cho môi trường Windows:

- **Chức năng:** Đo lường tác động bảo mật của một ứng dụng đối với hệ điều hành Windows bằng cách quét và phát hiện các thay đổi hệ thống sau khi cài đặt ứng dụng.
- **Lợi ích:**
    - Xem các thay đổi cụ thể trên bề mặt tấn công của Windows.
    - Đánh giá sự thay đổi tổng thể về bề mặt tấn công trong môi trường doanh nghiệp.
    - Đánh giá rủi ro cho nền tảng nơi ứng dụng dự kiến sẽ tồn tại.
    - Cung cấp thông tin chi tiết cho đội ngũ ứng phó sự cố.
- **Ưu điểm:** Công cụ này hoạt động **độc lập** với ứng dụng đang được kiểm thử, giúp nó trở thành một lựa chọn lý tưởng cho bước kiểm thử bảo mật cuối cùng trong quy trình SDL.

Penetration Testing

Dưới đây là nội dung chi tiết của phần **Penetration Testing** (Kiểm thử xâm nhập) trong Chương 11:

1. Khái niệm Penetration Testing

- **Định nghĩa:** Kiểm thử xâm nhập (Pen-testing) là một hình thức kiểm tra **chủ động** (active form) để tìm kiếm các điểm yếu và lỗ hổng trong hệ thống.
- **Sự khác biệt với Scanning:** Trong khi việc quét lỗ hổng (scanning) mang tính thụ động, pen-testing tận dụng trí tuệ của con người để thực hiện các cuộc kiểm tra có mục tiêu cụ thể hơn.
- **Mục tiêu:** Mô phỏng tư duy và phương pháp của kẻ tấn công (attacker’s ethos) nhằm phát hiện các vấn đề bảo mật trước khi bị kẻ xấu lợi dụng.

2. Vai trò trong Vòng đời Phát triển Phần mềm (SDL)

- **Đánh giá hiệu quả:** Pen-testing giúp phân tích chương trình và xác định xem các biện pháp giảm thiểu (mitigations) đã lập kế hoạch có thực sự hiệu quả hay không.
- **Phát hiện lỗ hổng ẩn:** Nó có thể tìm ra những lỗ hổng mà đội ngũ phát triển chưa từng nghĩ tới.
- **Các mô hình thực hiện:** Có thể triển khai theo phương pháp hộp trắng (white-box), hộp đen (black-box) hoặc hộp xám (gray-box).

3. Quy trình thực hiện (4 Bước chính)

Quy trình này mang tính cấu trúc chặt chẽ và có hệ thống:

1. **Reconnaissance (Trinh sát):** Giai đoạn khám phá và liệt kê các thành phần của hệ thống.
2. **Attack and exploitation (Tấn công và khai thác):** Thực hiện tấn công dựa trên các thông tin đã thu thập.
3. **Removal of evidence (Xóa dấu vết/Bằng chứng):** Xóa bỏ các dấu hiệu để mô phỏng khả năng ẩn mình của kẻ tấn công.
4. **Reporting (Báo cáo):** Tổng hợp kết quả và đưa ra khuyến nghị.

4. Đặc điểm của quá trình kiểm thử

- **Thiết lập mục tiêu:** Quá trình bắt đầu với các mục tiêu cụ thể như kiểm tra lỗ hổng xác thực đầu vào (input validation), lỗi cấu hình hoặc lỗ hổng trên nền tảng máy chủ khi triển khai.
- **Tính lặp lại và ghi chép:** Các kiểm thử viên sẽ thăm dò phần mềm, ghi chép lại các phản hồi và lỗi để định hình cho các bài kiểm tra tiếp theo.
- **Khả năng tái hiện (Reproducible):** Hồ sơ kiểm thử phải chứng minh được một tình huống có thể tái hiện lại lỗi để đội ngũ phát triển hiểu rõ nguyên nhân gốc rễ và tiến hành sửa chữa.

- **Tính chất:** Đây là một quy trình **chậm và có phương pháp**, mọi bước thực hiện và kết quả đều phải được xác thực.

Common Methods

Dưới đây là nội dung chi tiết của phần **Common Methods** (Các phương pháp kiểm thử phổ biến) trong Chương 11. Các phương pháp này được sử dụng để đánh giá mức độ phù hợp và an toàn của phần mềm qua nhiều kịch bản khác nhau.

1. Fuzzing (Kiểm thử mờ)

- **Cơ chế:** Là phương pháp dùng lực lượng thô (brute-force) áp dụng một lượng lớn các đầu vào (inputs) để xác định cái nào gây ra lỗi hoặc lỗ hổng.
- **Phạm vi:** Có thể áp dụng cho các giao thức mạng, tệp tin và web. Đây là phương pháp chính để tìm lỗi trình duyệt và các vấn đề về xác thực đầu vào như XSS hoặc SQL injection.
- **Phân loại:**
    - **Smart (Thông minh) vs. Dumb (Ngớ ngẩn):** Dựa trên mức độ logic được sử dụng để tạo dữ liệu đầu vào.
    - **Generation-based (Dựa trên tạo mới):** Dựa trên đặc tả giao thức đầu vào để tạo dữ liệu.
    - **Mutation-based (Dựa trên đột biến):** Lấy dữ liệu mẫu đúng và làm biến đổi chúng theo các cách cụ thể.

2. Scanning (Quét)

- **Định nghĩa:** Là quá trình liệt kê tự động các đặc tính cụ thể của ứng dụng hoặc mạng.
- **Ứng dụng:** Bao gồm quét cổng mạng, nhận diện hệ điều hành (OS fingerprinting) và quét lỗ hổng ứng dụng.

- **Lợi ích:** Giúp đội ngũ phát triển hiểu rõ "dấu chân" (footprint) của ứng dụng trên nền tảng đích trước khi phát hành, thường được yêu cầu bởi các tiêu chuẩn như PCI DSS.

3. Simulations (Mô phỏng)

- **Môi trường:** Kiểm thử ứng dụng trong môi trường phản chiếu chính xác môi trường sản xuất (production).
- **Mục tiêu:** Kiểm tra cấu hình, các vấn đề về dữ liệu, quá trình khởi động và tính tương tác với hệ điều hành.
- **Load testing:** Một phần của mô phỏng nhằm đảm bảo hệ thống chịu được tải trọng thực tế mà không làm hỏng các cơ chế kiểm soát bảo mật.

4. Failure Modes (Các chế độ lỗi)

- **Mục tiêu:** Xác định các lỗi ngay cả khi chúng chưa gây ra thất bại ngay lập tức (ví dụ: mã nguồn "chết" không được thực thi).
- **Fault testing (Break test):** Cố tình sử dụng các đầu vào được thiết kế để gây lỗi nhằm kiểm tra khả năng xử lý của phần mềm.
- **Stress testing:** Sử dụng tải trọng cực lớn để đảm bảo phần mềm vẫn hoạt động tin cậy.

5. Cryptographic Validation (Xác thực mật mã)

- **Nguyên tắc:** Sử dụng các thuật toán đã được phê duyệt và triển khai chúng một cách chính xác.
- **Vấn đề then chốt:** Bảo vệ khóa (keys), giá trị mầm (seeds) và đảm bảo tạo số ngẫu nhiên thực sự (không phải pseudo-random có tính lặp lại).
- **Tiêu chuẩn:** Thường tuân theo **FIPS 140-2** đối với các hệ thống của chính phủ Mỹ.

6. Regression Testing (Kiểm thử hồi quy)

- **Mục đích:** Đảm bảo rằng các thay đổi (vá lỗi) không làm hỏng các chức năng đang hoạt động hoặc các phiên bản cũ hơn của mã nguồn.

- **Thách thức:** Rất tốn kém và mất thời gian, đặc biệt khi phải hỗ trợ nhiều phiên bản phần mềm khác nhau trên thị trường.

7. Integration Testing (Kiểm thử tích hợp)

- **Định nghĩa:** Kiểm tra xem các thành phần riêng lẻ khi kết hợp lại có hoạt động đúng như một hệ thống thống nhất hay không.
- **Trọng tâm:** Kiểm tra sự di chuyển dữ liệu qua các ranh giới tin cậy (trust boundaries) và đảm bảo tính tương hợp giữa các mô-đun.

8. Continuous Testing (Kiểm thử liên tục)

- **Cơ chế:** Sử dụng kiểm thử tự động như một phần của quy trình phân phối phần mềm (CI/CD).
- **Lợi ích:** Thu thập dữ liệu về rủi ro kinh doanh một cách nhanh chóng ngay trong quá trình phát triển thay vì đợi đến cuối quy trình.

chapter 12 Security Testing Strategy and Plan

Develop a Security Testing Strategy and a Plan

Nội dung phần **Develop a Security Testing Strategy and a Plan** (Phát triển Chiến lược và Kế hoạch Kiểm thử Bảo mật) trong Chương 12 tập trung vào việc thiết lập một cách tiếp cận có hệ thống để kiểm tra tính an toàn của phần mềm. Dưới đây là các chi tiết cụ thể:

1. Kế hoạch và Chiến lược Kiểm thử

- **Bước đầu tiên:** Xây dựng một **kế hoạch kiểm thử (test plan)**, đây là tài liệu chi tiết về cách tiếp cận hệ thống để kiểm tra phần mềm.
- **Chiến lược kiểm thử (test strategy):** Bắt đầu bằng một bản phác thảo mô tả phương pháp kiểm thử tổng thể.
- **Mục tiêu của kế hoạch:** Phải nêu rõ phạm vi, cách tiếp cận, nguồn lực, và lịch trình của hoạt động kiểm thử. Nó cũng định nghĩa các **tiêu chí hoàn thành kiểm thử** và các quy định về xác định rủi ro cũng như phân bổ nguồn lực.

2. Xây dựng các trường hợp kiểm thử (Test Cases)

- **Câu hỏi cốt lõi:** Mỗi trường hợp kiểm thử được thiết kế để trả lời: "Tôi sẽ kiểm tra cái gì, và kết quả 'đúng' sẽ trông như thế nào?".
- **Thông tin trong tài liệu Test Case:** Bao gồm mã định danh kiểm thử duy nhất, liên kết đến các yêu cầu từ đặc tả thiết kế, ghi chú về điều kiện tiên quyết, tham chiếu đến các khung thử nghiệm (test harnesses), kịch bản (scripts) và các ghi chú bổ sung.
- **Dữ liệu đầu vào và kết quả:** Mỗi trường hợp kiểm thử phải quy định rõ giá trị đầu vào thực tế và kết quả mong đợi.

3. Các loại hình kiểm thử đan xen

Tài liệu lưu ý rằng việc kiểm thử thường có sự chồng chéo giữa ba loại:

- **Kiểm thử chức năng (Functional testing):** Đảm bảo đơn vị mã thực hiện đúng chức năng và xử lý lỗi phù hợp.
- **Kiểm thử bảo mật (Security testing):** Đảm bảo phần mềm không có lỗi hoặc lỗ hổng làm tăng rủi ro.
- **Kiểm thử chất lượng/chấp nhận (Qualification/Acceptance testing):** Đảm bảo sản phẩm hoàn thiện đã sẵn sàng để sử dụng.

4. Các yếu tố của Kiểm thử chất lượng (Qualification Testing)

Thiết kế và thực thi kiểm thử này thường được quy định trong hợp đồng, bao gồm:

- Các tính năng bắt buộc phải kiểm thử.
- Giới hạn tải (load limits) và các loại kiểm thử áp lực (stress tests).
- Tất cả các bài kiểm tra giảm thiểu rủi ro và bảo mật cần thiết.
- Mức độ hiệu suất và các giao diện cần kiểm tra.

5. Các câu hỏi thực tế mà Kế hoạch Kiểm thử phải trả lời

Một kế hoạch kiểm thử hoàn chỉnh cần xác định rõ trách nhiệm và quy trình:

- Ai chịu trách nhiệm thiết kế, thực thi các bài kiểm tra và duy trì môi trường thử nghiệm?.
- Ai chịu trách nhiệm quản lý cấu hình?.
- **Tiêu chí để dừng hoặc bắt đầu lại** nỗ lực kiểm thử là gì?.
- Khi nào mã nguồn sẽ được đưa vào kiểm soát thay đổi (change control)?.
- Báo cáo bất thường (anomaly reports) sẽ được viết ở cấp độ nào?.

Mục tiêu cuối cùng là sử dụng số lượng trường hợp kiểm thử ít nhất nhưng vẫn đạt được sự hiểu biết đầy đủ về chất lượng và bảo mật của sản phẩm.

Functional Security Testing

Nội dung phần **Functional Security Testing** (Kiểm thử Bảo mật Chức năng) trong Chương 12 tập trung vào việc xác minh xem phần mềm có thực hiện đúng các chức năng bảo mật như đã thiết kế hay không. Dưới đây là các chi tiết cụ thể từ nguồn tài liệu:

1. Mục tiêu của Kiểm thử Chức năng

- **Đánh giá sự phù hợp:** Kiểm thử chức năng được thực hiện để đánh giá mức độ mà phần mềm đáp ứng các kỳ vọng của người dùng cuối về mặt vận hành.
- **Các lĩnh vực tuân thủ:** Nó xác định xem phần mềm có tuân thủ các yêu cầu về **độ tin cậy (reliability)**, **logic**, **hiệu suất (performance)** và **khả năng mở rộng (scalability)** hay không.
- **Phân biệt khái niệm:**
    - **Reliability (Độ tin cậy):** Đo lường việc phần mềm hoạt động đúng chức năng vào mọi thời điểm theo mong đợi của khách hàng.
    - **Resiliency (Tính đàn hồi/Khả năng phục hồi):** Đo lường khả năng phần mềm vẫn hoạt động mạnh mẽ ngay cả khi đang bị kẻ tấn công nhắm mục tiêu.

2. Quy trình 5 bước thực hiện Kiểm thử Chức năng

Việc kiểm thử chức năng cần được thực hiện theo một trình tự chặt chẽ:

1. **Xác định các chức năng (yêu cầu):** Liệt kê những gì phần mềm được mong đợi phải thực hiện dựa trên tài liệu yêu cầu.
2. **Tạo dữ liệu kiểm thử đầu vào:** Xây dựng các tập dữ liệu dựa trên đặc tả kỹ thuật của từng chức năng.
3. **Xác định kết quả đầu ra mong đợi:** Dự đoán kết quả "đúng" mà phần mềm phải trả về dựa trên đặc tả chức năng.
4. **Thực thi các trường hợp kiểm thử (test cases):** Chạy các bài kiểm tra tương ứng với từng yêu cầu chức năng.
5. **So sánh kết quả:** Đối chiếu đầu ra thực tế với đầu ra mong đợi để xác định mức độ tuân thủ.

3. Kiểm thử Đơn vị (Unit Testing)

Đây là cấp độ kiểm thử đầu tiên và rất quan trọng trong bảo mật chức năng.

- **Người thực hiện:** Do chính các lập trình viên thực hiện trong quá trình viết mã.
- **Vai trò:** Đảm bảo các thành phần logic là chính xác và đơn vị mã đó đáp ứng các yêu cầu đã công bố.
- **Lợi ích:** Giúp phát hiện lỗi cực sớm, trước khi mã nguồn rời khỏi giai đoạn phát triển, đồng thời đảm bảo mức độ kiểm soát và giảm thiểu lỗ hổng hợp lý ở cấp độ thấp nhất,.

Nonfunctional Security Testing

Dưới đây là nội dung chi tiết của phần **Nonfunctional Security Testing** (Kiểm thử Bảo mật Phi chức năng) trong Chương 12:

1. Bản chất của Yêu cầu Phi chức năng

- **Tầm quan trọng:** Việc kiểm thử các yêu cầu như **độ tin cậy (reliability)**, **hiệu suất (performance)** và **khả năng mở rộng (scalability)** cũng quan trọng tương đương với các yêu cầu chức năng.

- **Đặc điểm:** Các yêu cầu này thường liên quan đến sự thuận tiện khi sử dụng và các yếu tố hệ thống khác. Nếu thiếu chúng, giá trị sử dụng của phần mềm sẽ bị giảm đáng kể.
- **Yêu cầu đo lường:** Giống như yêu cầu chức năng, các yêu cầu phi chức năng cần được quy định bằng các thuật ngữ có thể đo lường được để đội ngũ phát triển có thể giải quyết.

2. Thỏa thuận mức dịch vụ (SLA)

- **Định nghĩa:** Các yêu cầu phi chức năng thường được thể hiện thông qua một **Thỏa thuận mức dịch vụ (Service Level Agreement - SLA)**.
- **Mục tiêu kiểm thử hiệu suất:** Mục tiêu chính không phải là tìm các lỗi (bugs) cụ thể, mà là xác định các **nút thắt cổ chai (bottlenecks)** và các yếu tố hiệu suất của hệ thống đang được kiểm thử.

