

**Kiểm tra/nhắc lại bài học trước**

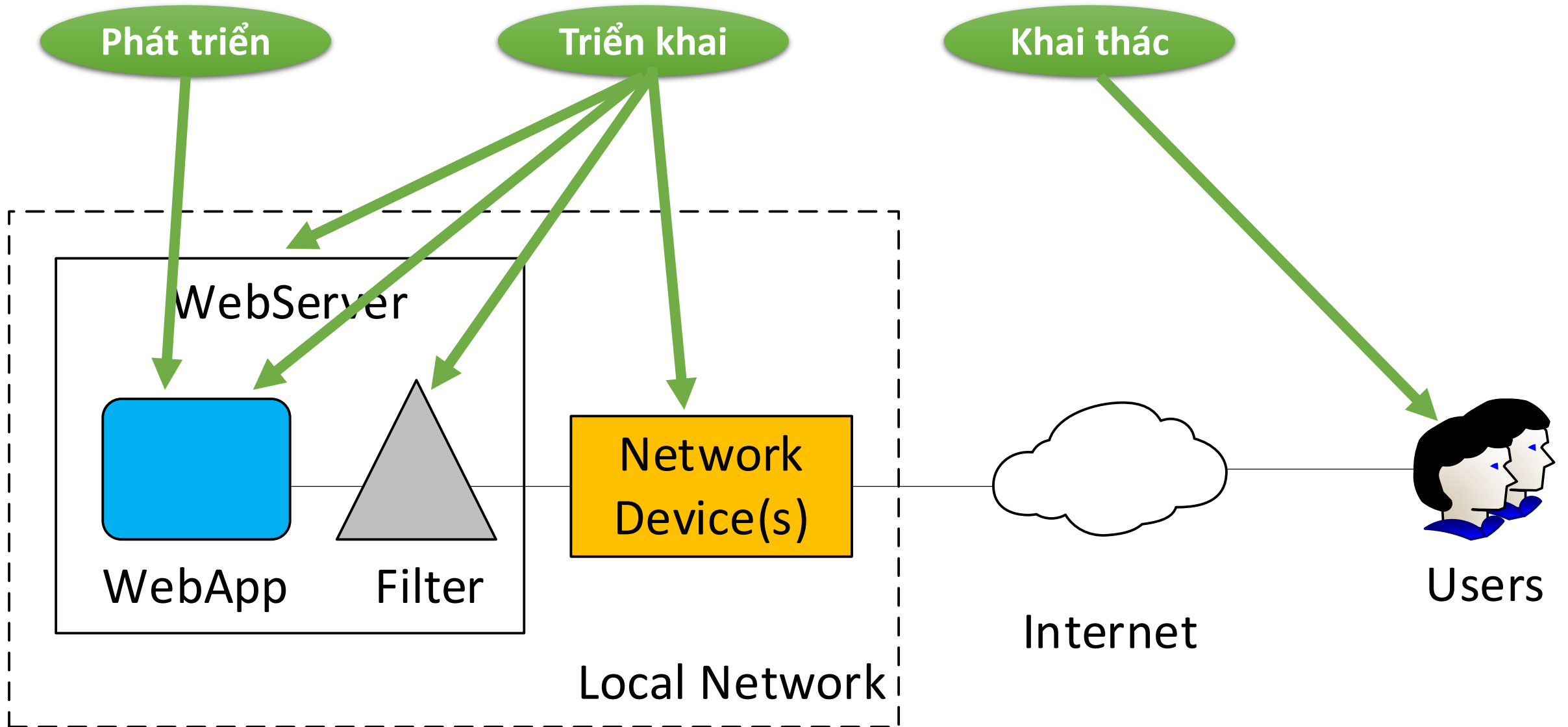
# CÔNG NGHỆ WEB AN TOÀN

Bài 4-2. Phát triển và triển khai ứng dụng web an toàn

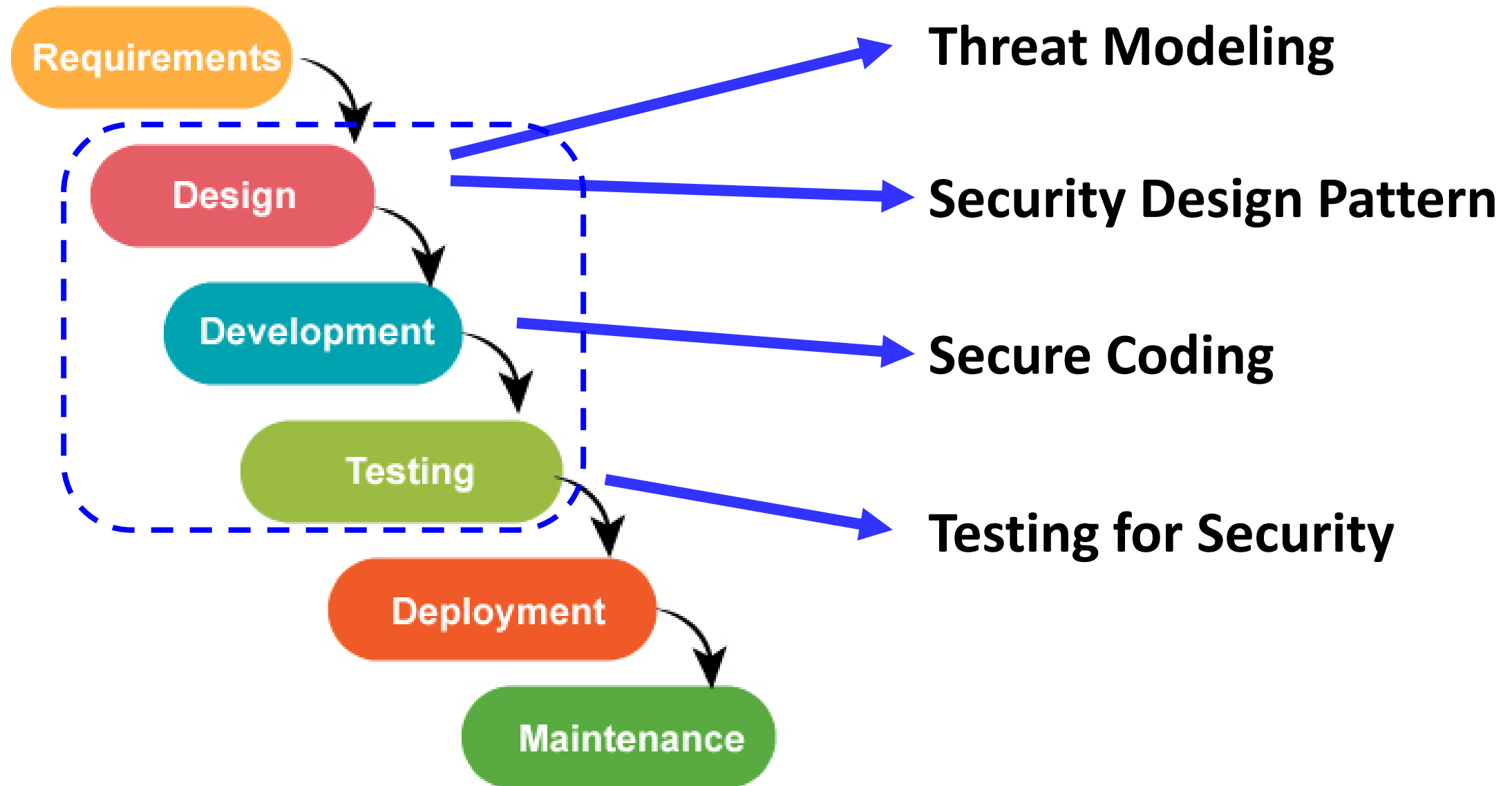
**ĐẶT VẤN ĐỀ:**

**“CẦN CHỐNG LẠI CÁC HIỂM HỌA AN TOÀN ĐỐI  
VỚI ỨNG DỤNG WEB THẾ NÀO?”**

# Giải pháp an toàn cho ứng dụng web



# Phát triển phần mềm (Waterfall model)



1

Mô hình hóa hiểm họa

2

Mẫu thiết kế an toàn

3

Lập trình an toàn

4

Kiểm thử an toàn

5

Triển khai an toàn

# Mục tiêu buổi học

---

1. Trình bày được tổng quan giải pháp đảm bảo an toàn cho ứng dụng web
2. Trình bày được bản chất, ý nghĩa của các giải pháp đảm bảo an toàn trong quá trình phát triển, triển khai ứng dụng web

## Tài liệu tham khảo

1. Nguyễn Tuấn Anh, Hoàng Thanh Nam, “Xây dựng ứng dụng web an toàn” (Chương 3), Học viện KTMM, 2013
2. Rolf Oppliger, “Security Technologies for the World Wide Web” (2-e), Artech House, 2002
3. Andrew Hoffman, “Web Application Security: Exploitation and Countermeasures for Modern Web Applications”, O’Reilly, 2020
4. Eduardo B. Fernandez, “Security Patterns in Practice: Designing Secure Architectures Using Software Patterns”, Wiley, 2013
5. Adam Shostack, “Threat Modeling: Designing for Security”, Wiley, 2014



1

**Mô hình hóa hiểm họa**

2

Mẫu thiết kế an toàn

3

Lập trình an toàn

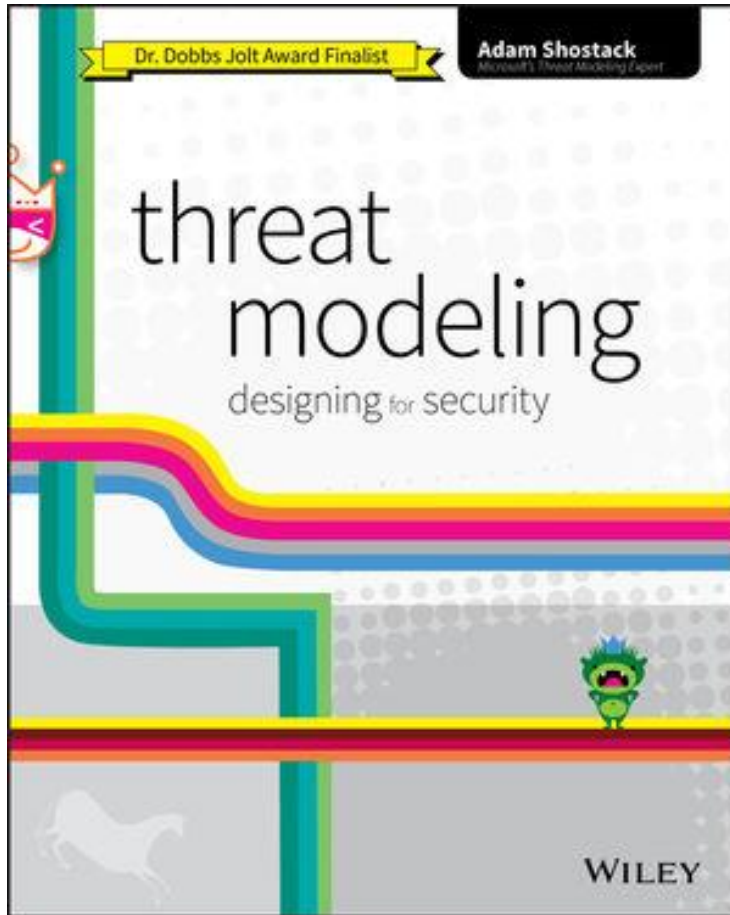
4

Kiểm thử an toàn

5

Triển khai an toàn

# Tài liệu tham khảo



<https://www.youtube.com/watch?v=7jB5OS6mepU>

<https://www.youtube.com/watch?v=nd02oPnMdR4>

# Tài liệu tham khảo bổ sung

---

1. Victoria Drake (at **OWASP**), **Threat Modeling**,  
[https://owasp.org/www-community/Threat\\_Modeling](https://owasp.org/www-community/Threat_Modeling)
2. Tomasz Andrzej Nidecki (at **Acunetix**), **Threat modeling for web application security**,  
<https://www.acunetix.com/blog/web-security-zone/threat-modeling-web-application-security/>
3. Ramya Mohanakrishnan (at **Spiceworks**), **Top 10 Threat Modeling Tools in 2021**,  
<https://www.spiceworks.com/it-security/vulnerability-management/articles/top-threat-modeling-tools/>

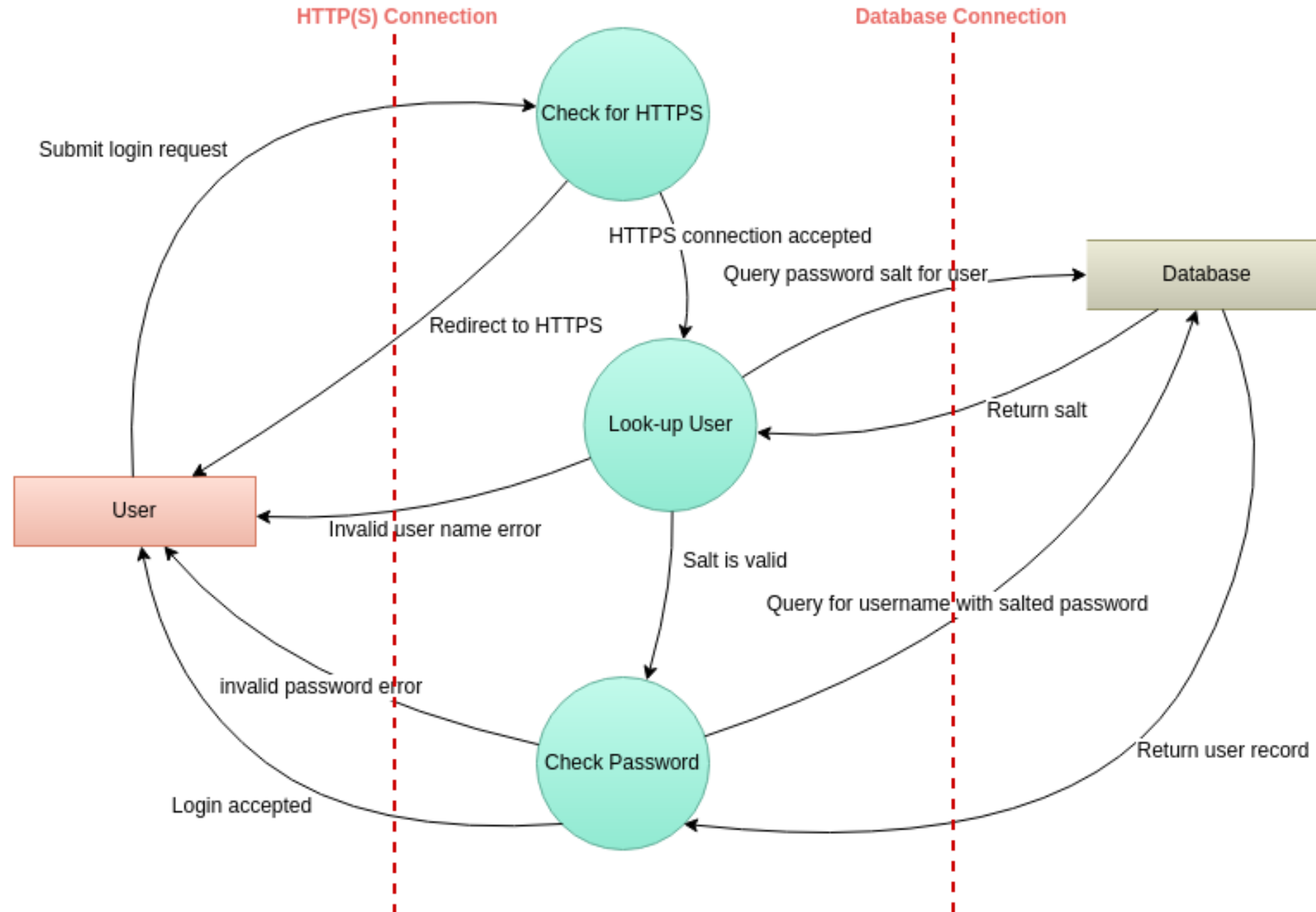
# Threat Model

- A **threat model** is a structured representation of all the information that affects the security of an application. In essence, it is a view of the application and its environment through the lens of security.
- **Mô hình hiểm họa** là một biểu diễn có cấu trúc của tất cả các thông tin mà có ảnh hưởng đến tính an toàn của một ứng dụng. Đó là hình ảnh về ứng dụng và môi trường của nó qua con mắt của chuyên gia an toàn thông tin.

*[OWASP]*

# Threat Modeling: What

- Mô hình hóa hiểm họa là việc sử dụng mô hình để xác định xác hiểm họa và cách thức xử lý các hiểm họa đó.



# Threat Modeling: Why

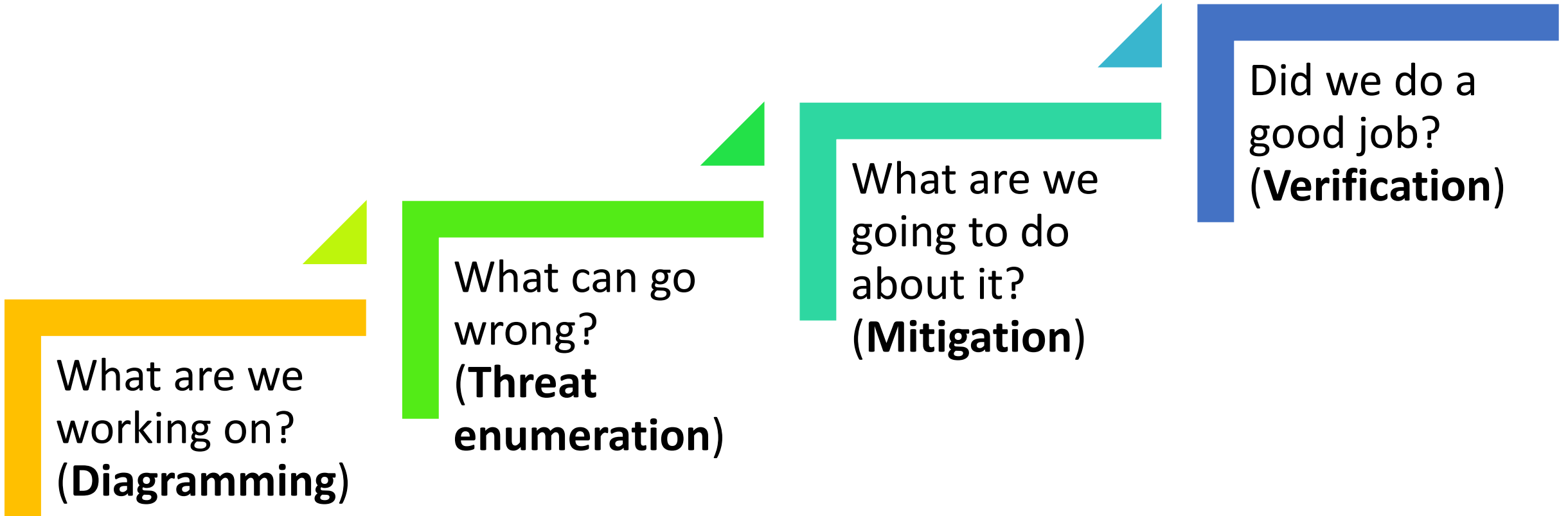
- **You model because** it enables you to find issues in things you haven't built yet, and because it enables you to catch a problem before it starts.