3. Kiểm thử Tải (Load Testing) và Kiểm thử Áp lực (Stress Testing)

Đây là hai phương pháp phổ biến để kiểm tra hiệu suất:

- **Load Testing:** Thực hiện chạy hệ thống dưới một môi trường tốc độ được kiểm soát để đo lường khả năng xử lý.
- **Stress Testing:** Đưa hệ thống vượt quá điểm vận hành bình thường để xem cách nó phản ứng với các **điều kiện quá tải**.

4. Khả năng phục hồi (Recoverability)

- **Định nghĩa:** Đây là khả năng của một ứng dụng trong việc **tự khôi phục** về mức độ chức năng mong đợi sau khi các biện pháp bảo vệ an ninh bị vi phạm hoặc bị vượt qua.

Testing Techniques

Dưới đây là nội dung chi tiết của phần **Testing Techniques** (Các kỹ thuật kiểm thử) trong Chương 12. Phần này tập trung vào lượng thông tin mà kiểm thử viên có trước khi thực hiện kiểm thử.

1. Phân loại kỹ thuật kiểm thử

Tài liệu chia các kỹ thuật kiểm thử thành ba loại chính dựa trên mức độ hiểu biết về hệ thống:

- **White-Box Testing (Kiểm thử hộp trắng):** Được thực hiện khi kiểm thử viên có **toàn bộ kiến thức** về các thành phần hoạt động, bao gồm mã nguồn và cách vận hành của nó. Kỹ thuật này thường được thực hiện sớm trong chu kỳ phát triển (ví dụ: Kiểm thử đơn vị) và tập trung vào **cấu trúc** của phần mềm.
- **Black-Box Testing (Kiểm thử hộp đen):** Kiểm thử viên **không có kiến thức** về hoạt động bên trong của phần mềm. Họ đóng vai trò như một người dùng cuối, chỉ biết các đầu vào yêu cầu và tập trung vào **đặc tính hành vi** của ứng dụng.
- **Gray-Box Testing (Kiểm thử hộp xám):** Là sự kết hợp, nơi kiểm thử viên có nhiều thông tin hơn hộp đen nhưng ít hơn hộp trắng (không có toàn quyền truy cập mã nguồn).

2. So sánh Kiểm thử Hộp trắng và Hộp đen

Đặc điểm	**White-Box Testing**	**Black-Box Testing**
Kiến thức	Đầy đủ (bao gồm mã nguồn)	Không có kiến thức
Trọng tâm	Đánh giá cấu trúc và thiết kế	Đánh giá hành vi phần mềm
Tỷ lệ lỗi giả (False Positives)	Thấp	Cao
Lỗi logic	Có thể phát hiện được	Thường không thấy được

3. Môi trường kiểm thử (Testing Environment)

Việc kiểm thử không chỉ giới hạn trong mã nguồn mà còn phải kiểm tra cả môi trường vận hành:

- **Ranh giới tin cậy (Trust Boundaries):** Rất quan trọng để kiểm tra sự di chuyển dữ liệu xuyên suốt ứng dụng, đặc biệt là tại các điểm kết nối giữa các mô-đun.
- **Test Harness (Khung thử nghiệm):** Là phương tiện để tài liệu hóa phần mềm, công cụ, dữ liệu mẫu và cấu hình được sử dụng để hoàn thành một bộ kiểm thử.
- **Test Scripts (Kịch bản kiểm thử):** Các bước được mã hóa để mô phỏng hành động người dùng, giúp loại bỏ lỗi thủ công và tự động hóa việc so sánh kết quả thực tế với kết quả mong đợi.
- **Test Suites (Bộ kiểm thử):** Các bài kiểm tra thường được nhóm lại theo chức năng (ví dụ: bộ kiểm thử bảo mật, bộ kiểm thử cơ sở dữ liệu) để dễ quản lý và tái sử dụng.

Environment

Phần **Environment** (Môi trường kiểm thử) trong Chương 12 nhấn mạnh tầm quan trọng của việc thiết lập một không gian làm việc chuẩn hóa để đảm bảo tính chính xác và hiệu quả của quy trình kiểm thử. Dưới đây là các nội dung chi tiết:

1. Vai trò của Môi trường Kiểm thử

- **Tích hợp và Thu thập dữ liệu:** Một môi trường kiểm thử được thiết lập tốt giúp tích hợp quy trình kiểm thử vào quy trình phát triển phần mềm và thu thập dữ liệu cần thiết để xác nhận việc hoàn thành kiểm thử.
- **Tính tương hợp:** Nó cho phép khả năng tương tác giữa các tạo tác (artifacts) và các số liệu đo lường (metrics) giữa các hệ thống khác nhau để cải thiện tính hữu dụng của kết quả.

2. Kiểm thử Kết nối và Ranh giới Tin cậy

- **Kiểm tra đầu-cuối:** Việc kiểm tra sự di chuyển dữ liệu qua các ranh giới tin cậy từ đầu đến cuối ứng dụng là cực kỳ quan trọng.

- **Lỗi giao tiếp giữa các mô-đun:** Không chỉ kiểm tra lỗi trong từng mô-đun riêng lẻ, mà còn phải kiểm tra lỗi tại các điểm kết nối giữa chúng. Một ví dụ điển hình là lỗi lệch đơn vị đo lường (ví dụ: foot-pounds và newton-meters) giữa các hệ thống con, dẫn đến thất bại toàn bộ hệ thống ngay cả khi từng phần hoạt động đúng dải thông số của nó.

3. Các công cụ hỗ trợ môi trường

- **Khung thử nghiệm (Test Harness):** Là phương tiện để **tài liệu hóa phần mềm, công cụ, dữ liệu mẫu đầu vào/đầu ra và các cấu hình** được sử dụng để hoàn thành một bộ kiểm thử.
- **Kịch bản kiểm thử (Test Scripts):** Các bước kiểm thử được mã hóa để mô phỏng hành động người dùng. Việc tự động hóa giúp **loại bỏ lỗi thủ công**, đồng thời cải thiện tốc độ và độ chính xác của việc diễn giải kết quả.
- **Bộ kiểm thử (Test Suites):** Các bài kiểm tra thường được nhóm lại theo chức năng (như bảo mật, cơ sở dữ liệu, kiểm tra ranh giới) để **dễ quản lý và thúc đẩy việc tái sử dụng** thay vì phải phát triển lại liên tục.

Standards

Phần **Standards** (Các tiêu chuẩn) trong Chương 12 giới thiệu các bộ tiêu chuẩn quốc tế và phương pháp luận giúp chuẩn hóa quy trình kiểm thử và đánh giá chất lượng an ninh phần mềm. Dưới đây là nội dung chi tiết:

1. ISO/IEC 25010:2011

- **Tên đầy đủ:** Hệ thống và kỹ thuật phần mềm – Yêu cầu chất lượng hệ thống và phần mềm và Đánh giá (SQuaRE) – Các mô hình chất lượng hệ thống và phần mềm.
- **Vai trò:** Cung cấp hướng dẫn để thiết lập chất lượng trong các sản phẩm phần mềm.
- **Trọng tâm:** Xây dựng mô hình chất lượng xoay quanh các đặc tính: **tính chức năng (functionality), độ tin cậy (reliability), và khả năng sử dụng (usability)**. Ngoài ra còn bao gồm tính hiệu quả, khả năng bảo trì và khả năng di động.

- **Khía cạnh con người:** Tiêu chuẩn này giải quyết cả khía cạnh con người trong chất lượng, nơi các yêu cầu có thể thay đổi hoặc không rõ ràng, giúp đội ngũ phát triển có cùng cách hiểu về mục tiêu dự án.

2. SSE-CMM (ISO/IEC 21827:2008)

- **Tên đầy đủ:** Mô hình trưởng thành năng lực kỹ thuật bảo mật hệ thống (Systems Security Engineering Capability Maturity Model).
- **Phạm vi:** Áp dụng cho toàn bộ vòng đời của sản phẩm tin cậy hoặc hệ thống an toàn, từ xác định khái niệm, thiết kế, phát triển cho đến vận hành, bảo trì và hủy bỏ.
- **Cấu trúc:** Bao gồm **11 quy trình** định nghĩa những gì cần đạt được trong kỹ thuật bảo mật và các **mức độ trưởng thành (maturity levels)** để đo lường mức độ hoàn thành các mục tiêu đó.
- **Giá trị:** Được sử dụng làm công cụ để đánh giá và cải thiện các thực hành kỹ thuật bảo mật trong một tổ chức.

3. OSSTMM (Open Source Security Testing Methodology Manual)

- **Đặc điểm:** Là một hệ thống được bình duyệt (peer-reviewed) cung cấp **phương pháp luận khoa học** để đánh giá an ninh vận hành dựa trên các số liệu phân tích.
- **Mục tiêu:** Tạo ra một hệ thống có thể đặc tả chính xác an ninh của một hệ thống vận hành một cách nhất quán và đáng tin cậy.
- **5 phần kiểm thử/kiểm toán chính:**
    1. Mạng dữ liệu (Data networks).
    2. Viễn thông (Telecommunications).
    3. Không dây (Wireless).
    4. Vật lý (Physical).
    5. Con người (Human).

Crowd Sourcing

Phần **Crowd Sourcing** (Tận dụng nguồn lực cộng đồng) trong Chương 12 đề cập đến việc huy động các nguồn lực kiểm thử từ bên ngoài quy trình phát triển chính thống để phát hiện các lỗ hổng bảo mật. Dưới đây là các nội dung chi tiết:

1. Các nhóm tham gia bên ngoài

- Kiểm thử từ cộng đồng có thể đến từ khách hàng, các bên thứ ba, hacker và các nhà nghiên cứu bảo mật.
- **Hacker:** Thường sử dụng các phương pháp như **fuzzing** để tìm lỗ hổng và sau đó khai thác hoặc bán thông tin đó cho bên khác mà không thông báo cho đội ngũ phát triển.
- **Nhà nghiên cứu bảo mật (Security Researchers):** Cũng thực hiện các quy trình tìm lỗi tương tự nhưng sẽ thông báo cho công ty về những phát hiện của họ.

2. Tiết lộ có trách nhiệm (Responsible Disclosure)

- Đây là quy tắc mà các nhà nghiên cứu bảo mật thường tuân theo: Họ giữ bí mật về lỗ hổng trong một khoảng thời gian nhất định để giúp doanh nghiệp có đủ thời gian tạo ra và phát hành bản vá bảo mật trước khi thông tin được công khai.

3. Chương trình Tiền thưởng lỗi (Bug Bounty Programs)

- Các công ty tiến bộ thường thiết lập các kênh chính thức để làm việc với cộng đồng nghiên cứu bảo mật thông qua chương trình **Bug Bounty**.
- **Cơ chế:** Công ty trả các khoản tiền thưởng (bounties) hoặc phần thưởng cho nhà nghiên cứu để đổi lấy việc họ tìm ra và báo cáo lỗi.
- **Lợi ích:** Phương pháp này tạo ra một hệ sinh thái có lợi cho tất cả các bên, giúp doanh nghiệp phát hiện lỗ hổng sớm hơn thông qua sức mạnh trí tuệ của cộng đồng.

chapter 13 Software Testing and Acceptance

Perform Verification and Validation Testing

Phần **Perform Verification and Validation Testing** (Thực hiện Kiểm thử Xác minh và Xác nhận) thuộc Chương 13 tập trung vào việc đảm bảo phần mềm được xây dựng đúng cách và đáp ứng đúng nhu cầu của người dùng. Dưới đây là nội dung chi tiết:

1. Khái niệm Xác minh (Verification) và Xác nhận (Validation)

Đây là hai chức năng riêng biệt nhưng bổ trợ cho nhau để xác định hệ thống có thỏa mãn các yêu cầu hay không:

- **Verification (Xác minh):** Kiểm tra xem các yêu cầu có được đáp ứng hay không. Câu hỏi cốt lõi là: **"Chúng ta có đang xây dựng sản phẩm đúng cách không?"** (Are we building the product right?).
- **Validation (Xác nhận):** Kiểm tra xem các yêu cầu đó có chính xác và đầy đủ hay không. Câu hỏi cốt lõi là: **"Chúng ta có đang xây dựng đúng sản phẩm (mà người dùng cần) không?"** (Are we building the right product?).

2. Kế hoạch V&V (SVVP)

Tất cả các hoạt động V&V phải được thiết lập và tài liệu hóa trong **Kế hoạch Xác minh và Xác nhận Phần mềm (Software Validation and Verification Plan - SVVP)**. Kế hoạch này bao gồm:

- Các yêu cầu quản trị về giải quyết và báo cáo bất thường (anomaly reports).
- Chính sách về ngoại lệ và sai lệch (deviation).
- Các thủ tục kiểm soát cấu hình và đường cơ sở (baseline).
- Các tiêu chuẩn và quy ước được áp dụng để hướng dẫn thực hiện.

3. Hai hình thức V&V chính

Tài liệu chia V&V thành hai loại dựa trên mục đích đánh giá:

- **Management V&V (V&V cấp quản lý):** Xem xét các kế hoạch quản lý, lịch trình, yêu cầu và phương pháp để đánh giá mức độ phù hợp của chúng với dự án. Hoạt động này giúp đưa ra quyết định về hành động khắc phục, phân bổ nguồn lực và phạm vi dự án.
- **Technical V&V (V&V cấp kỹ thuật):** Đánh giá trực tiếp sản phẩm phần mềm, bao gồm tài liệu yêu cầu, thiết kế, mã nguồn, tài liệu hướng dẫn sử dụng và các thủ tục cài đặt. Mục tiêu là xác nhận sản phẩm tuân thủ các thông số kỹ thuật, quy định và tiêu chuẩn.

4. Kiểm thử Độc lập (Independent Testing - IV&V)

Để đảm bảo sự tin tưởng tối đa vào tính toàn vẹn của sản phẩm, các bên thứ ba không liên quan đến quá trình phát triển có thể thực hiện kiểm thử, gọi là **Xác minh và Xác nhận Độc lập (IV&V)**.

- **Yêu cầu cốt lõi:** Tính độc lập. Đội ngũ kiểm thử không được nằm dưới quyền quản lý của bộ phận sản xuất sản phẩm.
- **Kiểm toán (Audits):** Đây là cơ chế phổ biến nhất cho đánh giá chấp nhận độc lập. Kiểm toán viên (thường là bên thứ ba) sẽ xác nhận sự tuân thủ của sản phẩm với các kế hoạch, quy định hoặc tiêu chuẩn thông qua bằng chứng khách quan.

Software Qualification Testing

Nội dung phần **Software Qualification Testing** (Kiểm thử Chất lượng Phần mềm) trong Chương 13 tập trung vào quy trình đánh giá chính thức để xác định xem sản phẩm phần mềm đã sẵn sàng để bàn giao cho khách hàng hay chưa. Dưới đây là các chi tiết cụ thể:

1. Khái niệm và Mục tiêu

- **Định nghĩa:** Đây là quá trình **phân tích chính thức** được thực hiện để xác định xem hệ thống hoặc sản phẩm phần mềm có thỏa mãn các **tiêu chí chấp nhận (acceptance criteria)** hay không.
- **Vai trò của khách hàng:** Về mặt thực tế, khách hàng là bên thực hiện kiểm thử chất lượng để quyết định có chấp nhận sản phẩm hay không.

- **Mục tiêu cốt lõi:** Đảm bảo rằng các yêu cầu của khách hàng đã được đáp ứng và tất cả các thành phần được tích hợp chính xác vào sản phẩm đã mua.
- **Bằng chứng tuân thủ:** Quá trình này cung cấp bằng chứng cho thấy sản phẩm tuân thủ các mức độ thiết kế, hiệu suất và sự đảm bảo (assurance) đã được quy định trong hợp đồng.

2. Phạm vi và Điều kiện Kiểm thử

- **Đánh giá sự chính xác:** Quy trình này được tinh chỉnh để đánh giá cụ thể liệu việc thiết kế và phát triển phần mềm có chính xác hay không, tìm kiếm các khiếm khuyết có thể gây ra lỗi hoặc có thể bị khai thác trong thực tế.
- **Tuân thủ đa phương diện:** Các bài kiểm tra không chỉ đánh giá sự tuân thủ các yêu cầu ban đầu mà còn cả các quy định về **hợp đồng, tiêu chuẩn và pháp lý**.
- **Môi trường thử nghiệm:** Phần mềm được đánh giá chi tiết dưới cả **điều kiện bình thường và bất thường**.
- **Vai trò của Kiểm toán (Audits):** Đảm bảo rằng tất cả các tài liệu cần thiết đã được hoàn thiện đúng cách và sản phẩm đáp ứng mọi yêu cầu về chức năng và hiệu suất trong hợp đồng.

Qualification Testing Hierarchy

Việc kiểm thử không diễn ra đơn lẻ mà đi từ cấp độ thấp đến cao:

- **Bắt đầu từ cấp độ thành phần (component level)** trong giai đoạn phát triển và tiến dần lên hệ thống tích hợp hoàn chỉnh trong giai đoạn chấp nhận.
- **Đối tượng mục tiêu:** Bao gồm kiến trúc phần mềm, các thành phần, giao diện (interfaces) và dữ liệu.
- **Các hoạt động chính:**
    - Phân tích khả năng truy xuất nguồn gốc thiết kế phần mềm (kiểm tra tính chính xác).
    - Đánh giá thiết kế phần mềm và phân tích giao diện thiết kế.