*[Shostack, p.3]*

- **It's very important because** it makes you look at security risks top-down, focus on decision-making and prioritize cybersecurity decisions, and consider how you can use your resources in the best possible way.

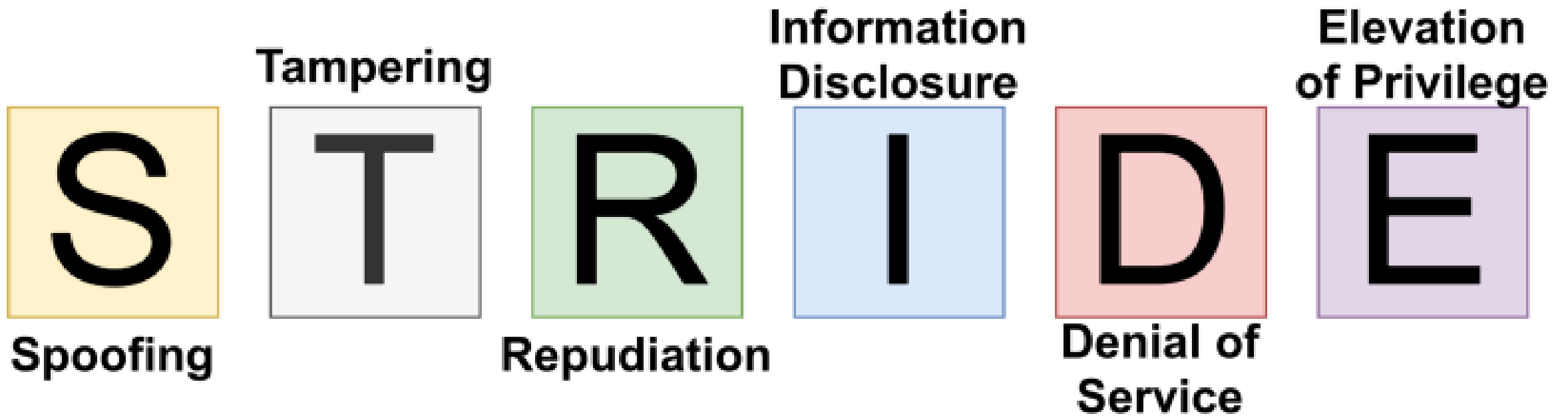
*[Acunetix]*

# 4 stages of threat modeling



*[Acunetix]*

# Threat Enumeration



STRIDE là một mô hình hiểm họa, được sử dụng để xác định các hiểm họa đối với một hệ thống.

*[Wikipedia]*



# Threat Modeling Tools

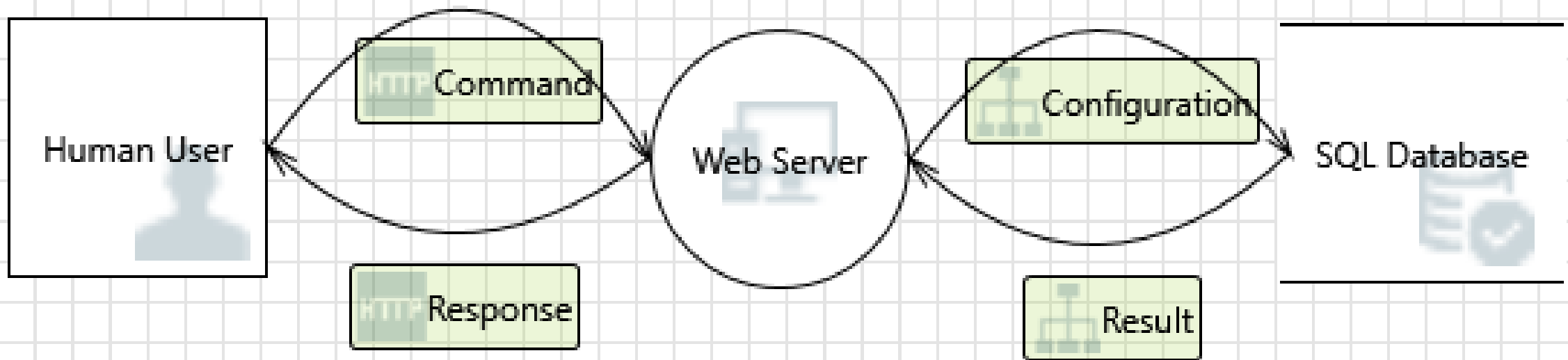
- Cairis [open-source]
- IriusRisk [paid, free]
- Microsoft Threat Modeling Tool [open-source]
- OWASP Threat Dragon [open-source]
- SecuriCAD by Foreseeti [paid, free]
- ...(and many others)...



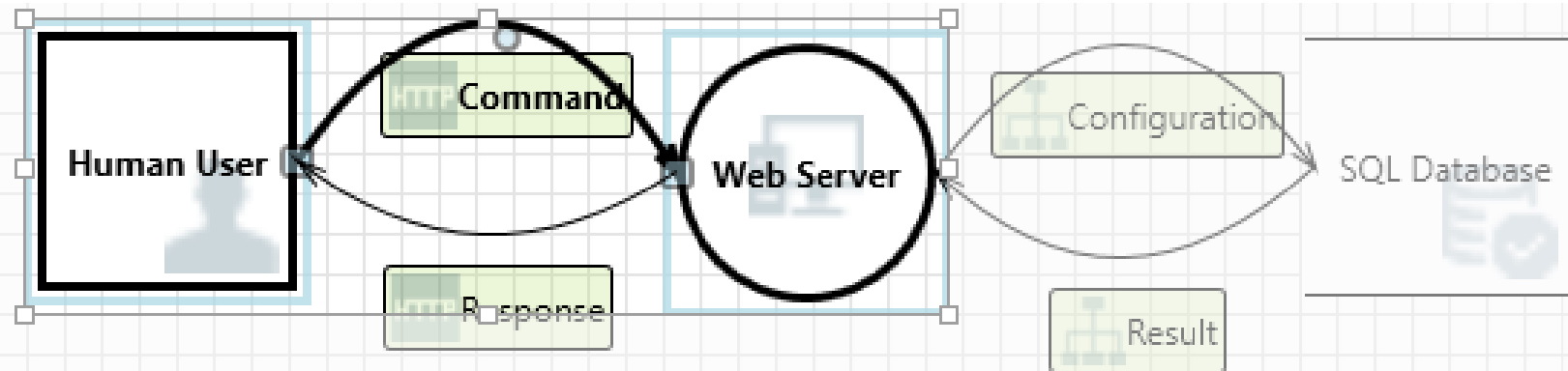
## Ví dụ

- **Getting started with the Microsoft Threat Modeling Tool**

<https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-getting-started>



# Ví dụ thông tin về một threat



Threat Properties	
ID: 13	Diagram: WebApp
Status:	Mitigated
Last Modified:	DESKTOP-LBLUQ39\Nguyen Tuan
Title:	Spoofing the Human User External Entity
Category:	Spoofing
Description:	Human User may be spoofed by an attacker and this may lead to unauthorized access to Web Server. Consider using a standard authentication mechanism to identify the external entity.
Justification:	
Interaction:	Command
Priority:	High