    - Lập kế hoạch và thiết kế kiểm thử cho từng cấp độ.

- **Hỗ trợ đảm bảo (Assurance):** Được thực hiện bằng cách phân tích mức độ tuân thủ của mã nguồn với đặc tả thiết kế và tiêu chuẩn lập trình, kết hợp với các trường hợp kiểm thử (test cases) mục tiêu được tạo riêng cho đối tượng đang được phân tích.

Identifying Undocumented Functionality

Phần **Identifying Undocumented Functionality** (Xác định chức năng không được tài liệu hóa) trong Chương 13 tập trung vào một nguyên tắc cốt lõi của bảo mật phần mềm: phần mềm chỉ nên thực hiện những gì nó được thiết kế để làm và không làm gì khác.

Dưới đây là các chi tiết cụ thể từ nguồn tài liệu:

1. Định nghĩa và Tầm quan trọng

- **Phần mềm an toàn:** Được định nghĩa là phần mềm thực hiện đúng các chức năng đã định rõ trong yêu cầu và **chỉ** thực hiện những chức năng đó.
- **Chức năng không được tài liệu hóa:** Là tập hợp các tính năng hoặc hành động tồn tại trong mã nguồn nhưng không được quy định trong tài liệu yêu cầu hoặc thiết kế.
- **Rủi ro:** Những chức năng này thường là các "điểm nhập" (entry points) tiềm năng mà kẻ tấn công có thể lợi dụng để thực hiện các hành vi ngoài ý muốn của nhà phát triển.

2. Nguồn gốc của chức năng không được tài liệu hóa

- **Sơ suất trong thiết kế:** Nhiều hệ thống thất bại khi người dùng tìm thấy một hoạt động (activity) không được xem xét trong quá trình thiết kế và xây dựng nhưng vẫn có thể thực hiện được trong mã nguồn thực tế.
- **Mối quan hệ giữa Bảo mật và Chất lượng:** Nếu một phần mềm được coi là "an toàn" nhưng lại thiếu "chất lượng" (lỗi quy trình), các chức năng không được tài liệu hóa có thể xuất hiện, dẫn đến các hành vi không mong muốn hoặc không hợp lệ.

3. Biện pháp ngăn chặn

- **Hiểu rõ các điểm nhập:** Đội ngũ phát triển phải xác định và hiểu rõ tất cả các điểm mà dữ liệu có thể đi vào hệ thống.
- **Tài liệu hóa đầy đủ:** Mọi hoạt động khả thi trên từng đối tượng (object) trong hệ thống cần được định nghĩa và ghi chép lại rõ ràng.
- **Áp dụng biện pháp bảo vệ:** Sau khi đã tài liệu hóa, cần áp dụng các biện pháp kiểm soát (safeguards) phù hợp để ngăn chặn việc sử dụng trái phép các điểm nhập này.

Analyze Security Implications of Test Results

Nội dung phần **Analyze Security Implications of Test Results** (Phân tích hệ quả bảo mật của kết quả kiểm thử) trong Chương 13 tập trung vào việc đánh giá mức độ nghiêm trọng của các lỗi tìm thấy và đưa ra quyết định xử lý phù hợp. Dưới đây là các chi tiết cụ thể:

1. Chấm điểm và Ưu tiên lỗi

- **Dựa trên tác động:** Các lỗi (bugs) tìm thấy trong quá trình phát triển được chấm điểm dựa trên **tác động (impact)**. Vì không thể sửa chữa tất cả các lỗi, việc ưu tiên sửa các lỗi gây ra hệ quả nghiêm trọng là yêu cầu bắt buộc.
- **Tiêu chí ưu tiên:** Tác động là yếu tố thúc đẩy chính để quyết định có sửa lỗi hay không, nhưng điều này không có nghĩa là bỏ qua các lỗi nhỏ, đặc biệt nếu việc khắc phục chúng tốn ít chi phí và thực hiện được ngay từ sớm.

2. Lỗi phá vỡ tiêu chuẩn đóng gói (Break the Build)

- **Quy trình Build định kỳ:** Nhiều công ty sử dụng quy trình xây dựng (build) thường xuyên, nơi các mô-đun được sửa đổi và phải được hợp nhất (merge) một cách sạch sẽ.
- **Xử lý bắt buộc:** Bất kỳ lỗi nào làm gián đoạn hoặc phá vỡ các tiêu chí của quy trình build này đều phải được giải quyết ngay lập tức để ngăn lỗi cũ bám rễ sâu vào các phiên bản mã nguồn mới.

3. Theo dõi và Quản lý lỗi

- **Hệ thống Bug Tracking:** Tất cả các khiếm khuyết (defects) đã biết nên được theo dõi trong hệ thống quản lý lỗi. Hệ thống này giúp phân loại lỗi nào cần sửa và lỗi nào có thể trì hoãn.
- **Giới hạn thực tế:** Trong thực tế, một số lỗi quá khó hoặc quá tốn kém để sửa chữa ngay lập tức (ví dụ: yêu cầu thiết kế lại toàn bộ kiến trúc). Nếu lỗi đó không gây rủi ro lớn, việc khắc phục có thể được hoãn lại cho đến đợt cập nhật hoặc thiết kế lại lớn tiếp theo.

4. Khái niệm Bug Bar (Thanh chắn lỗi)

- **Định nghĩa:** Đây là một chỉ số đo lường dựa trên tiêu chuẩn rủi ro, xác định mức chất lượng tối thiểu của mã nguồn.
- **Quy tắc phát hành:** Nếu một lỗi được chấm điểm nằm **tại hoặc trên mức Bug Bar** lỗi đó **bắt buộc phải được khắc phục hoặc giảm thiểu** trước khi phần mềm được phát hành. Các lỗi nằm dưới mức này có thể tiếp tục được theo dõi.

Classify and Track Security Errors

Phần **Classify and Track Security Errors** (Phân loại và Theo dõi Lỗi Bảo mật) trong Chương 13 tập trung vào việc quản lý các khiếm khuyết phần mềm một cách có hệ thống để cải thiện chất lượng và độ an toàn của sản phẩm. Dưới đây là các nội dung chi tiết:

1. Phân loại Khiếm khuyết (Defects)

Tài liệu phân chia các khiếm khuyết phần mềm thành 5 loại chính dựa trên nguồn gốc hoặc hệ quả của chúng:

- **Flaws (Thiếu sót):** Các lỗi phát sinh từ khâu **thiết kế**.
- **Bugs (Lỗi):** Các lỗi phát sinh trong quá trình **viết mã (coding)**.
- **Behavioral anomalies (Bất thường về hành vi):** Các vấn đề liên quan đến cách ứng dụng vận hành trong thực tế.
- **Errors and faults (Lỗi và sai sót):** Các vấn đề về kết quả đầu ra có nguồn gốc từ các yếu tố khác.

- **Vulnerabilities (Lỗ hổng):** Các điểm có thể bị thao túng để khiến hệ thống hoạt động không đúng mục đích.

2. Theo dõi Lỗi (Bug Tracking)

Việc theo dõi lỗi giúp xác định trạng thái và ưu tiên xử lý:

- **Quyết định kinh tế:** Một số lỗi mờ nhạt, khó khai thác và tốn kém để sửa chữa có thể được để lại cho đến đợt viết lại mã nguồn lớn tiếp theo nhằm tiết kiệm chi phí.
- **Cơ sở dữ liệu khiếm khuyết:** Cần lưu trữ các thông tin như: nơi xảy ra lỗi, bản build nào, ai phát triển, ai phát hiện, và cách phát hiện. Đây là nguồn học tập quan trọng để tránh lặp lại sai lầm trong tương lai.

3. Mô hình DREAD (Scoring Bugs)

DREAD là một hệ thống chấm điểm rủi ro giúp so sánh mức độ nghiêm trọng của các lỗi khác nhau. Công thức chung là **Risk = Impact (Tác động) × Probability (Xác suất)**. Các thành phần của **DREAD** bao gồm:

- **D**amage potential (Tiềm năng gây hại): Đánh giá dựa trên tính bảo mật, toàn vẹn và sẵn có (CIA).
- **R**eproducibility (Khả năng tái hiện): Lỗi có dễ lặp lại hoặc có thể viết thành kịch bản tấn công không?
- **E**xploitability (Khả năng khai thác): Việc thực hiện cuộc tấn công khó hay dễ?
- **A**ffected users (Người dùng bị ảnh hưởng): Phạm vi người dùng bị tác động lớn đến mức nào?
- **D**iscoverability (Khả năng phát hiện): Lỗi có dễ bị tìm thấy bởi kẻ tấn công không?

4. Hệ thống chấm điểm lỗ hổng phổ biến (CVSS)

CVSS (Common Vulnerability Scoring System) là tiêu chuẩn thực tế để đánh giá điểm rủi ro của lỗ hổng:

- **Base Metric Group (Nhóm số liệu cơ bản):** Xem xét các đặc tính nội tại của lỗ hổng như độ phức tạp của cuộc tấn công, đặc quyền yêu cầu và tương tác của người dùng.
- **Temporal Metric Group (Nhóm số liệu thời gian):** Đại diện cho cách rủi ro thay đổi theo thời gian.
- **Environmental Metric Group (Nhóm số liệu môi trường):** Các đặc tính thay đổi tùy theo hệ thống và các biện pháp phòng thủ cụ thể.

Secure Test Data

Dưới đây là nội dung chi tiết của phần **Secure Test Data** (Dữ liệu Kiểm thử An toàn) thuộc Chương 13:

1. Tầm quan trọng của Dữ liệu Kiểm thử

- **Kiểm thử toàn diện:** Việc kiểm thử phải diễn ra xuyên suốt quá trình phát triển, từ khâu xác định yêu cầu (sử dụng Use Cases và Misuse Cases) đến khâu hoàn thiện hệ thống.
- **Tính đại diện:** Để kết quả kiểm thử phản ánh đúng thực tế, dữ liệu kiểm thử cần phải có tính **đại diện** cho dữ liệu thực tế mà hệ thống sẽ xử lý.
- **Dữ liệu cho Misuse Cases:** Ngoài dữ liệu "đúng", cần tạo ra các tập dữ liệu được thiết kế riêng để gây lỗi hệ thống nhằm kiểm tra khả năng chống chịu.

2. Thách thức trong việc tạo dữ liệu

- Việc tạo dữ liệu kiểm thử chất lượng rất khó khăn vì cần đảm bảo:
    - **Referential integrity (Tính toàn vẹn tham chiếu):** Các mối quan hệ giữa các bảng dữ liệu không bị phá vỡ.
    - **Statistical qualities (Đặc tính thống kê):** Dữ liệu giả nhưng phải có phân phối và tính chất giống dữ liệu thật.

3. Tái sử dụng dữ liệu sản xuất (Production Data)

- **Lý do:** Trong các môi trường phức tạp, việc tự tạo dữ liệu rất khó, nên các đội ngũ thường lấy dữ liệu từ môi trường thực tế (production) để kiểm thử.
- **Rủi ro bảo mật:** Việc sử dụng trực tiếp dữ liệu khách hàng trong môi trường kiểm thử đe dọa nghiêm trọng đến quyền riêng tư,.
- **Quy trình làm sạch dữ liệu (Cleansing/Anonymization):** Đây là bước bắt buộc và không hề đơn giản. Nó bao gồm:
    - Xác định các trường thông tin nhạy cảm cần bảo vệ.
    - **Anonymizing (Ẩn danh hóa):** Xóa bỏ các mối liên hệ giữa dữ liệu và danh tính thực.
    - **Tokenizing (Mã hóa thẻ):** Thay thế một số khóa dữ liệu để giữ tính toàn vẹn nhưng vẫn bẻ gãy các mối quan hệ định danh.
    - Xem xét tác động của việc **cộng dồn dữ liệu (data aggregation)**: Đôi khi các mẩu tin riêng lẻ là ẩn danh, nhưng khi kết hợp lại vẫn có thể suy đoán ra danh tính.

Part VI Secure Software Lifecycle Management

chapter 14 Secure Configuration and Version Control

Secure Configuration and Version Control

Phần **Secure Configuration and Version Control** (Cấu hình An toàn và Kiểm soát Phiên bản) trong Chương 14 tập trung vào việc quản lý sự thay đổi và thiết lập các thông số vận hành cho phần mềm trong suốt vòng đời của nó. Dưới đây là nội dung chi tiết:

1. Khái niệm Kiểm soát Cấu hình (Configuration Control)

- **Định nghĩa:** Cấu hình đề cập đến các yếu tố cụ thể (như các tệp cấu hình) điều khiển cách phần mềm vận hành và kích hoạt các tính năng tại thời điểm thực thi.

- **Phạm vi quản lý:** Quản lý cấu hình bao gồm các vấn đề liên quan đến cấu hình phần cứng, phần mềm, tài liệu, giao diện và các quy trình vá lỗi.

2. Kiểm soát Phiên bản (Version Control/Revision Control)

- **Mục tiêu:** Đánh dấu duy nhất và quản lý từng bản phát hành khác nhau của phần mềm.
- **Cơ chế hoạt động:**
    - Sử dụng các con số hoặc ký tự để phân biệt (ví dụ: số bên trái dấu chấm là bản phát hành chính, số bên phải là mức độ thay đổi nhỏ).
    - Cho phép quay lại (rollback) các phiên bản cũ hơn nếu bản build hiện tại gặp lỗi.
    - **Quản lý truy cập:** Hệ thống có thể khóa các đoạn mã nguồn để tại một thời điểm chỉ có một lập trình viên được chỉnh sửa, tránh việc ghi đè công việc của nhau.
- **Tính tự động hóa:** Do khối lượng chi tiết rất lớn, việc quản lý phiên bản và cấu hình tốt nhất nên được thực hiện bằng hệ thống tự động để loại bỏ lỗi do con người.

3. Chiến lược và Lộ trình (Strategy and Roadmap)

- **Trách nhiệm:**
    - **Khách hàng (Customer):** Chịu trách nhiệm bảo trì sản phẩm sau khi phát hành.
    - **Nhà cung cấp (Supplier):** Chịu trách nhiệm quản lý cấu hình sản phẩm trước khi phát hành và viết **Kế hoạch Quản lý Cấu hình**.
- **Sự phối hợp:** Kế hoạch quản lý cấu hình cần được xây dựng chung giữa nhà cung cấp và khách hàng để đảm bảo mọi trách nhiệm được hiểu rõ và duy trì.
- **Kiểm toán (Audits):** Việc đảm bảo an ninh và chất lượng được thực hiện thông qua các cuộc kiểm toán độc lập với đội ngũ phát triển.

4. Quản lý An ninh trong các Phương pháp luận Phát triển

An ninh có thể được nhúng vào bất kỳ phương pháp luận nào (Agile hoặc Waterfall):

- **Phương pháp thích ứng (Adaptive/Agile):** Tập trung vào các chu kỳ phát triển nhỏ, nhanh (như Scrum, XP). Việc nhúng an ninh vào Agile giúp phát hiện lỗi cực sớm, tiết kiệm chi phí sửa chữa.
- **Phương pháp dự đoán (Predictive/Waterfall):** Là quy trình tuyến tính, tuần tự. An ninh trong Waterfall cần được xác định ngay từ giai đoạn yêu cầu và thiết kế vì việc quay lại các bước trước đó rất khó khăn và tốn kém.

5. Các thành phần cốt lõi của SDL trong Chương 14

Để vận hành an ninh cấu hình, quy trình SDL bao gồm:

- **Security Gates (Cổng an ninh):** Các điểm kiểm tra bắt buộc phải vượt qua để đảm bảo các yêu cầu an ninh được đáp ứng trước khi tiến sang giai đoạn tiếp theo.
- **Bộ công cụ:** Bao gồm theo dõi lỗi (bug tracking), mô hình hóa mối đe dọa (threat modeling) và kiểm thử xâm nhập/fuzzing.

Define Strategy and Roadmap

Nội dung phần **Define Strategy and Roadmap** (Xác định Chiến lược và Lộ trình) thuộc Chương 14 tập trung vào việc phân định trách nhiệm và các quy trình quản lý thay đổi giữa nhà cung cấp và khách hàng. Dưới đây là chi tiết từ nguồn tài liệu:

1. Vai trò và Trách nhiệm

Việc quản lý cấu hình liên quan đến hai vai trò chính với các trách nhiệm cụ thể:

- **Nhà cung cấp (Supplier):** Chịu trách nhiệm quản lý cấu hình của sản phẩm **trước khi phát hành** và duy trì sản phẩm trong suốt vòng đời phát triển. Nhà cung cấp là bên trực tiếp soạn thảo **Kế hoạch Quản lý Cấu hình**.

- **Khách hàng (Customer):** Chịu trách nhiệm bảo trì sản phẩm **sau khi phát hành**. Khách hàng phối hợp với nhà cung cấp để xây dựng kế hoạch quản lý nhằm đảm bảo tính liên tục và hiểu rõ mọi trách nhiệm.

2. Nhân sự quản lý

- **Configuration Managers (Trình quản lý cấu hình):** Nhà cung cấp bổ nhiệm các giám sát viên cụ thể chịu trách nhiệm đảm bảo các yêu cầu trong kế hoạch được thực hiện xuyên suốt quá trình phát triển.
- **Đại diện khách hàng (Customer Representative):** Một cá nhân đại diện cho khách hàng được chỉ định tham gia vào quá trình phát triển. Người này có thẩm quyền giải quyết các vấn đề kiểm soát cấu hình phát sinh, phê duyệt các đề xuất thay đổi và đảm bảo quá trình chuyển giao quản lý cấu hình cho khách hàng diễn ra suôn sẻ.

3. Quản lý Nhà thầu phụ (Subcontractors)

- Nhà cung cấp (thông qua trình quản lý cấu hình) có trách nhiệm đảm bảo các nhà thầu phụ hiểu và tham gia đầy đủ vào việc duy trì kiểm soát cấu hình phù hợp.
- Tất cả các điều khoản trong thỏa thuận giữa bên sản xuất và khách hàng thường áp dụng cho cả nhà thầu phụ và phải được đưa vào kế hoạch quản lý cấu hình.

4. Các Quy trình và Thực thể Quản lý chính

Quản lý cấu hình bao gồm hai quy trình lớn là **Kiểm soát cấu hình (Configuration control)** và **Kiểm soát xác minh (Verification control)**, được thực hiện thông qua ba thực thể quản lý tương tác lẫn nhau:

- **Quản lý quy trình thay đổi (Change process management):** Bao gồm việc ủy quyền thay đổi, kiểm soát xác minh và xử lý phát hành.
- **Kiểm soát đường cơ sở (Baseline control):** Bao gồm hạch toán thay đổi và quản lý thư viện.
- **Xác minh cấu hình (Configuration verification):** Bao gồm hạch toán trạng thái để xác minh tính tuân thủ với các thông số kỹ thuật.

Manage Security Within a Software Development Methodology

Nội dung phần **Manage Security Within a Software Development Methodology** (Quản lý Bảo mật trong Phương pháp luận Phát triển Phần mềm) thuộc Chương 14 nhấn mạnh rằng bảo mật không phụ thuộc vào bất kỳ phương pháp luận cụ thể nào, mà có thể được nhúng vào bất kỳ quy trình nào để giảm thiểu lỗ hổng.

Dưới đây là các chi tiết chính:

1. Các thành phần SDL cốt lõi

Dù sử dụng phương pháp nào, quy trình SDL (Secure Development Lifecycle) thường bao gồm:

- **Đào tạo đội ngũ:** Thành viên cần kiến thức về thực hành, chính sách và thủ tục bảo mật hiện tại.
- **Cổng an ninh (Security Gates):** Các điểm kiểm tra bắt buộc phải vượt qua để đảm bảo tuân thủ yêu cầu bảo mật trước khi dự án tiến triển sang giai đoạn tiếp theo.
- **Bộ công cụ:** Sử dụng theo dõi lỗi (bug tracking), mô hình hóa mối đe dọa (threat modeling) và kiểm thử fuzzing.
- **Đánh giá bảo mật (Security Review):** Kiểm tra xem tất cả các hoạt động bảo mật bắt buộc đã được thực hiện đúng mức chưa.

2. Bảo mật trong Phương pháp luận Thích ứng (Adaptive/Agile)

- **Đặc điểm:** Tập trung vào sự đổi mới, hiệu quả của các nhóm nhỏ thông qua các vòng lặp phát triển nhanh và lặp lại.
- **Các loại hình phổ biến:** Bao gồm **Scrum** (tập trung vào quy trình), **XP** (Extreme Programming - tập trung vào khía cạnh con người), cùng với Lean, Kanban, và FDD.
- **Lợi ích:** Việc nhúng bảo mật vào Agile mang lại lợi ích lớn vì các lỗi được phát hiện và ngăn chặn rất sớm, giúp tiết kiệm thời gian và chi phí chỉnh sửa.

3. Bảo mật trong Phương pháp luận Dự đoán (Predictive/Waterfall)

- **Đặc điểm:** Một quy trình tuyến tính, tuần tự, trong đó mỗi bước phải được hoàn thành trước khi chuyển sang bước tiếp theo.
- **Hạn chế:** Rất khó để quay lại các giai đoạn trước nếu phát hiện yêu cầu mới hoặc lỗi thiết kế, dẫn đến chi phí và thời gian tăng cao.
- **Chiến lược bảo mật:** Cần xác định các vấn đề bảo mật càng sớm càng tốt vì việc can thiệp muộn trong mô hình thác nước rất tốn kém.

Identify Security Standards and Frameworks và Decommission Software

Dưới đây là nội dung chi tiết của hai phần **Identify Security Standards and Frameworks** và **Decommission Software** thuộc Chương 14:

1. Identify Security Standards and Frameworks (Xác định các Tiêu chuẩn và Khung Bảo mật)

Việc áp dụng các tiêu chuẩn và khung bảo mật giúp doanh nghiệp giảm thiểu số lượng và mức độ nghiêm trọng của các lỗ hổng, đồng thời thiết lập quy trình sửa lỗi an toàn.

- **Các nguồn tài liệu uy tín:** Bao gồm **BSA | The Software Alliance**, **OWASP**, và **SAFECode**.
- **Các thực hành cốt lõi cần hướng dẫn:**
    - Xác định tiêu chí cho các lần kiểm tra và cổng an ninh (security gates).
    - Cấu hình quy trình biên dịch và xây dựng (build) để cải thiện bảo mật file thực thi.
    - Bảo vệ môi trường build khỏi sự truy cập trái phép và giả mạo.
    - Sử dụng chữ ký số để xác minh tính toàn vẹn của phần mềm xuyên suốt vòng đời.
    - Kiểm tra tất cả mã nguồn (bao gồm cả phần mềm bên thứ ba) để tìm lỗ hổng.
    - Thiết lập các cấu hình bảo mật ở mức mặc định (secure by default).

- **Tầm quan trọng của quy trình:** Tài liệu nhấn mạnh việc có các quy trình được thiết kế tốt quan trọng hơn việc chỉ dựa vào kỹ năng cá nhân của lập trình viên.

2. Decommission Software (Hủy bỏ/Ngừng hoạt động Phần mềm)

Mục tiêu của quá trình này là chấm dứt sự tồn tại của một hệ thống hoặc thực thể phần mềm một cách an toàn.

- **Rủi ro từ dữ liệu cũ:** Các hệ thống cũ vẫn giữ lại thông tin (khái niệm **magnetic remanence**), do đó cần xử lý để đảm bảo dữ liệu được bảo mật ngay cả khi không còn sử dụng.
- **Kế hoạch hủy bỏ (Disposal Plan):** Cần một kế hoạch bằng văn bản chi tiết về cách thức ngừng hỗ trợ, lưu trữ các tạo tác (artifacts) và thông báo cho người dùng.
- **Thông báo cho người dùng:** Cần thông báo rõ lý do ngừng hỗ trợ, các lựa chọn thay thế hoặc nâng cấp, và ngày sản phẩm mới sẽ sẵn sàng.
- **Chính sách End-of-Life (EOL):** Phải bao gồm các tiêu chí về việc dừng hoạt động (sunsetting), thông báo về phần cứng/phần mềm bị thay thế và thời gian hỗ trợ kỹ thuật còn lại sau thông báo.
- **Xử lý dữ liệu (Data Disposition):** Khi hủy bỏ phần mềm, dữ liệu cần được đánh giá để quyết định giữ lại (nếu còn giá trị kinh doanh) hoặc tiêu hủy an toàn.

Define and Develop Security Documentation

Phần **Define and Develop Security Documentation** (Xác định và Xây dựng Tài liệu Bảo mật) trong Chương 14 nhấn mạnh rằng tài liệu chỉ có giá trị khi nó được chia sẻ minh bạch và mọi thành viên trong đội ngũ đều hiểu rõ để cùng phối hợp.

1. Tính minh bạch và Sự phối hợp (Transparency and Playbook)

- **Giá trị của tài liệu:** Các tài liệu mô tả quy trình xây dựng phần mềm an toàn sẽ trở nên vô giá trị nếu đội ngũ thực hiện không thể tiếp cận hoặc không hiểu các bước trong đó.

- **Nguyên tắc "Cùng một kịch bản":** Tất cả các nhóm—từ người thu thập yêu cầu, nhà thiết kế, kiến trúc sư, lập trình viên đến kiểm thử viên và quản lý—đều phải làm việc dựa trên **cùng một bộ tài liệu hướng dẫn (playbook)** để đảm bảo sự đồng bộ.

2. Các loại tài liệu bảo mật trọng yếu cần chia sẻ

- **Mô hình hóa mối đe dọa (Threat Modeling):** Đây không chỉ là công việc của các chuyên gia mô hình hóa mà là một **công cụ giao tiếp mở**. Nó giúp minh họa nơi các mối đe dọa tương tác với phần mềm và nêu bật những nơi cần biện pháp giảm thiểu.
- **Báo cáo lỗi (Bug Reports):** Việc chia sẻ thông tin về các lỗi bảo mật trước đó là cực kỳ quan trọng. Tài liệu cần mô tả chi tiết lỗ hổng, nguồn gốc phát sinh và cách khắc phục để **tránh lặp lại sai lầm cũ**.
- **Yêu cầu bảo mật và Tiêu chuẩn lập trình:** Những tài liệu này chỉ có giá trị khi mọi thành viên đều biết đến và tuân thủ chúng một cách nghiêm ngặt.
- **Kế hoạch kiểm thử (Test Plans):** Nội dung kiểm thử cụ thể không chỉ quan trọng với người kiểm thử mà còn giúp các nhà thiết kế và lập trình viên hiểu rõ mục tiêu bảo mật cần đạt được.

Develop Security Metrics

Phần **Develop Security Metrics** (Phát triển các Chỉ số Bảo mật) trong Chương 14 tập trung vào việc định lượng và đo lường các nỗ lực bảo mật để quản lý quy trình phát triển một cách hiệu quả. Dưới đây là các nội dung chi tiết từ nguồn tài liệu:

1. Vai trò của Chỉ số Bảo mật

- **Mục đích:** Là một phần của quy trình quản lý nhằm đo lường các mục tiêu và có phương tiện để chấm điểm tiến độ thực hiện.
- **Vấn đề với việc đếm lỗi đơn thuần:** Việc chỉ đếm số lượng lỗi được khắc phục là cách làm phổ biến nhưng nó **không có khả năng mở rộng tốt (scale well)** khi quy mô dự án thay đổi.

- **Chỉ số chuẩn hóa:** Tài liệu đề xuất sử dụng các chỉ số đã được chuẩn hóa như **"số khiếm khuyết trên mỗi nghìn dòng mã" (defects per thousand lines of code)** để phản ánh chính xác hơn chất lượng mã nguồn.

2. Các chỉ số quan trọng cần đo lường

Để hiểu rõ những gì đang xảy ra và chi phí liên quan đến lỗi, các tổ chức nên theo dõi:

- **Số lượng lỗi lặp lại (repeated errors):** Đo lường việc đội ngũ có học hỏi từ sai lầm cũ hay không.
- **Số lượng lỗi phổ biến:** Ví dụ như các lỗi nằm trong danh sách Top 10 (OWASP).
- **Tỷ lệ phần trăm các mục nằm trên một mức độ nghiêm trọng cụ thể.**
- **Thời gian khắc phục (Remediation time):** Ví dụ như thời gian trung bình để sửa một lỗi bảo mật.
- **Thước đo về độ phức tạp liên quan đến lỗi.**

3. Giá trị của việc theo dõi xu hướng

- **Ít giá trị tức thời:** Hầu hết các phép đo này có rất ít giá trị khi xem xét tại một thời điểm đơn lẻ (spot utility).
- **Giá trị dài hạn:** Các **xu hướng theo thời gian (trends over time)** mới là yếu tố giúp cải thiện quy trình phát triển.
- **Học hỏi từ Quản lý Chất lượng:** Việc áp dụng các khái niệm từ quản lý chất lượng vào việc quản lý khiếm khuyết giúp gia tăng giá trị cho các nỗ lực cải thiện quy trình lập trình an toàn.

Decommission Software

Phần **Decommission Software** (Hủy bỏ/Ngừng hoạt động phần mềm) là giai đoạn cuối cùng trong vòng đời phần mềm, tập trung vào việc chấm dứt sự tồn tại của một hệ thống một cách an toàn và có kiểm soát.

1. Mục đích và Rủi ro khi hủy bỏ phần mềm

- **Mục đích:** Chấm dứt sự tồn tại của một hệ thống hoặc thực thể phần mềm một cách an toàn.
- **Rủi ro từ dữ liệu cũ:** Các hệ thống cũ thường vẫn lưu giữ thông tin thông qua hiện tượng **từ dư (magnetic remanence)**. Do đó, việc tiêu hủy phải đảm bảo mọi thông tin nhạy cảm đã được bảo mật hoặc xóa sạch hoàn toàn.
- **Ngừng hỗ trợ:** Quyết định hủy bỏ cũng đồng nghĩa với việc tổ chức sẽ ngừng các hoạt động hỗ trợ tích cực cho sản phẩm đó.

2. Kế hoạch Hủy bỏ (Disposal Plan)

Việc hủy bỏ không được diễn ra tùy tiện mà phải theo một kế hoạch bằng văn bản được phê duyệt bởi chủ sở hữu hệ thống. Kế hoạch này bao gồm:

- **Cách thức thực hiện:** Chi tiết các bước để vô hiệu hóa, tháo dỡ và loại bỏ các thành phần của sản phẩm.
- **Thông báo cho người dùng:** Phải thông báo kịp thời cho người dùng về lý do ngừng hỗ trợ, các lựa chọn thay thế hoặc nâng cấp và ngày sản phẩm mới sẽ sẵn sàng.
- **Lưu trữ tạo tác (Artifacts):** Tất cả tài liệu phát triển, nhật ký (logs) và mã nguồn của hệ thống cũ phải được lưu trữ an toàn để duy trì lịch sử tổ chức.

3. Chính sách End-of-Life (EOL) và Xử lý dữ liệu

- **Chính sách EOL:** Cần quy định rõ tiêu chí dừng hoạt động (sunsetting), thông báo về các sản phẩm bị thay thế và thời gian hỗ trợ kỹ thuật còn lại sau khi có thông báo chính thức.
- **Xử lý dữ liệu (Data Disposition):** Khi phần mềm ngừng hoạt động, dữ liệu đi kèm vẫn phải được đánh giá. Nếu dữ liệu còn giá trị kinh doanh, nó cần được lưu trữ tiếp; nếu không, nó phải được tiêu hủy an toàn theo quy trình đã định.
- **Thu hồi thông tin xác thực:** Cần lập kế hoạch để thu hồi các thông tin xác thực (credentials) dành riêng cho phần mềm đó để tránh bị lợi dụng sau khi hệ thống đã đóng cửa.

Report Security Status

Phần **Report Security Status** (Báo cáo Trạng thái Bảo mật) trong Chương 14 tập trung vào việc chuyển đổi các dữ liệu đo lường thành thông tin hữu ích để cấp quản lý có thể đưa ra quyết định chính xác.

Dưới đây là các nội dung chi tiết từ nguồn tài liệu:

1. Tầm quan trọng của số liệu đối với quản lý

- Cấp quản lý vận hành thông qua việc đo lường các hoạt động và hành động chính, sau đó phân tích các kết quả thu được.
- Việc thiết lập một bộ chỉ số quy trình then chốt (key process metrics) là bắt buộc để cấp quản lý có thể đánh giá chính xác các điều kiện hiện tại.
- Quản lý mà không có chỉ số cũng giống như "bắn vào bóng tối" để giải quyết những vấn đề không xác định, thường gây ra nhiều rắc rối hơn là giải pháp.

2. Nội dung báo cáo hiệu suất

- Mặc dù các dự án thường bám sát câu thần chú "đúng hạn, đúng ngân sách", nhưng việc quản lý thực tế đòi hỏi các báo cáo và chỉ số chi tiết hơn nhiều so với hai yếu tố này.
- Báo cáo hiệu suất chứa thông tin về những gì doanh nghiệp đã xác định là chỉ số chính để giữ cho ban quản lý nắm bắt được các hoạt động đang diễn ra.

3. Phương pháp báo cáo trực quan

- **Hiểu về lịch sử:** Để hiểu trạng thái hiện tại của hầu hết các chỉ số, cần phải có sự hiểu biết về các tiền lệ lịch sử.
- **Biểu đồ tiến trình (Run charts):** Đây là một phương pháp báo cáo phổ biến, hiển thị cách các chỉ số ghi điểm theo thời gian.
- **Bảng điều khiển (Dashboards):** Các triển khai hiện đại thường sử dụng Dashboard để cấp quản lý có thể xem xét nhanh chóng. Một hệ thống bảng điều khiển tốt sẽ cho

phép người dùng nhấp vào một trạng thái cụ thể để xem chi tiết (drill-down) sâu hơn vào biểu đồ tiến trình tương ứng.