**LIVE DEMO**

**Kết quả của việc threat modeling là danh mục các  
hiểm họa và cách thức xử lý các hiểm họa**

1

Mô hình hóa hiểm họa

2

**Mẫu thiết kế an toàn**

3

Lập trình an toàn

4

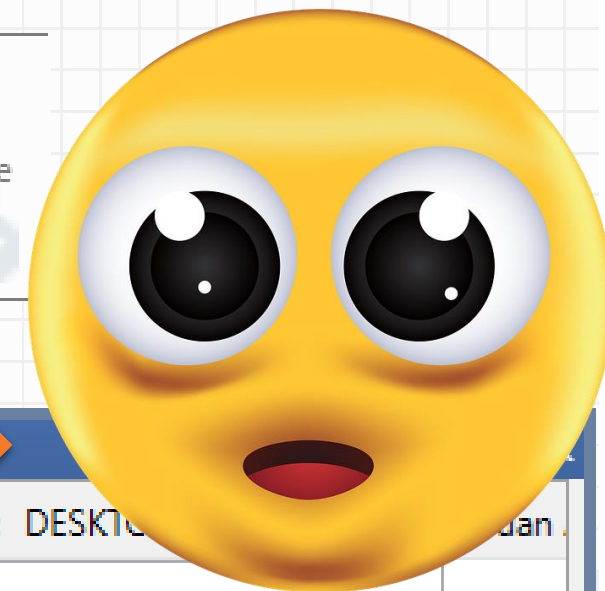
Kiểm thử an toàn

5

Triển khai an toàn

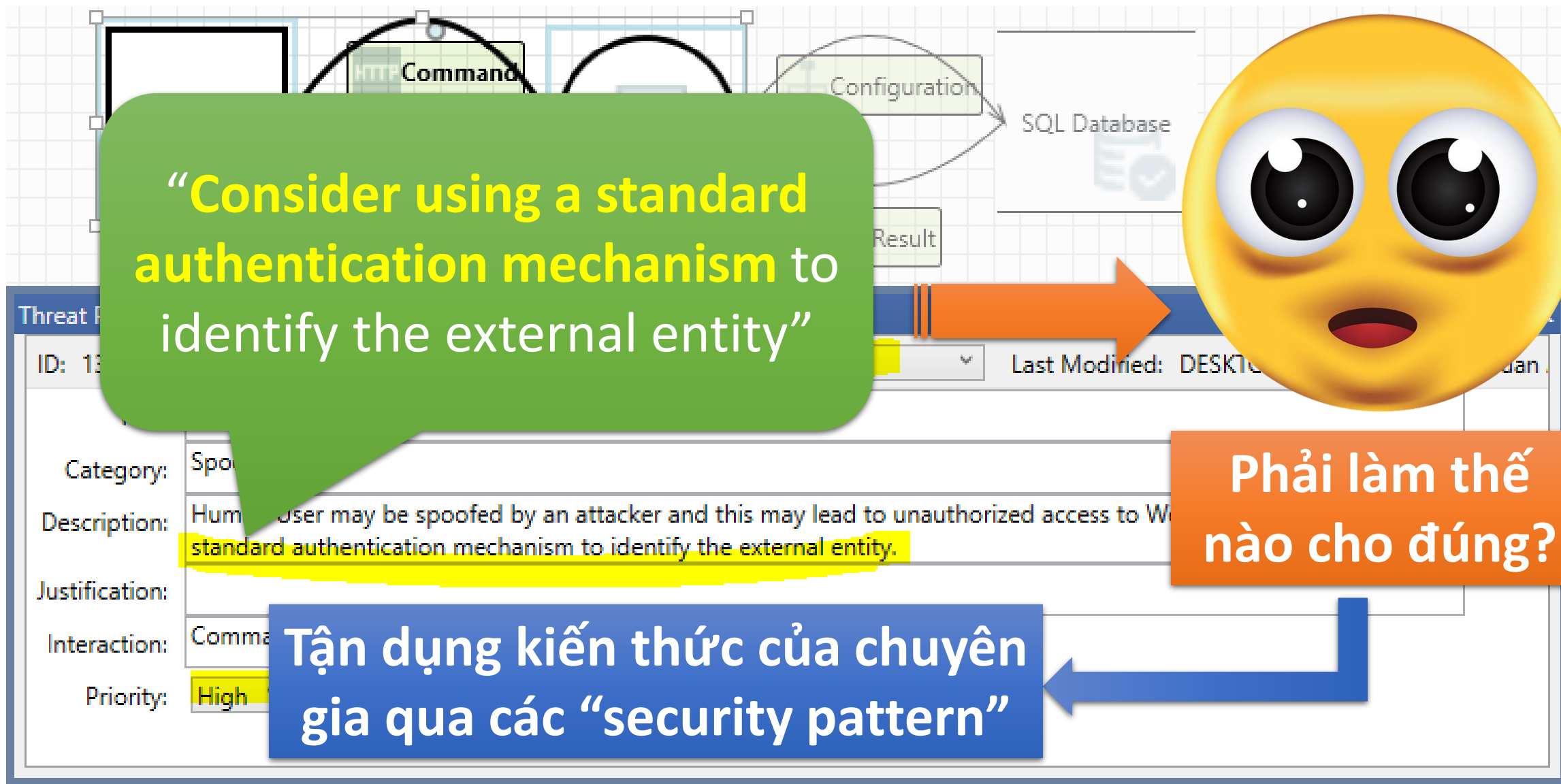
# Hãy làm...! Làm thế nào?

**“Consider using a standard authentication mechanism to identify the external entity”**

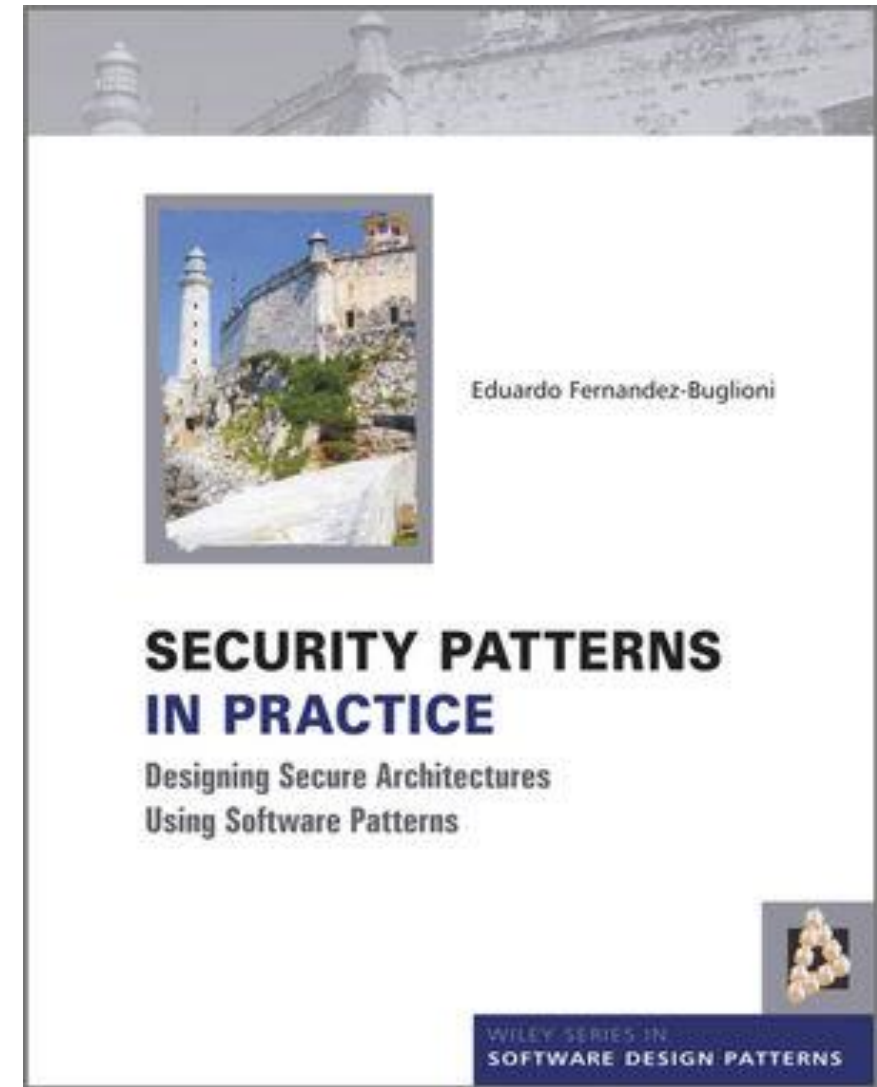
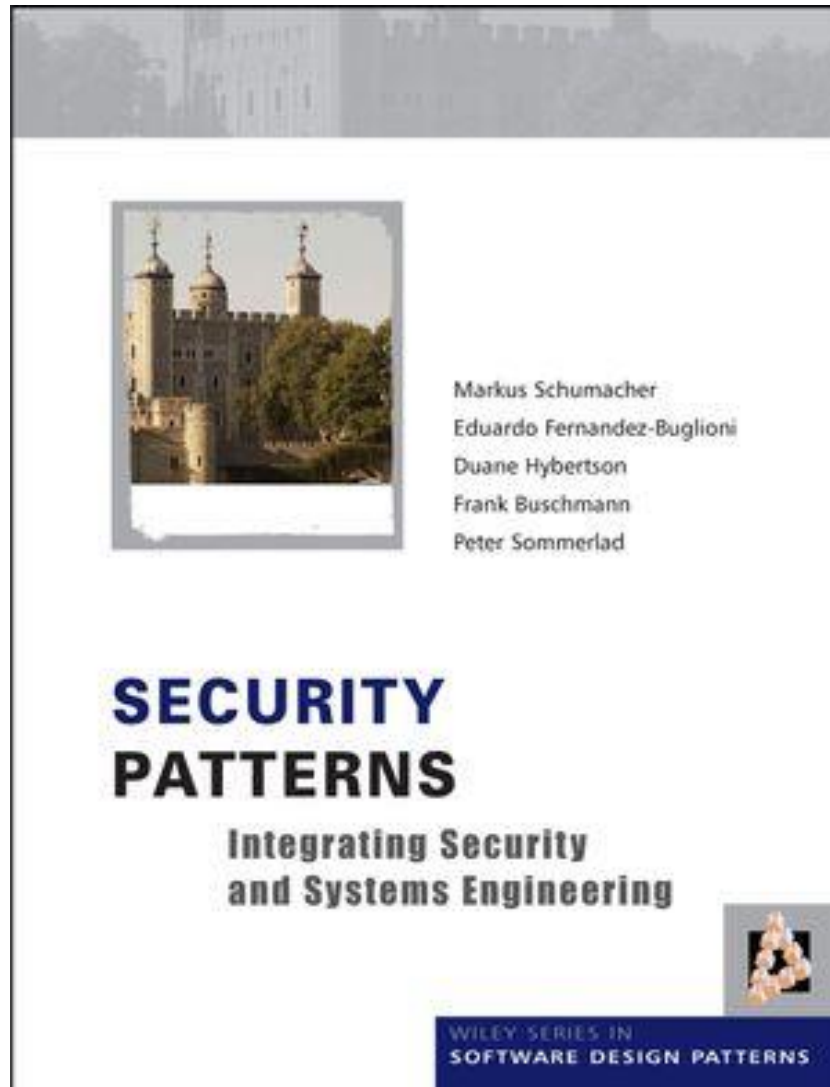