4. Lựa chọn chỉ số phù hợp

- Khi xây dựng báo cáo, cần tham khảo lại phần "Phát triển các chỉ số bảo mật" để chọn lọc những thông tin thực sự phản ánh quy trình, tránh sử dụng những số liệu có thể gây hiểu lầm.

chapter 15 Software Risk Management

Incorporate Integrated Risk Management

Phần **Incorporate Integrated Risk Management (IRM - Tích hợp quản trị rủi ro)** trong Chương 15 của tài liệu cung cấp một khung làm việc toàn diện để kết nối rủi ro phần mềm với quản trị doanh nghiệp. Dưới đây là nội dung chi tiết theo các đầu mục con của phần này:

1. Tổng quan về IRM

Quản trị rủi ro tích hợp (IRM) là việc sử dụng một bộ các thực hành và quy trình, được hỗ trợ bởi các công nghệ giúp quản lý rủi ro từ nhiều nguồn khác nhau trong toàn bộ doanh nghiệp. Mục tiêu của IRM là tạo ra một cái nhìn tích hợp về rủi ro để cải thiện hiệu suất và khả năng ra quyết định. Việc kết nối nỗ lực rủi ro phần mềm với hệ thống IRM của doanh nghiệp giúp tạo ra các báo cáo quản trị theo định dạng quen thuộc và thu thập thông tin rủi ro toàn diện hơn các hệ thống báo cáo đặc thù.

2. Quy định và Tuân thủ (Regulations and Compliance)

Ban quản lý có trách nhiệm đảm bảo tuân thủ các yêu cầu từ nhiều nguồn khác nhau gắn liền với mục tiêu kinh doanh.

- **Sự khác biệt:** Tài liệu phân biệt **Tuân thủ (Compliance)** là việc đáp ứng các yêu cầu bên ngoài (luật pháp, quy định), trong khi **Phù hợp (Conformance)** liên quan đến các yêu cầu nội bộ (chính sách, tiêu chuẩn tổ chức).

- **Ưu tiên:** Các hoạt động tuân thủ (Compliance) thường được ưu tiên hơn vì việc vi phạm các quy định bên ngoài thường dẫn đến các **hình phạt tài chính trực tiếp và đáng kể**.

3. Vấn đề Pháp lý (Legal)

Quản trị bao gồm việc quản lý các yếu tố rủi ro do pháp luật thúc đẩy, nổi bật nhất là:

- **Sở hữu trí tuệ (Intellectual Property):** Đây là tài sản vô hình giá trị nhưng dễ bị đánh cắp, đặc biệt là qua biên giới quốc tế, nên cần các biện pháp kiểm soát phòng ngừa thay vì chỉ dựa vào biện pháp pháp lý sau khi đã mất.
- **Vi phạm dữ liệu (Data Breach):** Khi thông tin định danh cá nhân (PII) bị mất, doanh nghiệp phải đối mặt với các luật thông báo vi phạm dữ liệu và chi phí phản ứng sự cố. **Mã hóa** là phương pháp chính được sử dụng để bảo vệ tính bảo mật và đáp ứng các yêu cầu pháp lý như PCI DSS.

4. Tiêu chuẩn và Hướng dẫn (Standards and Guidelines)

Các tiêu chuẩn thiết lập các chuẩn mực chung để định nghĩa các quy tắc hành vi và đảm bảo mức độ chất lượng cụ thể cho sản phẩm.

- **Nguồn tiêu chuẩn:** Bao gồm ISO, PCI, NIST, OWASP, SAFECode, SAMM và BSIMM.
- Việc áp dụng các tiêu chuẩn này cung cấp một bản thiết kế để thiết kế, tạo ra và vận hành các hệ thống phản ánh các thực hành tốt nhất (best practices).

5. Lựa chọn xử lý rủi ro (Risk Management Options)

Khi xác định được rủi ro, ban quản lý có bốn lựa chọn chính:

- **Remediation (Sửa chữa/Khắc phục):** Giải quyết tận gốc nguyên nhân gây ra lỗ hổng. Đây là phương pháp được ưu tiên nhất.
- **Mitigation (Giảm thiểu):** Giảm tác động của rủi ro bằng cách loại bỏ tính năng gây lỗi hoặc thêm các biện pháp kiểm soát bù đắp (như mã hóa).
- **Transfer (Chuyển giao):** Đẩy rủi ro sang bên thứ ba (như mua bảo hiểm) hoặc cảnh báo để người dùng tự chịu rủi ro.

- **Accept (Chấp nhận):** Chấp nhận rủi ro và các hệ quả đi kèm nếu nó xảy ra. Điều này thường áp dụng cho **Rủi ro còn lại (Residual risk)** - phần rủi ro còn tồn tại sau khi các biện pháp kiểm soát đã được áp dụng.

6. Thuật ngữ Quản trị rủi ro (Terminology)

Các khái niệm cốt lõi cần nắm vững:

- **Threat (Mối đe dọa):** Bất kỳ tình huống hoặc sự kiện nào có khả năng gây hại cho tài sản.
- **Vulnerability (Lỗ hổng):** Đặc điểm của tài sản có thể bị mối đe dọa khai thác.
- **Impact (Tác động):** Sự tổn thất xảy ra khi một mối đe dọa khai thác thành công lỗ hổng.
- **Controls (Biện pháp kiểm soát):** Các biện pháp (kỹ thuật, hành chính hoặc vật lý) được thực hiện để phát hiện, ngăn chặn hoặc giảm thiểu rủi ro.

7. Rủi ro Kỹ thuật vs. Rủi ro Kinh doanh

- **Rủi ro kinh doanh (Business Risk):** Gắn liền với các hoạt động vận hành doanh nghiệp nói chung.
- **Rủi ro kỹ thuật (Technical Risk):** Là một tập hợp con quan trọng của rủi ro kinh doanh, liên quan đến các công nghệ được sử dụng trong quá trình phát triển hoặc các khía cạnh chức năng của phần mềm.

Promote Security Culture in Software Development

Phần **Promote Security Culture in Software Development** (Thúc đẩy Văn hóa Bảo mật trong Phát triển Phần mềm) thuộc Chương 15 nhấn mạnh rằng con người và thái độ của họ đối với bảo mật là yếu tố quyết định sự thành công của bất kỳ quy trình kỹ thuật nào.

Dưới đây là các nội dung chi tiết:

1. Xây dựng Văn hóa Bảo mật

- **Bản chất của văn hóa:** Phát triển phần mềm là một "môn thể thao đồng đội". Để bảo mật trở thành một phần của kết quả công việc, nó phải được nhúng vào văn hóa làm việc của nhóm.
- **Vai trò của Quản lý:** Văn hóa không thể bị ép buộc hoặc làm giả. Nếu ban quản lý không "nói đi đôi với làm" và không thực sự tham gia vào văn hóa đó, nhân viên sẽ nhận ra sự giả tạo và các nỗ lực bảo mật sẽ bị xem thường.

2. Đội ngũ Security Champions (Đại sứ Bảo mật)

- **Định nghĩa:** Họ là những người dẫn dắt và chỉ đường khi các thành viên khác bị mất phương hướng. Họ đóng vai trò là một nguồn lực hỗ trợ tại chỗ.
- **Vị trí:** Họ không nhất thiết phải là quản lý; thực tế, họ thường là thành viên trong các nhóm sản xuất bình thường để đồng nghiệp dễ tiếp cận hơn.
- **Nhiệm vụ:** Họ đóng vai trò như "người cổ vũ" để giữ cho tinh thần và văn hóa bảo mật của nhóm luôn đồng nhất. Họ không cần phải có mọi câu trả lời, nhưng cần có kinh nghiệm để biết nơi nào có thể tìm thấy câu trả lời.

3. Giáo dục và Hướng dẫn Bảo mật

- **Đào tạo theo vai trò:** Việc đào tạo phải tập trung vào vai trò và trách nhiệm cụ thể của từng thành viên trong nhóm.
- **Phân loại kiến thức:**
    - **Kiến thức cơ bản:** Đây là vấn đề của toàn đội ngũ (all-hands); mọi thành viên đều cần có kiến thức vận hành về cách bảo mật được áp dụng trong SDL.
    - **Chủ đề nâng cao:** Nhắm vào các vai trò cụ thể như nhà thiết kế, lập trình viên, kiểm thử viên hoặc quản lý dự án (ví dụ: các mối đe dọa mới, công cụ mới).
- **Thời điểm:** Điều quan trọng nhất là đảm bảo tất cả các thành viên được trang bị kiến thức đúng đắn **trước khi** họ bắt đầu tham gia vào quy trình phát triển. Đào tạo định kỳ là cần thiết để cập nhật các xu hướng và mối đe dọa luôn thay đổi.

Implement Continuous Improvement

Phần **Implement Continuous Improvement** (Triển khai Cải tiến Liên tục) trong Chương 15 nhấn mạnh rằng việc cải thiện quy trình không phải là một hành động đơn lẻ mà là một nỗ lực chủ động và lâu dài để đạt được kết quả kinh doanh tốt hơn.

Dưới đây là các nội dung chi tiết:

1. Bản chất của Cải tiến Liên tục

- **Định nghĩa:** Là quá trình hoàn thiện các quy trình làm việc nhằm nâng cao kết quả kinh doanh.
- **Tính chủ động:** Cải tiến liên tục phải là một bước **chủ động (proactive)** chứ không phải phản ứng (reactive) khi có sự cố xảy ra.
- **Tầm quan trọng:** Đây là yếu tố sống còn để doanh nghiệp tồn tại lâu dài, giúp kiểm soát quy trình và khắc phục các điểm chưa tối ưu.

2. Đánh giá sau hành động (After-Action Reviews)

Khi kết quả không như mong đợi, nhóm cần thực hiện các buổi nhìn lại (retrospective). Một bài học quan trọng từ lịch sử quân sự (Chiến tranh Việt Nam) được tài liệu nhắc đến là:

- **Gác lại cấp bậc và cái tôi:** Trong các buổi đánh giá, **cấp bậc (rank) không có chỗ đứng**. Mọi thành viên phải được tự do báo cáo sự thật mà không sợ bị trù dập bởi người có chức vụ cao hơn.
- **Tập trung vào sự thật:** Việc loại bỏ các yếu tố gây nhiễu (cái tôi, địa vị) giúp nhóm tập trung vào mục tiêu duy nhất là cải tiến quy trình.

3. Phân tích Nguyên nhân Gốc rễ (Root-Cause Analysis)

- **Hành động dựa trên nguyên nhân:** Việc cải tiến chỉ hiệu quả khi tác động vào **nguyên nhân gốc rễ** chứ không phải chỉ xử lý các triệu chứng bên ngoài.
- **Đào sâu thông tin:** Nhóm cần phải đào sâu qua các lớp thông tin gây nhiễu để tìm ra nguyên nhân thực sự, nếu không, nỗ lực cải tiến sẽ không mang lại kết quả mong muốn,.

4. Thích nghi với sự thay đổi

Môi trường phần mềm luôn biến đổi về công nghệ, mối đe dọa, quy định và thực hành kinh doanh. Quy trình cải tiến liên tục giúp tổ chức:

- Luôn cập nhật và theo kịp những thay đổi này.
- Cải thiện các chỉ số (metrics) mong muốn theo thời gian.

Part VII Secure Software Deployment, Operations, Maintenance

Chapter 16 Secure Software Deployment

Perform Operational Risk Analysis

Phần **Thực hiện Phân tích Rủi ro Vận hành (Perform Operational Risk Analysis)** thuộc Chương 16 tập trung vào việc đảm bảo mọi rủi ro tiềm ẩn được hạch toán và đánh giá đầy đủ trước khi bàn giao và nghiệm thu sản phẩm phần mềm,. Dưới đây là nội dung chi tiết của các đầu mục con trong phần này:

1. Các thuộc tính rủi ro cần xem xét

Do tính phức tạp của phần mềm và sự đa dạng của các mối đe dọa không gian mạng, rủi ro là một yếu tố hiển nhiên trong bất kỳ sản phẩm bàn giao nào. Quy trình phân tích rủi ro vận hành phải xem xét các thuộc tính sau để đảm bảo hệ thống vận hành trong giới hạn an toàn:

- **Yêu cầu an toàn (Safety) và bảo mật (Security):** Bao gồm cả các yêu cầu rõ ràng và ngầm định.
- **Mức độ phức tạp của phần mềm:** Phần mềm càng phức tạp thì rủi ro tiềm ẩn càng cao.
- **Các yếu tố hiệu suất (Performance) và độ tin cậy (Reliability)**.

2. Quá trình đánh giá rủi ro (Risk Evaluation)

Một quy trình đánh giá rủi ro đúng chuẩn phải trả lời được hai câu hỏi then chốt:

- **Xác suất (Likelihood):** Mức độ chắc chắn hoặc khả năng xảy ra sự kiện rủi ro là bao nhiêu?
- **Tác động (Impact):** Ước tính mức độ tổn thất, nguy hiểm hoặc sai sót nếu rủi ro thực sự xảy ra.

Mục tiêu cơ bản: Tối đa hóa việc sử dụng nguồn lực bằng cách ưu tiên xử lý những rủi ro có xác suất xảy ra cao nhất và gây ra tác động nghiêm trọng nhất.

3. Tầm quan trọng của bằng chứng cụ thể (Concrete Evidence)

Vì phần mềm là một sản phẩm vô hình và năng động, các đánh giá rủi ro không thể chỉ dựa trên suy đoán mà phải được xây dựng dựa trên **bằng chứng hữu hình**,. Các bằng chứng này thu được thông qua:

- Các bài kiểm thử (tests).
- Các buổi đánh giá (reviews) và phân tích mối đe dọa.
- Các đánh giá kỹ thuật và quản lý có liên quan khác.

4. Phương pháp luận và thu thập dữ liệu

Dữ liệu rủi ro phải được thu thập một cách có hệ thống, phối hợp tốt và sử dụng phương pháp luận có khả năng lặp lại để tạo ra bằng chứng đáng tin cậy, có thể xác minh độc lập.

- **Tính chia nhỏ (Compartmentalization):** Tổ chức có thể thực hiện một chuỗi các đánh giá tập trung vào từng mối đe dọa cụ thể, sau đó tổng hợp lại thành một bức tranh rủi ro toàn diện thay vì thực hiện một nỗ lực đơn lẻ duy nhất,.

5. Đánh giá tình huống (Situational Assessment)

Bức tranh đe dọa (threat picture) phải được cập nhật liên tục dựa trên ba yếu tố tới hạn:

1. **Mối quan hệ tương quan** giữa tất cả các tài sản của hệ thống.
2. **Các mối đe dọa cụ thể** đối với từng tài sản.
3. **Rủi ro kinh doanh và kỹ thuật chính xác** liên quan đến từng lỗ hổng.

- **Lưu ý:** Cần phải theo dõi cả các **mối đe dọa tiềm tàng (latent threats)** vì chúng có thể trở thành mối đe dọa hiện hữu nếu các điều kiện môi trường thay đổi.

6. Đào tạo nhân sự (Personnel Training)

Việc đào tạo là yếu tố then chốt để đảm bảo phần mềm vận hành an toàn. Có ba nhóm đối tượng cần được đào tạo chuyên biệt:

- **Quản trị viên (Administrators):** Cần kiến thức cụ thể để thiết lập và khởi tạo hệ thống đúng cách.
- **Người dùng quyền năng (Power Users):** Cần hiểu cách triển khai hệ thống để hướng dẫn người dùng khác.
- **Người dùng tiêu chuẩn (Standard Users):** Cần biết cách thực hiện các nhiệm vụ kinh doanh trên hệ thống mới một cách an toàn.

7. Tính trọng yếu về an toàn (Safety Criticality)

Đối với các ứng dụng được chỉ định là an toàn trọng yếu, rủi ro vận hành cần tập trung đặc biệt vào quản trị rủi ro, kiểm soát thay đổi và độ tin cậy để tích hợp trơn tru vào môi trường làm việc thực tế.

8. Tích hợp hệ thống (System Integration)

Trong các hệ thống của hệ thống (system-of-systems), mục tiêu là tạo ra sự cộng hưởng từ việc liên kết các thành phần. Điều này đòi hỏi:

- Nỗ lực kỹ thuật và quản lý phối hợp cao để đảm bảo khả năng tương tác đồng thời và an toàn.
- Áp dụng các nguyên tắc thiết kế như: trừu tượng hóa (abstraction), tính mô-đun (modularity) và ẩn giấu thông tin (information hiding).

Release Software Securely

Nội dung phần **Release Software Securely** (Phát hành Phần mềm An toàn) thuộc Chương 16 tập trung vào các phương pháp hiện đại để đưa phần mềm từ môi trường phát triển đến tay khách hàng mà vẫn bảo toàn được tính bảo mật và toàn vẹn của mã nguồn.

Dưới đây là chi tiết các tiểu mục trong phần này:

1. Phương pháp DevOps và Secure DevOps

- **Khái niệm:** DevOps là một hình thức bảo trì phần mềm trong đó các thay đổi được đưa vào sản xuất gần như trực tiếp thông qua các quy trình tự động hóa cao để giảm thiểu rủi ro.
- **Lợi ích của thay đổi nhỏ:** Thay vì đợi các bản phát hành lớn (hàng quý hoặc hàng năm), DevOps áp dụng các thay đổi nhỏ, tập trung. Điều này giúp việc kiểm thử nhanh hơn, tương tác giữa các thành phần dễ kiểm soát hơn và sửa lỗi kịp thời hơn.
- **Tự động hóa:** Một biện pháp bảo vệ quan trọng trong DevOps là mức độ tự động hóa cao để triển khai và hoàn tác (rollback) các thay đổi nếu chúng thất bại.
- **Secure DevOps:** Đây là quy trình mà đội ngũ phát triển chịu trách nhiệm về mọi khía cạnh của mã, bao gồm cả bảo mật, cho đến khi đưa vào sản xuất.

2. Đường ống CI/CD An toàn (Continuous Integration & Delivery)

CI/CD là cách tiếp cận hiện đại để xây dựng, kiểm thử và triển khai các hệ thống IT bằng cách sử dụng tự động hóa để quản lý nhiều chức năng bảo trì.

- **Continuous Integration (Tích hợp liên tục):** Đánh dấu bằng nhiều thay đổi nhỏ và cam kết (commit) mã vào kho lưu trữ. Tại mỗi lần commit, mã sẽ được kiểm thử và xác minh tính phù hợp và chức năng ngay lập tức.
- **Continuous Delivery (Chuyển giao liên tục):** Mở rộng CI vào giai đoạn chuẩn bị sản xuất. Nó tự động hóa các cơ chế di chuyển mã vào sản xuất, giúp quy trình nhanh hơn và dễ quản lý hơn thông qua các kịch bản (scripts).
- **Continuous Deployment (Triển khai liên tục):** Mở rộng giao hàng liên tục đến quá trình triển khai thay đổi cho khách hàng mà không cần sự can thiệp thủ công của con người. Nếu xảy ra lỗi, các script sẽ tự động xử lý và đưa hệ thống về phiên bản tốt trước đó.

3. Chuỗi công cụ phần mềm an toàn (Secure Software Tool Chain)

- Để xây dựng phần mềm an toàn, đội ngũ cần được đào tạo và sử dụng các công cụ hỗ trợ quy trình bảo mật một cách nhất quán.

- Cần có sự tiêu chuẩn hóa các công cụ được sử dụng cũng như phương pháp sử dụng chúng, bao gồm: kho lưu trữ mã nguồn, các phương pháp kiểm tra lỗi và cú pháp tiêu chuẩn, và thiết lập các tùy chọn trình biên dịch (compiler options) phù hợp.

4. Xác minh tạo tác xây dựng (Build Artifact Verification)

- Khi các thành phần của sản phẩm di chuyển qua chuỗi cung ứng, chúng phải được xác thực và đảm bảo tính toàn vẹn tại các điểm giao tiếp giữa các tổ chức hoặc các cấp độ khác nhau.
- **Kỹ thuật sử dụng:** Việc này được thực hiện thông qua sử dụng tổng kiểm (checksums), hàm băm (hashes) và chữ ký số.
- **Mục tiêu:** Loại bỏ các khả năng tấn công kiểu người đứng giữa (man-in-the-middle) và ngăn chặn việc đưa các thành phần giả mạo vào hệ thống.

Securely Store and Manage Security Data

Nội dung phần **Securely Store and Manage Security Data** (Lưu trữ và Quản lý Dữ liệu Bảo mật một cách An toàn) trong Chương 16 tập trung vào việc bảo vệ các tài sản nhạy cảm nhất của phần mềm khi nó được đưa vào vận hành thực tế.

Dưới đây là chi tiết các tiểu mục từ nguồn tài liệu:

1. Bảo vệ Kho lưu trữ Mã nguồn (Code Repository)

Mã nguồn là một tài sản trí tuệ có giá trị kinh tế cao và cần được bảo vệ khỏi mất mát hoặc thay đổi trái phép. Việc duy trì các kho lưu trữ an toàn thường được thực hiện thông qua các phần mềm quản lý kho lưu trữ (repository software), giúp quản lý các phiên bản cũng như kiểm tra tổng kiểm (checksum), hàm băm và chữ ký số.

2. Quản lý Thông tin xác thực (Credentials)

Tất cả quyền truy cập vào mã nguồn phải thông qua các thông tin xác thực đã được phê duyệt. Một nguyên tắc quan trọng là phải tách biệt hoàn toàn các môi trường: **phát triển (development), kiểm thử (testing) và vận hành (production)**. Việc sử dụng một bộ thông tin xác thực duy nhất cho nhiều môi trường là một thực hành xấu vì có thể dẫn đến việc

thay đổi nhầm lẫn dữ liệu vận hành khi người dùng tưởng rằng mình đang ở trong môi trường phát triển.

3. Quản lý Bí mật (Secrets)

Các bí mật trong phần mềm rất đa dạng, bao gồm khóa mã hóa, các tham số cấu hình, thông tin xác thực mà chương trình sử dụng và cả dữ liệu nhạy cảm chảy qua hệ thống. Đội ngũ phát triển cần xác định rõ dữ liệu nào cần được bảo vệ, mức độ bảo vệ và ai có quyền truy cập. Ngay cả khi hệ thống được thiết kế bảo mật, nó vẫn cần được kiểm thử lại khi triển khai để đảm bảo các biện pháp bảo vệ hoạt động đúng như mong đợi của nhà thiết kế.

4. Khóa và Chứng chỉ (Keys/Certificates)

Các khóa mã hóa là yếu tố then chốt trong bảo mật và cần được bảo vệ nghiêm ngặt. Việc triển khai thực tế bao gồm cấu hình hệ thống với môi trường vận hành, và các thiết lập này có thể ảnh hưởng đến tính bảo mật. Người kiểm thử cần biết về các khóa/chứng chỉ cần bảo vệ và thực hiện các bước kiểm tra sau khi triển khai để đảm bảo các kế hoạch bảo vệ đang hoạt động.

5. Cấu hình (Configurations)

Cấu hình phần mềm đóng vai trò quan trọng trong vị thế bảo mật của ứng dụng. Việc đảm bảo các cấu hình được bảo mật theo đúng thông số thiết kế là một khía cạnh then chốt để phần mềm vận hành an toàn trong sản xuất. Các phương pháp triển khai nên được kiểm thử để xác nhận các yếu tố cấu hình đã được bảo mật trước khi coi quá trình triển khai là thành công.

Ensure Secure Installation

Phần **Ensure Secure Installation** (Đảm bảo Cài đặt An toàn) trong Chương 16 tập trung vào việc đảm bảo phần mềm được triển khai và vận hành đúng với các thiết lập bảo mật đã định sẵn, ngay cả khi chuyển giao cho khách hàng.

Dưới đây là chi tiết các tiểu mục:

1. Bootstrapping (Khởi động tự lùi)

- **Định nghĩa:** Bootstrapping trong triển khai là quy trình diễn ra một lần để **đảm bảo tính chính xác của cấu hình ban đầu**. Nó bao gồm việc thiết lập các giá trị mặc

định, tham số thực thi và đảm bảo các tính năng bảo mật hoạt động chính xác trong sản phẩm vận hành.

- **Ví dụ:** Việc cấu hình các thiết lập màn hình tham chiếu trong hệ điều hành để đảm bảo kiểm soát truy cập, hoặc định nghĩa các thiết lập quản lý hạ tầng khóa công khai (PKI).
- **Tầm quan trọng:** Các quy trình khởi động như **POST (Power-on self-test)** và **IPL (Initial program load)** phải đảm bảo tính toàn vẹn tuyệt đối, vì nếu không, mọi quy trình tiếp sau trên máy đó đều không thể tin cậy.

2. Least Privilege (Quyền tối thiểu)

- **Nguyên tắc cốt lõi:** Hạn chế đặc quyền ở mức vừa đủ để thực hiện nhiệm vụ nhằm ngăn chặn rủi ro từ các sự cố ngoài ý muốn.
- **Cảnh báo:** Không nên để phần mềm chạy ở quyền quản trị viên (administrator) vì điều này sẽ loại bỏ các cơ chế bảo vệ của hệ điều hành và môi trường hệ thống.
- **Trách nhiệm lập trình viên:** Lập trình viên phải hiểu rõ rủi ro của việc nâng quyền và đảm bảo mã nguồn thực thi ở **mức đặc quyền thấp nhất có thể**.

3. Environment Hardening (Cường hóa môi trường)

- **Phạm vi:** Phần mềm hoạt động trong nhiều môi trường khác nhau (máy đơn, máy ảo, container, điện toán đám mây), và mỗi môi trường cần có biện pháp **cường hóa chống lại tấn công và lỗi**.
- **Yêu cầu triển khai:** Cần xác định và tài liệu hóa các yêu cầu cụ thể đối với môi trường như: các tài khoản duy nhất với quyền hạn riêng biệt, các quy tắc tường lửa cụ thể. Điều này đảm bảo các thiết lập bảo mật không bị vô hiệu hóa sau các kỳ kiểm toán tương lai.

4. Secure Activation (Kích hoạt An toàn)

- **Cấu hình và kết nối:** Sau khi cài đặt, phần mềm cần được kết nối với hệ thống quản lý thông tin xác thực của doanh nghiệp và đăng ký vào các danh sách trắng (whitelists) nếu hệ điều hành yêu cầu.

- **Tham số mạng:** Các thông số như cổng tường lửa phải được thiết lập và duy trì để các kênh liên lạc hoạt động ổn định và an toàn.
- **Cấp phép (Licensing):** Cần lưu ý các phần mềm yêu cầu gọi về máy chủ bản quyền qua Internet, vì chúng có thể bị hạn chế trong các **môi trường bảo mật bị cô lập**.

5. Security Policy Implementation (Triển khai Chính sách Bảo mật)

- **Truyền đạt kỳ vọng:** Chính sách bảo mật được dùng để truyền đạt các kỳ vọng về hiệu suất và trách nhiệm an ninh mạng cho người lao động.
- **Thực thi qua phần mềm:** Ứng dụng có thể thực thi các chính sách như **không lưu giữ dữ liệu khách hàng**. Những chính sách này cần được truyền đạt rõ ràng cho người dùng cuối trước khi họ cài đặt và sử dụng ứng dụng.

6. Secrets Injection (Tiêm mã bí mật)

- **Quản lý bí mật động:** Một số bí mật (như khóa SSH, mã thông báo OAuth) không có sẵn từ đầu mà được tạo ra và tiêm vào phần mềm trong quá trình vận hành.
- **Bảo vệ liên tục:** Cần có các phương pháp an toàn để nhận và sử dụng các bí mật này, đồng thời duy trì việc bảo vệ chúng xuyên suốt quá trình triển khai và vận hành.

Perform Post-Deployment Security Testing

Nội dung phần **Perform Post-Deployment Security Testing** (Thực hiện Kiểm thử Bảo mật sau Triển khai) trong Chương 16 nhấn mạnh rằng việc bảo mật phần mềm không kết thúc khi sản phẩm được cài đặt, mà phải được duy trì và kiểm tra liên tục trong môi trường vận hành thực tế.

Dưới đây là các chi tiết chính:

1. Mục đích của kiểm thử sau triển khai

- Việc thiết kế và phát triển phần mềm an toàn là chưa đủ; phần mềm phải được vận hành theo cách **duy trì các kỳ vọng bảo mật** đã đặt ra từ giai đoạn yêu cầu và thiết kế.

- Trong quá trình vận hành, những thay đổi trong môi trường hệ thống có thể ảnh hưởng đến cấu hình bảo mật của phần mềm, do đó cần có các quy trình kiểm tra để đảm bảo tính an toàn vẫn nguyên vẹn.

2. Kiểm tra các bản vá và cập nhật (Patch Testing)

- **Ví dụ điển hình:** Các dịch vụ cập nhật tự động (như của Microsoft) thực hiện nhiều bước kiểm tra "ngầm" để đảm bảo bản vá tải xuống là chính xác, không bị thay đổi và các tệp thực thi mới là hợp lệ.
- **Cơ chế mật mã:** Các bước kiểm tra này thường được thực hiện thông qua mật mã học mà không cần sự can thiệp của người dùng.
- **Khả năng hoàn tác (Rollback):** Nếu bất kỳ bước kiểm tra nào thất bại, quá trình cập nhật sẽ bị hủy bỏ và hệ thống sẽ được đưa về trạng thái ổn định trước đó.

3. Tự động hóa và Tuân thủ

- **Tự động xác minh:** Trong các môi trường có rủi ro cao, việc tự động hóa các quy trình xác minh hệ thống đang vận hành an toàn là cực kỳ quan trọng.
- **Hỗ trợ kiểm toán:** Việc ghi nhật ký (logging) các thông tin then chốt giúp chủ sở hữu hệ thống cung cấp bằng chứng cho các kiểm toán viên tuân thủ một cách tự động thông qua các dòng dữ liệu hoạt động an toàn.

Chapter 17 Secure Software Operations and Maintenance

Obtain Security Approval to Operate

Phần **Obtain Security Approval to Operate** (Nhận Phê duyệt Bảo mật để Vận hành) trong Chương 17 tập trung vào quy trình kiểm soát quản trị cuối cùng trước khi đưa phần mềm vào sử dụng thực tế. Dưới đây là nội dung chi tiết:

1. Bản chất của Phê duyệt Vận hành

- **Đánh giá rủi ro tổng thể:** Phần mềm trong môi trường kết nối cao hiện nay có thể gây ra nhiều rủi ro cho doanh nghiệp, dù là trực tiếp bên trong mã nguồn hay từ xa

thông qua các hệ thống liên kết. Do đó, phần mềm không được coi là một "hòn đảo rủi ro" biệt lập mà phải được xem xét trong bối cảnh toàn doanh nghiệp.

- **Sự tham gia của ban quản lý:** Hầu hết các doanh nghiệp áp dụng quy trình đánh giá của ban quản lý đối với rủi ro liên quan đến phần mềm mới, bao gồm việc yêu cầu chữ ký phê duyệt từ các nhân sự có thẩm quyền.
- **Nguồn gốc thuật ngữ:** Thuật ngữ "approval to operate" (phê duyệt vận hành) bắt nguồn từ nhiều quy trình quản lý của chính phủ.

2. Quy trình và Thời điểm thực hiện

- **Các mốc thời gian:** Việc phê duyệt rủi ro có thể diễn ra tại nhiều thời điểm trong vòng đời dự án, chẳng hạn như trước khi bắt đầu dự án hoặc trước các bước then chốt.
- **Thời điểm bắt buộc:** Quy trình này hầu như luôn được yêu cầu thực hiện trước khi dự án chính thức "go live" (đi vào hoạt động).
- **Trách nhiệm của cấp điều hành:** Điều quan trọng là các giám đốc điều hành cấp cao phù hợp phải xem xét tất cả rủi ro cấu hình và rủi ro hệ thống. Họ phải chính thức chấp thuận để hệ thống được kết nối và vận hành trên môi trường sản xuất thực tế.

Perform Information Security Continuous Monitoring

Phần **Perform Information Security Continuous Monitoring** (Thực hiện Giám sát Liên tục An toàn Thông tin - ISCM) trong Chương 17 tập trung vào việc duy trì nhận thức về trạng thái bảo mật của hệ thống sau khi đã triển khai.

Dưới đây là nội dung chi tiết:

1. Bản chất của ISCM

- **Định nghĩa:** Là quá trình duy trì nhận thức liên tục về an toàn thông tin, các lỗ hổng và mối đe dọa đối với doanh nghiệp.

- **Tính "liên tục":** Thuật ngữ này không có nghĩa là tức thời (instantaneous) mà có nghĩa là ở một tần suất đủ để hỗ trợ các quyết định quản lý rủi ro.
- **Thành phần chính:** Bao gồm các cảm biến (sensors) để thu thập dữ liệu và các nền tảng (platforms) để phân tích dữ liệu kết hợp với thông tin tình báo về mối đe dọa.
- **Tích hợp:** Phải được kết nối chặt chẽ với hệ thống phát hiện xâm nhập và phản ứng sự cố của tổ chức.