**Phải làm thế nào cho đúng?**

**Tận dụng kiến thức của chuyên gia qua các “security pattern”**



# Tài liệu tham khảo





# Tài liệu tham khảo bổ sung

---

1. Open Group (2004), **Security Design Patterns: Technical Guide**,  
<https://pubs.opengroup.org/onlinepubs/9299969899/toc.pdf>
2. Ronald Wassermann and Betty H.C. Cheng, **Security Patterns**,  
<https://www.cse.msu.edu/~cse870/Materials/security-patterns.pdf>
3. GIAC (2004), **An Introduction to Security Design Patterns**,  
<https://www.giac.org/paper/gsec/3689/design-distance-introduction-security-design-patterns/105960>

**“Each **pattern** describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice”**

*Christopher Alexander*

# Mẫu thiết kế (Design Pattern)

- **Mẫu thiết kế (design pattern)** là một **giải pháp tổng thể cho các vấn đề chung** trong thiết kế phần mềm.
- Một mẫu thiết kế không phải là một thiết kế hoàn thiện để có thể được chuyển đổi trực tiếp thành mã; nó chỉ là một mô tả hay là sườn (template) mô tả cách giải quyết một vấn đề mà có thể được dùng trong nhiều tình huống khác nhau.
- Giải thuật không được xem là mẫu thiết kế, vì chúng giải quyết các vấn đề về tính toán hơn là các vấn đề về thiết kế.

# Security Pattern

- ❑ A **security pattern** describes a solution to the problem of controlling (stopping or mitigating) a set of specific threats through some security mechanisms.
- ❑ **Mẫu an toàn** là một mẫu mô tả cách thức ngăn ngừa một hiểm họa bằng việc sử dụng các cơ chế an toàn.
- ❑ Ví dụ: RBAC là một security pattern

# Cấu trúc của một security pattern

- Tình huống (Context)
- Vấn đề (Problem)
- Giải pháp (Solution)
- Cấu trúc (Structure)
- Hoạt động (Dynamics)
- Cài đặt (Implementation)
- ... (và các thành phần khác)...

# Cấu trúc của một security pattern

- **Tình huống (Context)**
- Vấn đề (Problem)
- Giải pháp (Solution)
- Cấu trúc (Structure)
- Hoạt động (Dynamics)
- Cài đặt (Implementation)
- ... (và các thành phần khác)...

Mô tả về (một phần của) hệ thống thông tin mà ở đó Security Pattern có thể được sử dụng

# Cấu trúc của một security pattern

- Tình huống (Context)
- **Vấn đề (Problem)**
- Giải pháp (Solution)
- Cấu trúc (Structure)
- Hoạt động (Dynamics)
- Cài đặt (Implementation)
- ... (và các thành phần khác)...

- Mô tả về hậu quả nếu không có một giải pháp tốt (Risk)
- Mô tả các yêu cầu, các ràng buộc cần được thỏa mãn (hiệu năng, chức năng...)

# Cấu trúc của một security pattern

- Tình huống (Context)
- Vấn đề (Problem)
- **Giải pháp (Solution)**
- Cấu trúc (Structure)
- Hoạt động (Dynamics)
- Cài đặt (Implementation)
- ... (và các thành phần khác)...

Mô tả ý tưởng (chung nhất) giải pháp để giải quyết vấn đề



# Cấu trúc của một security pattern

- Tình huống (Context)
- Vấn đề (Problem)
- Giải pháp (Solution)
- **Cấu trúc (Structure)**
- Hoạt động (Dynamics)
- Cài đặt (Implementation)
- ... (và các thành phần khác)...

Mô tả các thành phần của giải pháp đề xuất  
(Có thể sử dụng class-diagram)

# Cấu trúc của một security pattern

- Tình huống (Context)
- Vấn đề (Problem)
- Giải pháp (Solution)
- Cấu trúc (Structure)
- **Hoạt động (Dynamics)**
- Cài đặt (Implementation)
- ... (và các thành phần khác)...

Mô tả hoạt động của giải pháp  
(Có thể sử dụng sequence-diagram)

# Cấu trúc của một security pattern

- Tình huống (Context)
- Vấn đề (Problem)
- Giải pháp (Solution)
- Cấu trúc (Structure)
- Hoạt động (Dynamics)
- **Cài đặt (Implementation)**
- ... (và các thành phần khác)...

Khuyến cáo về cách thức cài đặt giải pháp trong sản phẩm cụ thể

# Cấu trúc của một security pattern

- Tình huống (Context)
- Vấn đề (Problem)
- Giải pháp (Solution)
- Cấu trúc (Structure)
- Hoạt động (Dynamics)
- Cài đặt (Implementation)
- ... (và các thành phần khác)...

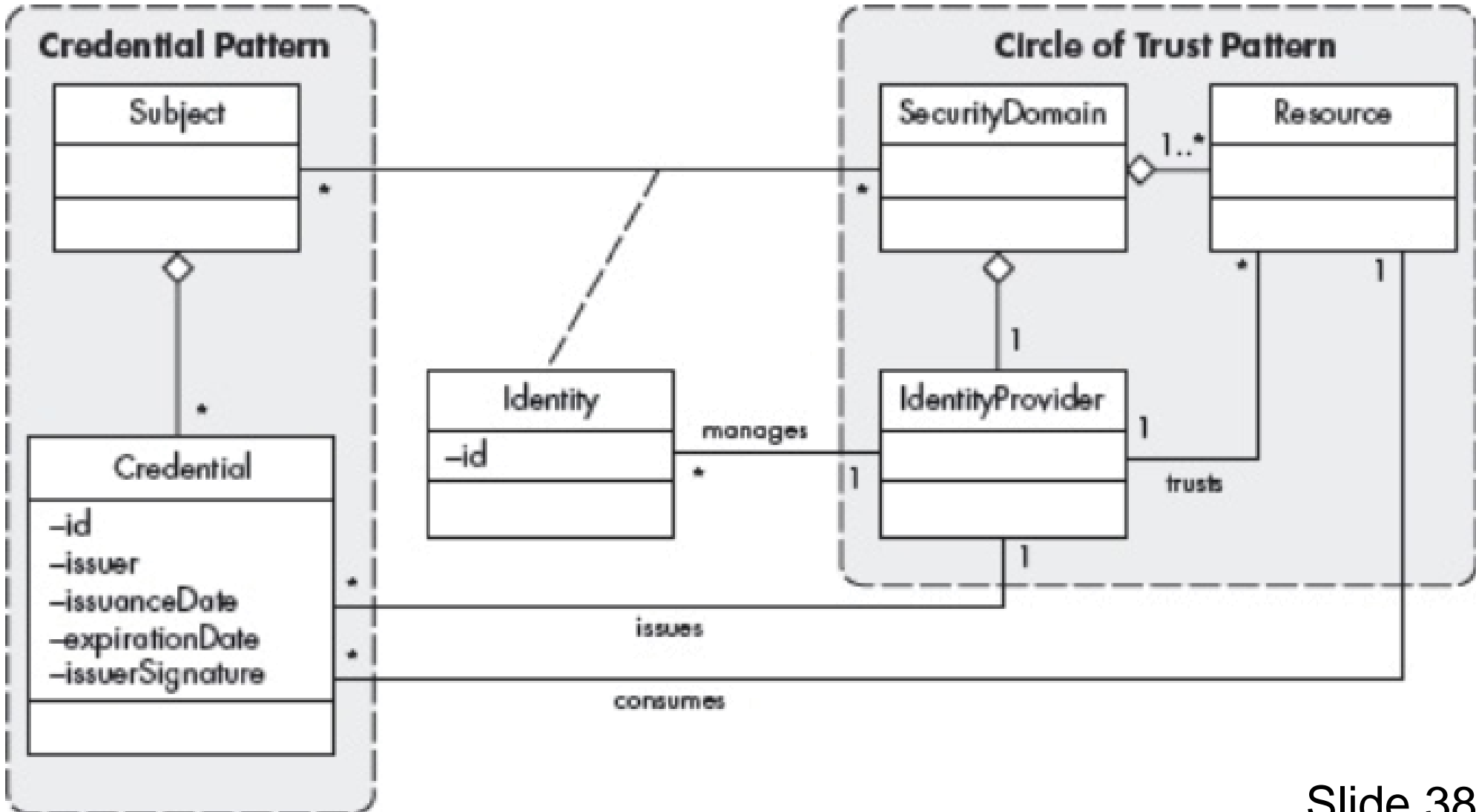
- Example Resolved
- Consequences
- Known Uses
- See Also (Related Patterns)

# IDENTITY PROVIDER

- **Context:** một tổ chức cung cấp nhiều dịch vụ, ứng dụng khác nhau cho người dùng
- **Problem:** nếu mỗi ứng dụng, dịch vụ thực hiện quản lý định danh một cách riêng rẽ thì gây ra khó khăn cho việc quản lý (người dùng có nhiều định danh; tổ chức khó đảm bảo tính nhất quán thông tin về người dùng...)
- **Solution:** thiết lập một IDENTITY PROVIDER để tập trung quản lý thông tin định danh cho người dùng của tổ chức.

## IDENTITY PROVIDER (tiếp)

- **Structure:**





**Các framwork (Spring, Laravel, ASP .NET...) đã tích hợp hỗ trợ nhiều security pattern khác nhau!**

1

Mô hình hóa hiểm họa

2

Mẫu thiết kế an toàn

3

**Lập trình an toàn**

4

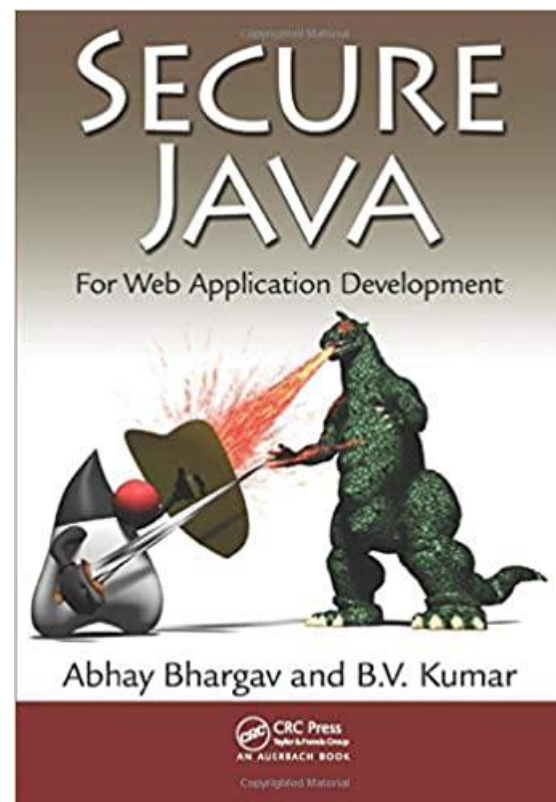
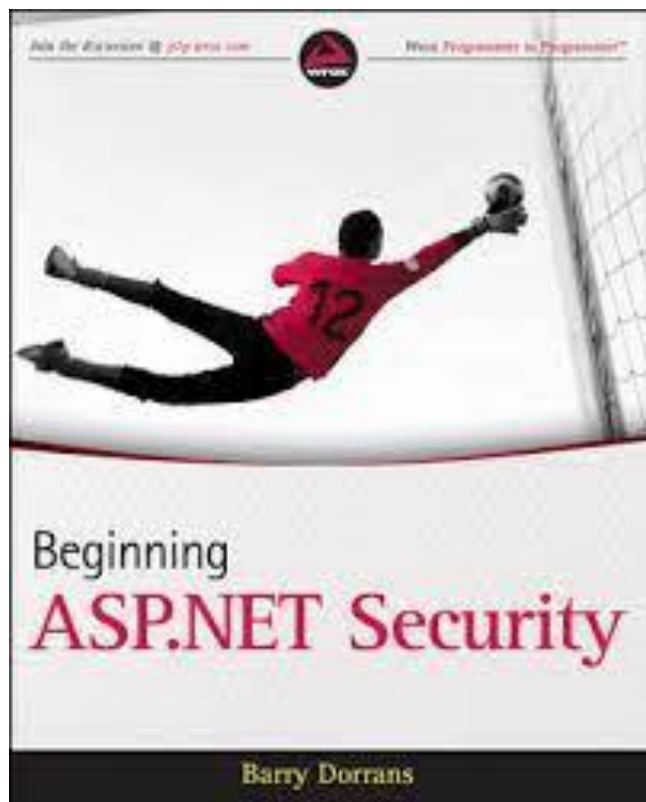
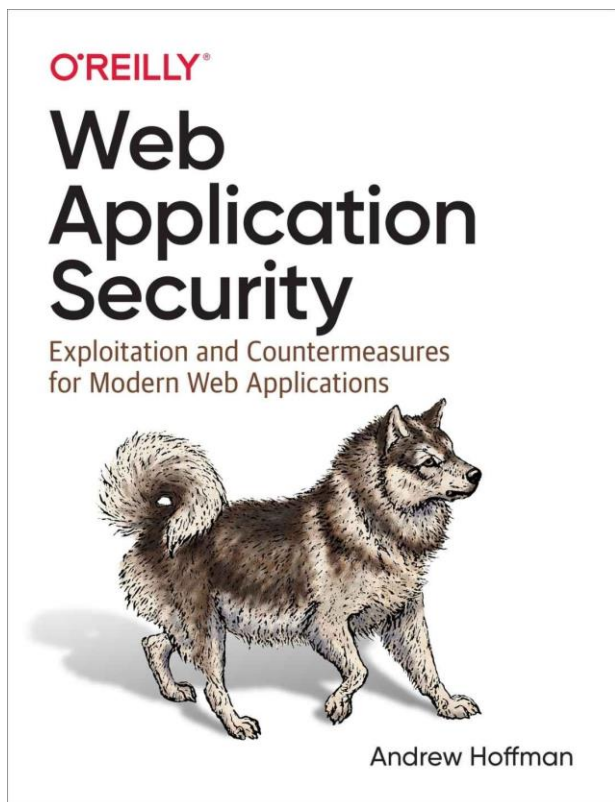
Kiểm thử an toàn

5

Triển khai an toàn



# Tài liệu tham khảo (ví dụ)



## Tài liệu tham khảo

---

- Sabrina Samuel, **Coding Policies for Secure Web Applications**, Master thesis, 2007
- Scarioni, Nardone, **Pro Spring Security** (2e), 2019
- Knutson et al., **Spring Security** (3e), 2017

# Lỗi hỏng do lập trình

- Buffer Overflow
- Double Free
- Use After Free
- Integer Overflow
- Format String
- Race Condition
- Use of Unsecure Component
- Use of Weak Cryptography
- Unsecure Deserialization
- XSS
- CSRF
- XXE
- SQL Injection
- Command Injection
- Path Traversal
- ... (and others)...



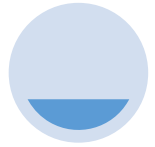
**Lưu ý:**

**Để tránh “phát minh lại bánh xe”, trong nỗ lực lập trình an toàn cần lưu ý khả năng mà các ngôn ngữ/framework/thư viện cung cấp**

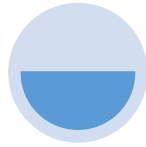
**“All input is evil — until proved otherwise.”**

*Michael Howard and David LeBlanc*

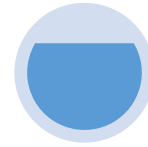
# Xử lý dữ liệu đầu vào an toàn



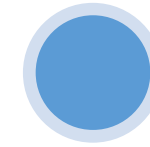
Escaping



Filtering



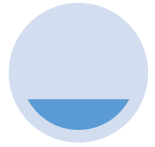
Validation



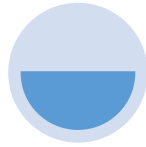
Sanitization

- Là việc chuyển đổi các ký tự điều khiển thành “escape sequence” tương ứng
- Ví dụ với HTML: ký tự “<” được chuyển thành “&lt;”

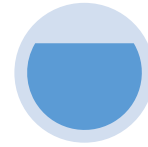
# Xử lý dữ liệu đầu vào an toàn



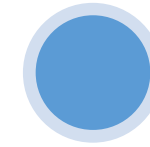
Escaping



Filtering



Validation



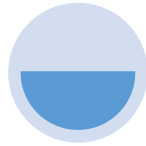
Sanitization

- Là việc loại bỏ các ký tự điều khiển có trong đầu vào
- Ví dụ với SQL: ; ` --

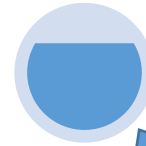
# Xử lý dữ liệu đầu vào an toàn



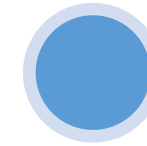
Escaping



Filtering



Validation



Sanitization

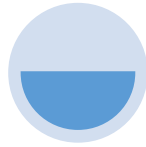
- Là việc so khớp đầu vào với mẫu định trước (thường sử dụng biểu thức chính quy – regular expression)
- Ví dụ với số điện thoại: **`^[0-9]{10}$`**
- Ví dụ với username: **`^[a-z][a-z0-9]{5,19}$`**