2. Các hoạt động cốt lõi trong ISCM

- **Thu thập và phân tích dữ liệu quan sát được (Security Observable Data):** Thu thập dữ liệu từ nhật ký (logs), trình kích hoạt sự kiện (event triggers), dữ liệu đo từ xa (telemetry) và dữ liệu vết (trace data) để phân biệt hoạt động kinh doanh bình thường với các hành vi bất thường.
- **Thông tin tình báo mối đe dọa (Threat Intel):** Sử dụng các nguồn tin từ nội bộ và bên ngoài để vẽ nên bức tranh về môi trường rủi ro hiện tại, từ đó điều chỉnh ứng dụng và các quy trình bảo mật cho phù hợp.
- **Phát hiện xâm nhập và Phản ứng (Intrusion Detection/Response):** Xác định các luồng giao tiếp mạng không phù hợp hoặc trái phép. Khi phát hiện hoạt động nghi vấn, hệ thống sẽ phân loại theo mức độ nghiêm trọng để có kế hoạch ứng phó.
- **Cấu hình an toàn (Secure Configuration):** Bảo vệ các tham số cấu hình (như tài khoản dịch vụ, kết nối cơ sở dữ liệu). Việc giám sát liên tục giúp phát hiện các thay đổi cấu hình trái phép.
- **Thay đổi quy định (Regulation Changes):** Giám sát các thay đổi về luật pháp hoặc quy định ảnh hưởng đến ứng dụng để đảm bảo duy trì tính tuân thủ (compliance) kịp thời.

Support Incident Response

Phần **Support Incident Response** (Hỗ trợ Phản ứng Sự cố) trong Chương 17 tập trung vào các quy trình và kỹ năng cần thiết để xử lý các sự kiện bất lợi làm gián đoạn điều kiện vận hành bình thường của phần mềm.

Dưới đây là nội dung chi tiết:

1. Tổng quan về Quản lý Sự cố

- **Định nghĩa:** Sự cố là bất kỳ sự kiện nào gây gián đoạn vận hành, từ lỗi người dùng, mất điện đến hoạt động tấn công độc hại.
- **Mục tiêu:** Quản lý sự cố nhằm duy trì khả năng phản ứng và tích hợp các hoạt động giám sát, phân tích để đưa ra phản hồi phù hợp.
- **Quy trình:** Phản ứng sự cố bắt đầu từ việc phát hiện sự kiện thông qua các chức năng giám sát (như IDS/IPS), sau đó là báo cáo chính thức để đảm bảo mọi sự cố đều được xử lý theo quy định của tổ chức.

2. Các hoạt động hỗ trợ then chốt

- **Root-Cause Analysis (Phân tích Nguyên nhân Gốc rễ):** Nhằm xác định lý do tại sao một sự cố xảy ra, phân biệt giữa lỗ hổng mã nguồn, nỗ lực khai thác hoặc lỗi vô ý của người dùng.
- **Incident Triage (Phân loại Sự cố):** Đảm bảo các báo cáo sự cố đến đúng người ra quyết định (Incident Manager) kịp thời. Qu trình này đánh giá loại hình và tác động của sự kiện, từ những lỗi nhỏ trong mã nguồn đến các mối đe dọa nghiêm trọng như Trojan hay Trapdoors.
- **Incident Response Teams:** Đội ngũ chuyên gia được đào tạo chuyên sâu để xử lý sự cố theo một quy trình nghiêm ngặt, giúp mang lại giải pháp tốt nhất thay vì để nhân sự tại chỗ xử lý.

- **Forensics (Pháp y máy tính):** Áp dụng các bài kiểm tra có thể lặp lại (repeatable tests) lên các hiện vật (artifacts) như mục nhật ký (log entries) để xác định ai đã làm gì và làm như thế nào.

3. Vai trò của Thiết kế Phần mềm trong Phản ứng Sự cố

- Phần mềm cần được thiết kế để hỗ trợ phản ứng sự cố thông qua việc **ghi nhật ký chủ động (proactive logging)**.
- **Threat modeling** (Mô hình hóa mối đe dọa) giúp xác định các điểm quan trọng cần giám sát.
- Phần mềm phải có khả năng cấu hình mức độ và chi tiết của nhật ký để hỗ trợ các cuộc điều tra pháp y.

Patch Management

Dưới đây là nội dung chi tiết cho hai phần **Patch Management** và **Vulnerability Management** thuộc Chương 17, cùng với bộ câu hỏi trắc nghiệm ôn tập.

1. Perform Patch Management (Thực hiện Quản lý bản vá)

- **Định nghĩa:** Việc thay đổi phần mềm để sửa các lỗ hổng hoặc lỗi được thực hiện thông qua các bản vá (patches), hotfixes hoặc QFE (Quick-Fix Engineering).
- **Đóng gói bản vá:** Các bản vá thường được gom lại và phát hành theo lịch trình cố định (ví dụ: Patch Tuesday) hoặc đóng gói thành các **Service Packs** để đơn giản hóa việc cập nhật cho các cài đặt mới.
- **Thách thức chính:** Khó khăn lớn nhất là việc **kiểm thử hồi quy (regression testing)** bản vá trên tất cả các cấu hình phần mềm khác nhau để đảm bảo việc sửa lỗi này không gây ra lỗi mới ở bộ phận khác.
- **Trách nhiệm người dùng:** Vì các nhà cung cấp không thể mô phỏng mọi môi trường doanh nghiệp, người dùng cuối nên thực hiện mức kiểm thử hồi quy cuối cùng trước khi đưa bản vá vào môi trường sản xuất (production).

- **Quy trình quản lý:** Quản lý bản vá là yếu tố then chốt cho môi trường vận hành an toàn và cần được tích hợp chặt chẽ vào quy trình **quản lý thay đổi (change management)** của tổ chức.

Vulnerability Management

2. Perform Vulnerability Management (Thực hiện Quản lý lỗ hổng)

- **Hai thành phần chính:**
    1. Xác định và sửa chữa các khiếm khuyết trong các thành phần chuỗi cung ứng (như các mô-đun mã nguồn) thông qua kiểm tra, thử nghiệm và kiểm toán.
    2. Vá các lỗ hổng sau khi chúng đã được phát hiện và sản phẩm đã được chuyển giao cho khách hàng.
- **Tính tất yếu:** Việc vá lỗi là cần thiết vì phần mềm hiện nay cực kỳ phức tạp, dẫn đến việc các sai sót luôn tồn tại trong bất kỳ sản phẩm bàn giao nào.
- **Tính kỷ luật:** Quy trình vá lỗi cần sự kỷ luật cao; các bản vá phải được phát hành kịp thời và được áp dụng ngay khi nhận được để tránh các thảm họa như virus SQL Slammer (xảy ra do người dùng không chịu áp dụng bản vá dù nó đã được phát hành).
- **Các hoạt động cốt lõi:** Quản lý lỗ hổng bao gồm việc **quét (scanning)**, **theo dõi (tracking)** và **phân loại (triaging)** các lỗ hổng dựa trên mức độ quan trọng (criticality) để phản ứng dựa trên rủi ro.

Runtime Protection

Nội dung phần **Runtime Protection** (Bảo vệ trong thời gian chạy) trong Chương 17 tập trung vào các cơ chế phòng thủ hoạt động trực tiếp khi ứng dụng đang thực thi trong môi trường sản xuất. Việc triển khai các biện pháp này không chỉ là nhiệm vụ của lập trình viên mà đòi hỏi sự phối hợp chặt chẽ với các bộ phận vận hành khác nhau trong doanh nghiệp.

Dưới đây là chi tiết các giải pháp bảo vệ runtime chính:

1. Phối hợp thiết lập bảo vệ

- Việc thiết lập các cơ chế bảo vệ thời gian chạy yêu cầu sự **phối hợp với các bộ phận chức năng** của doanh nghiệp để đảm bảo các công cụ này không gây gián đoạn hoạt động bình thường.

2. Các công nghệ bảo vệ runtime phổ biến

- **Web Application Firewall (WAF):** Đây là tường lửa cấp ứng dụng giúp kiểm tra và lọc các lưu lượng HTTP/HTTPS. Việc thiết lập WAF đòi hỏi phải phối hợp với **đội ngũ mạng (network team)**.
- **Address Space Layout Randomization (ASLR):** Kỹ thuật này xáo trộn ngẫu nhiên các địa chỉ vùng nhớ để ngăn chặn kẻ tấn công dự đoán vị trí các hàm quan trọng (chống lại các cuộc tấn công tràn bộ đệm). Việc triển khai ASLR yêu cầu phối hợp với **quản trị viên hệ thống (system administrators)** của hệ điều hành.
- **Runtime Application Self-Protection (RASP):** Đây là công nghệ bảo vệ tiên tiến nhất, sử dụng các cảm biến được tích hợp vào ứng dụng để giám sát hành vi và môi trường xung quanh.
    - **Phối hợp:** Việc thiết lập và giám sát RASP yêu cầu sự hỗ trợ của **cả quản trị viên hệ thống và quản trị viên ứng dụng**.
    - **Cơ chế:** RASP có thể cảnh báo hoặc ngăn chặn ngay lập tức các hành vi bất thường hoặc đầu vào không mong muốn khi ứng dụng đang chạy thực tế.
    - **Đặc điểm:** Do làm tăng độ phức tạp đáng kể cho ứng dụng và môi trường, RASP thường chỉ được **dành riêng cho các ứng dụng cực kỳ quan trọng (critical applications)**.

Support Continuity of Operations

Phần **Support Continuity of Operations** (Hỗ trợ Duy trì Hoạt động) trong Chương 17 tập trung vào khả năng của doanh nghiệp trong việc duy trì các tiến trình kinh doanh thiết yếu khi đối mặt với sự cố và cách thức phục hồi về trạng thái bình thường,.

Dưới đây là nội dung chi tiết:

1. Bản chất của Duy trì Hoạt động (COOP)

- **Định nghĩa:** Đây là trạng thái mà một doanh nghiệp bước vào khi các sự cố tác động đến hoạt động ở một mức độ xác định hoặc lớn hơn.
- **Mục tiêu:** Cho phép doanh nghiệp tập trung nguồn lực hạn chế vào các tiến trình kinh doanh quan trọng nhất (critical business processes) trong giai đoạn hoạt động bị suy giảm.
- **Lập kế hoạch:** Khi một ứng dụng mới được đưa vào doanh nghiệp, nó phải được tích hợp vào kế hoạch COOP. Kế hoạch này chi tiết hóa những gì sẽ được chạy (thiết yếu, bán thiết yếu hoặc không chạy) và các phụ thuộc liên quan.

2. Sao lưu, Lưu trữ và Giữ lại (Backup, Archiving, Retention)

- **Tầm quan trọng:** Sao lưu là nền tảng của một chương trình bảo mật. Cần duy trì các bản lưu trữ (archives) của các phiên bản phần mềm cũ và các bộ dữ liệu đi kèm để có thể khôi phục hệ thống khi cần thiết.
- **Bảo mật lưu trữ:** Các tập tin lưu trữ cần được **mã hóa** để bảo vệ tính nhạy cảm của dữ liệu.
- **Retention cycle (Chu kỳ giữ lại):** Được định nghĩa là một bộ sao lưu đầy đủ cần thiết để khôi phục dữ liệu (có thể là một bản full backup hoặc full backup cộng với các bản sao lưu tăng dần - incrementals).

3. Phục hồi sau thảm họa (Disaster Recovery - DR)

- **Ba trạng thái vận hành:**
    1. **Vận hành bình thường:** Trạng thái hoạt động hàng ngày.
    2. **Vận hành bị suy giảm:** Trạng thái COOP, chỉ chạy những thứ thiết yếu.
    3. **Vận hành phục hồi sau thảm họa:** Là lộ trình chuyển đổi từ trạng thái suy giảm quay trở lại trạng thái bình thường.
- **Đặc điểm phục hồi:** Khác với COOP (thường là một sự chuyển dịch đột ngột khi sự cố xảy ra), DR là một nỗ lực có kế hoạch, được lập lịch trình tỉ mỉ và kiểm thử

từng bước để đảm bảo mọi thứ hoạt động ổn định trước khi chính thức hoạt động bình thường trở lại.

4. Tính đàn hồi (Resiliency)

- **Định nghĩa:** Là thước đo khả năng một hệ thống phản ứng với sự xáo trộn và sau đó quay trở lại trạng thái vận hành chính xác.
- **Cơ chế hỗ trợ:** Bao gồm việc dư thừa vận hành (operational redundancy) cho ứng dụng và các thành phần phụ thuộc (cơ sở dữ liệu, máy chủ hỗ trợ).
- **Erasure code:** Một phương pháp bảo vệ dữ liệu trong đó dữ liệu được chia nhỏ thành các phân đoạn dư thừa để việc mất dữ liệu hoặc lỗi giao tiếp không làm dừng quá trình xử lý.

Integrate Service Level Objectives and Service Level Agreements

Phần **Integrate Service Level Objectives and Service Level Agreements** (Tích hợp các Mục tiêu Cấp độ Dịch vụ và Thỏa thuận Cấp độ Dịch vụ) trong Chương 17 tập trung vào việc thiết lập các tiêu chuẩn hiệu suất và cam kết pháp lý trong vận hành phần mềm.

Dưới đây là nội dung chi tiết:

1. Bản chất của SLA và SLO

- **SLA (Service Level Agreement):** Là thỏa thuận cấp độ dịch vụ nhằm thiết lập mức độ hiệu suất yêu cầu đối với một dịch vụ hợp đồng giữa khách hàng và nhà cung cấp. Một khi đã được ký kết, SLA trở thành một tài liệu có tính ràng buộc pháp lý.
- **SLO (Service Level Objective):** Một SLA bao gồm một chuỗi các mục tiêu cụ thể được gọi là các Mục tiêu Cấp độ Dịch vụ.

2. Tiêu chuẩn của một SLA tốt

Một SLA chất lượng cần thỏa mãn hai quy tắc cơ bản:

- **Tính rõ ràng:** Phải mô tả toàn bộ tập hợp các mục tiêu cấp độ sản phẩm hoặc dịch vụ một cách chi tiết để tránh gây hiểu lầm.

- **Tính đo lường được:** Phải cung cấp phương tiện rõ ràng để xác định xem một chức năng hoặc dịch vụ có đạt được mức độ hiệu suất như đã thỏa thuận hay không.

3. Thực thi và Quản lý

- **Cơ chế thực thi:** Hợp đồng là cơ chế chính để thực thi SLA. Hiệu suất phải được mô tả thông qua các **hành vi có thể quan sát được từ bên ngoài**.
- **Tiêu chí khách quan:** Vì phần mềm rất phức tạp, việc đánh giá hiệu suất thường dựa trên các tiêu chí hành vi khách quan làm "đại diện" (proxy) cho các hoạt động vô hình bên trong hệ thống.
- **Sử dụng nội bộ:** SLA cũng có thể được dùng trong nội bộ tổ chức để làm rõ các chi tiết về bảo trì, hiệu suất, tính sẵn có và nhân sự có trình độ, giúp các bộ phận đặt ra kỳ vọng phù hợp.
- **Trách nhiệm duy trì:** Tổ chức mua dịch vụ có trách nhiệm duy trì các thay đổi đối với thỏa thuận hợp đồng trong hệ thống kiểm soát thay đổi hợp đồng của mình.

Part VIII Secure Software Supply Chain

Chapter 18 Software Supply Chain Risk Management

Implement Software Supply Chain Risk Management

Nội dung phần **Implement Software Supply Chain Risk Management** (Triển khai Quản lý Rủi ro Chuỗi Cung ứng Phần mềm) trong Chương 18 tập trung vào việc thiết lập các quy trình để đảm bảo tính toàn vẹn của sản phẩm khi có sự tham gia của các bên thứ ba.

1. Mục đích của Đánh giá Rủi ro Nhà cung cấp

Mục tiêu tổng quát của hoạt động này là **xác định và duy trì một bộ kiểm soát rủi ro phù hợp** trong toàn bộ chuỗi cung ứng . Quá trình đánh giá sẽ xác định các mối đe dọa cụ thể, đánh giá khả năng xảy ra cũng như hậu quả của từng mối đe dọa đó để làm cơ sở xây dựng chiến lược tin cậy cho tổ chức.

2. 5 Danh mục Đe dọa chính theo GAO

Theo Văn phòng Trách nhiệm Chính phủ Hoa Kỳ (GAO), các mối đe dọa đối với chuỗi cung ứng bao gồm 5 nhóm có tác động khác nhau đến tính toàn vẹn của sản phẩm:

- **Cài đặt các logic độc hại** vào phần cứng hoặc phần mềm ``.
- **Cài đặt phần cứng hoặc phần mềm giả mạo** ``.
- **Sự thất bại hoặc gián đoạn** trong quá trình sản xuất hoặc phân phối các sản phẩm và dịch vụ quan trọng ``.
- **Sự phụ thuộc vào các nhà cung cấp dịch vụ độc hại hoặc không đủ năng lực** để thực hiện các dịch vụ kỹ thuật ``.
- **Sự tồn tại của các lỗ hổng không cố ý** trong phần mềm hoặc phần cứng ``.

3. Mô hình và Quy trình Quản lý Rủi ro

Quá trình đánh giá rủi ro giúp nhà quản lý triển khai các **kiểm soát chủ động (proactive) và phản ứng (reactive)** để ứng phó với các mối đe dọa, đồng thời giám sát hiệu quả của chúng ``. Một mô hình quản lý rủi ro nhà cung cấp tiêu chuẩn thường bao gồm 4 yếu tố chính:

- **Xác định (Identification):** Xác định các hạng mục có thể mang lại rủi ro cho doanh nghiệp. Trong trường hợp phần mềm, điều này bao gồm việc xem xét toàn bộ danh mục nguyên vật liệu phần mềm (**SBOM**) ``.
- **Đánh giá (Assessment):** Phân tích các mối đe dọa và lỗ hổng liên quan đến từng tài sản cũng như khả năng xảy ra của chúng ``.
- **Phản hồi (Response):** Thiết lập kế hoạch phản hồi chi tiết, phân định rõ trách nhiệm giữa nhà cung cấp và bên mua . Kế hoạch này cần nêu cụ thể quy trình bảo trì, vá lỗi và các mốc thời gian thực hiện khi phát hiện lỗ hổng.
- **Giám sát (Monitoring):** Đảm bảo rằng các điều khoản đã thỏa thuận được tuân thủ và quy trình đang hoạt động hiệu quả trong thực tế ``.

Analyze Security of Third-Party Software

Nội dung phần **Analyze Security of Third-Party Software** (Phân tích Bảo mật của Phần mềm Bên thứ ba) trong Chương 18 tập trung vào việc quản lý các rủi ro đến từ mã nguồn không được phát triển nội bộ.

1. Bản chất của Phần mềm Bên thứ ba

- **Định nghĩa:** Phần mềm bên thứ ba là bất kỳ phần mềm nào không do nhân sự nội bộ viết, bao gồm cả các **thư viện mã nguồn mở (open source libraries)**.
- **Sự phổ biến:** Phần lớn mã nguồn được phát triển ngày nay đều chứa các lời gọi thư viện bên thứ ba, vì vậy rủi ro này áp dụng cho hầu hết các doanh nghiệp.

2. Các chiến lược quản lý rủi ro

Có hai phương án chính thường được sử dụng kết hợp để quản lý lỗ hổng từ mã nguồn bên thứ ba:

- **Chương trình giám sát cập nhật:** Thiết lập quy trình để nhận thông báo về các bản cập nhật và vá lỗi của bên thứ ba, sau đó tích hợp chúng vào quy trình cập nhật nội bộ. Các lỗ hổng nghiêm trọng có thể kích hoạt việc phát hành bản vá ngay lập tức cho sản phẩm nội bộ.
- **Kiểm thử bảo mật đồng bộ:** Áp dụng cùng một chế độ kiểm thử bảo mật nghiêm ngặt cho phần mềm bên thứ ba giống như đối với mã nguồn tự viết.

3. Danh mục Nguyên vật liệu Phần mềm (SBOM)

- **Khái niệm: SBOM (Software Bill of Materials)** là danh sách liệt kê nguồn gốc (provenance) và dòng dõi (lineage) của tất cả các thành phần trong phần mềm, bao gồm cả thư viện, mã bên thứ ba và mã nội bộ.
- **Tầm quan trọng:** Việc liệt kê các phiên bản của mọi thành phần là yếu tố sống còn để xác định và khắc phục lỗ hổng.

- **Bài học thực tế:** Các sự cố như **Heartbleed**, **OpenVPN** hoặc virus **SQL Slammer** (ảnh hưởng đến các phần mềm sử dụng công cụ SQL của bên thứ ba) cho thấy rủi ro tiềm ẩn khi không nắm rõ các thành phần cấu thành bên trong một bản build.

Verify Pedigree and Provenance

Phần **Verify Pedigree and Provenance** (Xác minh Phả hệ và Nguồn gốc) trong Chương 18 tập trung vào việc đảm bảo tính xác thực và tính toàn vẹn của các thành phần phần mềm khi chúng di chuyển qua chuỗi cung ứng.

1. Nội dung chi tiết về Phả hệ và Nguồn gốc

- **Định nghĩa Phả hệ (Pedigree):** Là một danh sách liệt kê nơi phần mềm xuất phát, bao gồm cả thông tin phần mềm đó đến từ những nhà cung cấp phụ (subsuppliers) nào.
- **Định nghĩa Nguồn gốc (Provenance):** Là hồ sơ về dòng dõi và nguồn gốc của tất cả các thành phần, bao gồm thư viện và các phiên bản cụ thể.
- **Quy trình xác minh:** Các thành phần phải được xác thực và đảm bảo tính toàn vẹn tại các điểm giao tiếp (interfaces) giữa các tổ chức hoặc các cấp độ khác nhau trong chuỗi cung ứng. Việc xác thực này thường dựa trên các nguyên tắc về **chống thoái thác nguồn gốc (nonrepudiation of origin)**.
- **Thách thức về hàng giả (Counterfeiting):** Các thành phần giả mạo từ các nhà cung cấp không uy tín có thể đe dọa toàn bộ hệ thống vì chúng được thiết kế để bắt chước các bộ phận chính hãng, khiến việc phát hiện bằng các phương pháp thông thường trở nên khó khăn.

2. Các biện pháp bảo vệ chính

- **Chuyển giao an toàn (Secure Transfer):** Phần mềm là vật phẩm kỹ thuật số dễ bị đánh chặn và thao túng trong quá trình truyền tải. Cần sử dụng các kênh truyền thông được mã hóa và **ký mã nguồn (code signing)** để đảm bảo nội dung không bị thay đổi.

- **Bảo mật kho lưu trữ mã nguồn (Code Repository Security):** Kho lưu trữ cần được bảo vệ nghiêm ngặt như tài khoản ngân hàng của công ty để ngăn chặn các thay đổi mã không được phép, dù là từ bên ngoài hay nội bộ.
- **Bảo mật môi trường Build (Build Environment Security):** Môi trường xây dựng hiện đại (CI/CD) cần được bảo vệ để đảm bảo phần mềm được cấu trúc đúng cách mà không có sự can thiệp trái phép.
- **Thành phần được ký kỹ thuật số và băm mã (Digitally Signed/Hashed Components):** Đây là phương pháp mạnh mẽ để phát hiện các thay đổi so với tài liệu nguồn đã được phê duyệt.
- **Quyền được kiểm toán (Right to Audit):** Bên mua nên có quyền kiểm toán các quy trình của nhà cung cấp để xác định xem các kiểm soát bảo mật có đang hoạt động hiệu quả như thiết kế hay không.

Chapter 19 Supplier Security Requirements

Ensure Supplier Security Requirements in the Acquisition Process

Phần **Ensure Supplier Security Requirements in the Acquisition Process** (Đảm bảo các yêu cầu bảo mật của nhà cung cấp trong quy trình mua sắm) thuộc Chương 19 tập trung vào việc thiết lập và thực thi các tiêu chuẩn bảo mật đối với các bên thứ ba ngay từ khi bắt đầu mối quan hệ kinh doanh.

Dưới đây là nội dung chi tiết dựa trên tài liệu:

1. Tổng quan về Yêu cầu Bảo mật trong Mua sắm

- **Quản lý rủi ro từ bên ngoài:** Chuỗi cung ứng cung cấp các thành phần thiết yếu (phần cứng, phần mềm) nhưng cũng mang lại rủi ro mà doanh nghiệp khó kiểm soát trực tiếp.

- **Định nghĩa rủi ro:** Bước then chốt là xác định loại rủi ro và mức độ rủi ro mà doanh nghiệp sẵn sàng chấp nhận. Các yêu cầu bảo mật phải được định nghĩa một cách **đo lường được** và đưa vào quy trình thẩm định nhà cung cấp.
- **Chuyển tiếp yêu cầu (Flow Down):** Các yêu cầu bảo mật không chỉ dừng lại ở nhà cung cấp trực tiếp mà phải được yêu cầu chuyển tiếp xuống các nhà thầu phụ (subcontractors) của họ để đảm bảo không có lỗ hổng trong toàn bộ chuỗi.

2. Các hoạt động cốt lõi

- **Lựa chọn nguồn cung (Supplier Sourcing):** Phải phân tích mọi khía cạnh hoạt động của nhà cung cấp, bao gồm cả mức độ ảnh hưởng hoặc kiểm soát của nước ngoài đối với sản phẩm. Việc lựa chọn thường dựa trên điểm số **Rủi ro so với Lợi nhuận**. Chi tiết bên dưới
- **Chuyển tiếp nhà cung cấp (Supplier Transitioning):** Cần có một kế hoạch chuyển tiếp chi tiết trong hợp đồng để đảm bảo tính toàn vẹn của thành phần tại các điểm giao tiếp giữa các tổ chức. Kế hoạch này cũng phải giải quyết các quyền sở hữu trí tuệ (IP) để tránh vi phạm bản quyền hoặc mất mát tài sản trí tuệ.
- **Kiểm toán tuân thủ (Audit of Security Policy Compliance):** Doanh nghiệp cần hiểu rõ các thực hành phát triển phần mềm an toàn của nhà cung cấp. Quy trình phát triển của nhà cung cấp là một yếu tố quan trọng có thể kiểm toán được để xác minh mức độ bảo mật.
- **Thông báo và Phản ứng Sự cố:** Doanh nghiệp phụ thuộc vào hệ thống phản ứng sự cố và quản lý lỗ hổng của nhà cung cấp. Các điều khoản về cách thức thông báo, điều phối và báo cáo khi có lỗi xảy ra phải được quy định rõ trong hợp đồng.
- **Cơ cấu Bảo trì và Hỗ trợ:** Cần xem xét mô hình cấp phép (thương mại hay cộng đồng), quyền chỉnh sửa mã nguồn và cam kết hỗ trợ dài hạn của nhà cung cấp.
- **Hồ sơ năng lực bảo mật (Security Track Record):** Đánh giá cách nhà cung cấp xử lý các sự cố trong quá khứ và liệu có các phát hiện kiểm toán lặp lại hay không.

Supplier Sourcing

Nội dung phần **Supplier Sourcing** (Lựa chọn nguồn cung ứng từ nhà cung cấp) trong Chương 19 tập trung vào việc thẩm định kỹ lưỡng các nhà cung cấp trước khi ký kết hợp đồng để đảm bảo tính tin cậy cho chuỗi cung ứng phần mềm.

Dưới đây là chi tiết các yếu tố then chốt:

1. Bản chất của việc lựa chọn nguồn cung

- **Thẩm định toàn diện:** Lựa chọn nguồn cung thực chất là quá trình thực hiện tất cả các phân tích cần thiết để hiểu rõ mọi khía cạnh trong hoạt động của nhà cung cấp trước khi đưa ra quyết định thực tế.
- **Xác định phạm vi công việc:** Bên mua cần hiểu rõ những thành phần công việc nào nhà thầu phụ (subcontractor) được phép thực hiện và thành phần nào không được phép. Việc phân bổ và quản lý công việc này phải được quy định rõ ràng trong hợp đồng.

2. Mối quan ngại về ảnh hưởng nước ngoài

- **Ảnh hưởng và kiểm soát:** Một yếu tố an ninh chiến lược là xác định mức độ ảnh hưởng, kiểm soát hoặc quyền sở hữu của nước ngoài (FOCI) đối với nhà cung cấp.
- **Tính tin cậy:** Sự kiểm soát từ các tổ chức hoặc quốc gia không thân thiện có thể ảnh hưởng trực tiếp đến mức độ tin cậy của sản phẩm phần mềm. Vì phần mềm hiện nay được tích hợp từ nhiều nguồn (thư viện, mô-đun) từ cả trong và ngoài nước, việc xác minh nguồn gốc là cực kỳ quan trọng.

3. Mô hình đánh giá Rủi ro so với Lợi nhuận

Các doanh nghiệp thường xếp hạng các nhà cung cấp dựa trên điểm số **Rủi ro so với Lợi nhuận (Risk versus Return)**. Các dự án đáp ứng hoặc vượt mong đợi về mặt kỹ thuật, đóng góp cho mục tiêu kinh doanh và phù hợp với nguồn lực sẽ nhận điểm cao hơn.

Các yếu tố đánh đổi (trade-offs) về bảo mật cần cân nhắc bao gồm:

- **Cải tiến chiến lược so với duy trì hoạt động hiện tại:** Đôi khi việc nhà cung cấp cố gắng hiện đại hóa hoạt động lại làm tăng nguy cơ bị xâm nhập bảo mật. Mọi sự thay đổi so với cấu hình đang hoạt động ổn định cần phải được chứng minh là có lợi về mặt an ninh.
- **Rủi ro cao so với rủi ro thấp:** Các phương tiếp cận rủi ro cao có thể mang lại lợi nhuận cao nhưng chỉ khi rủi ro đó có khả năng quản lý được. Ban quản lý phải cân bằng giữa mức rủi ro họ có thể chấp nhận với khả năng kiểm soát rủi ro đó.
- **Tác động giữa các nhà cung cấp:** Một nhà cung cấp mới có thể ảnh hưởng đến các nhà cung cấp hiện có trong chuỗi. Việc thêm một nhà cung cấp ở cấp cao hơn có thể gây ra những "gợn sóng" ảnh hưởng tiêu cực đến tính an toàn và bảo mật của toàn bộ cấu trúc hạ nguồn.
- **Chi phí cơ hội:** Cần xem xét liệu việc đầu tư lớn vào một nhà cung cấp hiện tại có làm mất đi hoặc trì hoãn những cơ hội tốt hơn trong tương lai hay không.

4. Kiểm soát tính toàn vẹn của hợp đồng

Sau khi lựa chọn được nhà cung cấp, các tiêu chí đánh giá phải được đưa vào hợp đồng để đảm bảo các hoạt động kiểm tra, thử nghiệm và kiểm toán bảo mật được thực hiện đúng như cam kết. Hợp đồng cũng cần quy định quy trình xử lý khi phát hiện các điểm không tuân thủ (nonconcurrence) và các hành động khắc phục tương ứng.

Support Contractual Requirements

Phần **Support Contractual Requirements** (Hỗ trợ các yêu cầu hợp đồng) trong Chương 19 tập trung vào việc thiết lập các điều khoản pháp lý để bảo vệ quyền lợi và phân định trách nhiệm rủi ro giữa bên mua và nhà cung cấp. Hợp đồng là công cụ cuối cùng để kiểm soát mọi khía cạnh của quá trình phát triển và bàn giao sản phẩm trong chuỗi cung ứng.

Nội dung chi tiết về các Yêu cầu Hợp đồng

- **Phạm vi của Hợp đồng:** Các hợp đồng cung cấp phần mềm cần chi tiết hóa một loạt các vấn đề bao gồm quyền sở hữu trí tuệ (IP), ký quỹ mã nguồn (code escrow), trách nhiệm pháp lý (liability), bảo hành (warranty), thỏa thuận cấp phép người dùng

cuối (EULA) và các thỏa thuận cấp độ dịch vụ (SLA). Các điều khoản này quyết định cách sản phẩm được sử dụng và ai chịu trách nhiệm về rủi ro.

- **Quyền sở hữu trí tuệ (Intellectual Property - IP):**
    - Phần mềm là tài sản trí tuệ vô hình, dễ bị đánh cắp hoặc vi phạm bản quyền khi di chuyển qua các mắt xích của chuỗi cung ứng.
    - Thông thường, **nhà cung cấp sẽ giữ quyền sở hữu IP** và chỉ chuyển giao **giấy phép sử dụng (license)** cho bên mua với các điều kiện cụ thể.
    - Việc thực hiện các cuộc kiểm toán và kiểm tra nghiêm ngặt là phương pháp tốt nhất để ngăn ngừa hành vi trộm cắp IP trong quá trình tích hợp hệ thống.
- **Ký quỹ mã nguồn (Code Escrow):** Đây là thỏa thuận mà trong đó một bản sao mã nguồn được giao cho một bên thứ ba trung gian (đại lý ký quỹ) nắm giữ. Điều này bảo vệ bên mua trong trường hợp **nhà cung cấp phá sản** hoặc ngừng hỗ trợ dòng sản phẩm, giúp bên mua vẫn có thể tiếp cận mã nguồn để duy trì hoạt động kinh doanh.
- **Trách nhiệm pháp lý và Bảo hành:** Trong hầu hết các trường hợp, **rủi ro thường được chuyển sang người dùng cuối** vì nhà cung cấp có khả năng hạn chế trong việc quản lý bảo hành khi mã nguồn của họ đã được nhúng sâu vào hệ thống khác. Hợp đồng phải xác định rõ mức độ trách nhiệm chia sẻ và quyền được vá lỗi (patch) mã nguồn của bên thứ ba.
- **Tuân thủ pháp luật (Legal Compliance):** Mọi thành phần trong chuỗi cung ứng phải tuân thủ các luật và quy định hiện hành (chính phủ, ngành, hợp đồng). Thách thức chính là việc phát triển các **bằng chứng khách quan, có thể kiểm toán được** để chứng minh rằng các yêu cầu tuân thủ đã được đáp ứng và báo cáo chúng cho các cơ quan quản lý phù hợp.