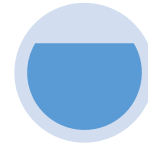
# Xử lý dữ liệu đầu vào an toàn



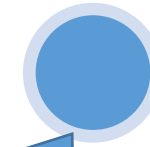
Escaping



Filtering



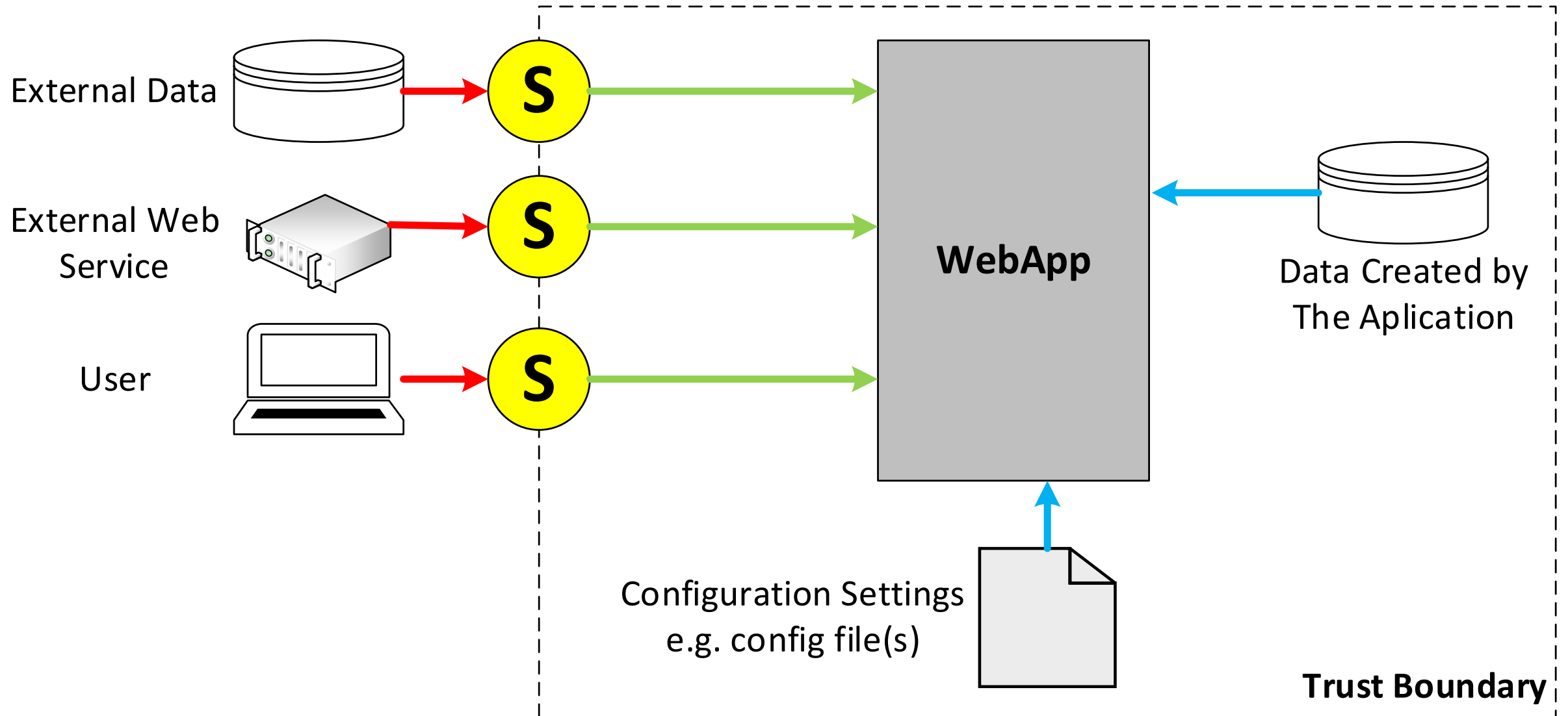
Validation



Sanitization

- Là việc kết hợp Escaping, Filtering và Validation để đảm bảo dữ liệu sau biến đổi là “sạch”, không gây nguy hại.

# Xử lý dữ liệu đầu vào an toàn





**Client-Side validation không thể thay thế Server-Side validation!**

# Xử lý đầu vào với PHP

- htmlspecialchars()    htmlentities()
- filter\_var()            filter\_var\_array()
- filter\_input()           filter\_input\_array()
- mysqli::real\_escape\_string()
- PDO::quote()
- **mysql\_real\_escape\_string()**    //Deprecated, Removed
- Regular Expression



# Ví dụ: validate dữ liệu với PHP

## ❑ Xử lý địa chỉ email

```
$email = $_POST['email'];  
if(filter_var($email, FILTER_VALIDATE_EMAIL))  
{  
    //do something with the email address  
}  
else  
{  
    die("Invalid email address");  
}
```



# Ví dụ: chống XSS với PHP

## ❑ XSS-vulnerable code

```
$name = $_POST['name'];  
echo 'Hello '.$name;
```

## ❑ XSS-safe code

```
$name = $_POST['name'];  
echo 'Hello '.htmlspecialchars($name);
```



# Escaping trong Laravel

## ❑ Xuất chuỗi được escaping

`{{ $data }}`

## ❑ Xuất chuỗi gốc (không escaping)

`{!! $data !!}`



# Escaping trong Laravel

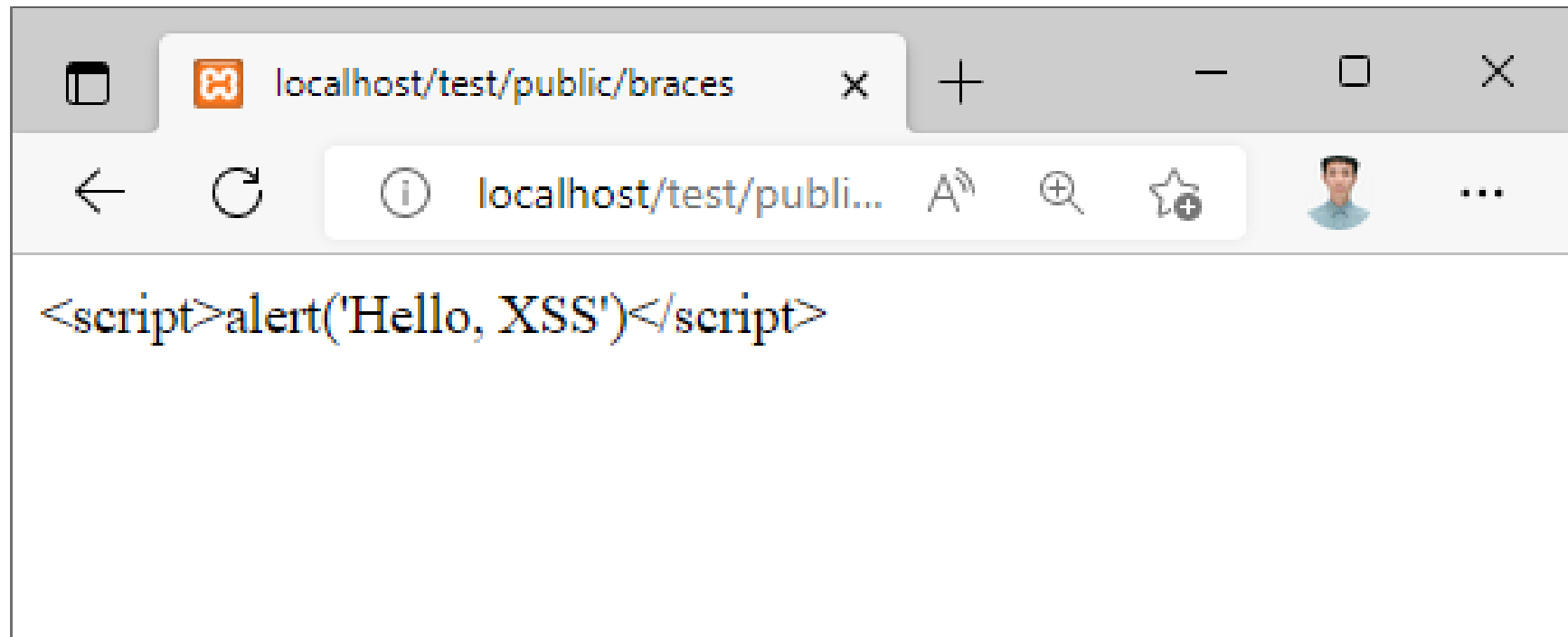
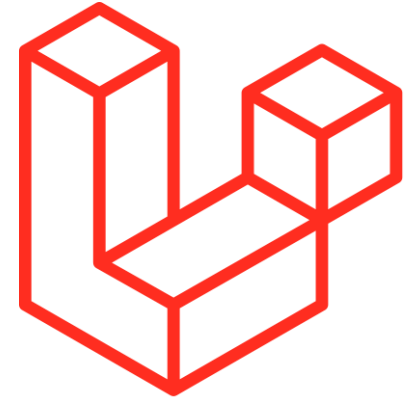
## ❑ Xuất chuỗi được escaping

@php

```
$text = "<script>alert('Hello, XSS')</script>";
```

@endphp

**{{ \$text }}**





# Escaping trong Laravel

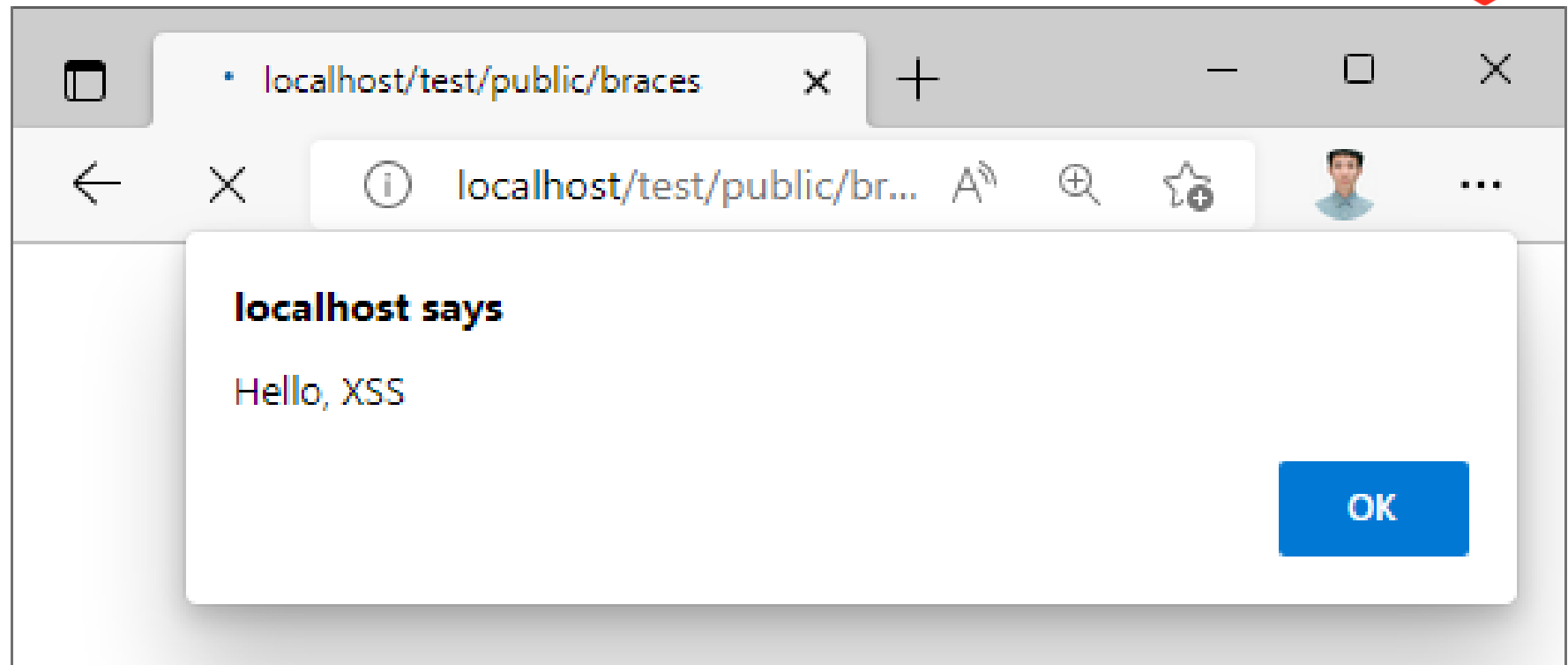
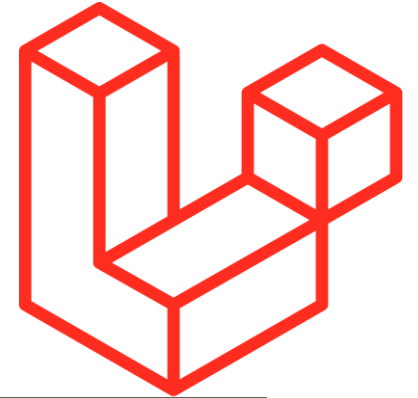
## ❑ Xuất chuỗi KHÔNG được escaping

@php

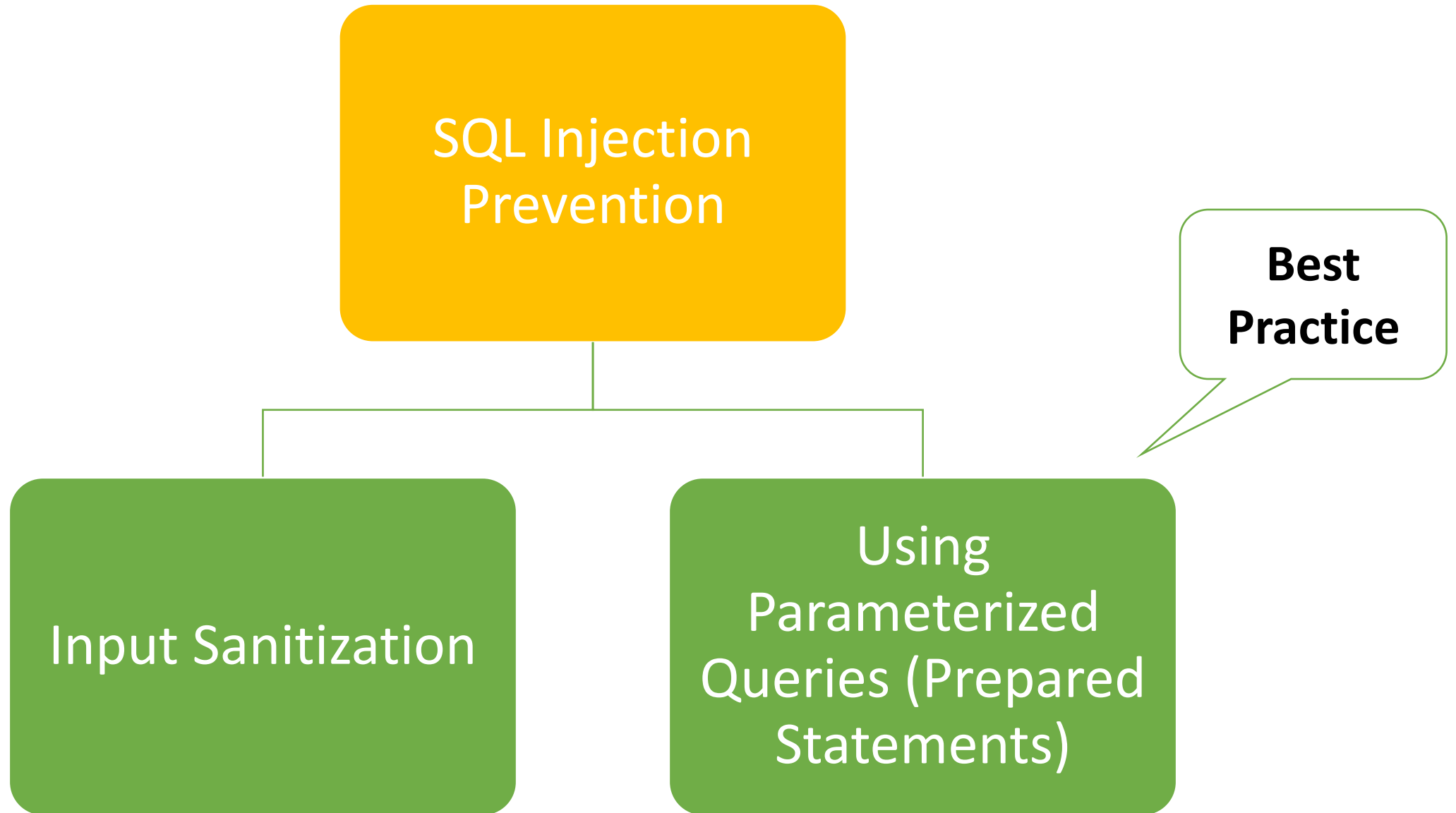
```
$text = "<script>alert('Hello, XSS')</script>";
```

@endphp

**{!! \$text !!}**



# Phòng chống SQL Injection



# Phòng chống SQL Injection: Sanitization

## ❑ Unsafe code

```
$id = $_POST['id'];  
$sql = "SELECT * FROM product WHERE id = `".$id."`";  
$result = $conn->query($sql);
```



## ❑ Safe code

```
$id = filter_var($_POST['id'], FILTER_SANITIZE_NUMBER_INT);  
$sql = "SELECT * FROM product WHERE id = `".$id."`";  
$result = $conn->query($sql);
```

# Phòng chống SQL Injection: Prepared Statement

## ❑ Unsafe code

```
$id = $_POST['id'];  
$sql = "SELECT * FROM product WHERE id = `".$id."`";  
$result = $conn->query($sql);
```



## ❑ Safe code

```
$id = $_POST['id'];  
$stmt = $conn->prepare("SELECT * FROM product WHERE id = ?");  
$stmt->bind_param("i", $id);  
$stmt->execute();
```

# Chống SQL Injection trong Laravel

## ❑ Safe code using Laravel Query Builder

```
public function login(Request $request)
{
    $user = User::where('username', $request->email)
        ->where('password', $request->password)
        ->first();
    if ($user) {
        return $user;
    }
    return "FAILED!";
}
```



# Phòng chống CSRF

- **Nguyên tắc:** sử dụng CSRF Token, là một giá trị khó đoán.
- **Bước 1:** chuẩn bị
  - Sinh và lưu Token vào session
  - Gắn Token vào mỗi form dưới dạng một hidden input (--> thường chỉ xem xét chống CSRF đối với các POST request)
- **Bước 2:** kiểm tra
  - So sánh Token trong request và Token trong session
- **Ý nghĩa:** ứng dụng web chỉ xử lý những request được tạo ra bởi chính nó

# Phòng chống CSRF với PHP

## ❑ Chèn token khi tạo form

```
<?php $_SESSION['csrf_token'] = bin2hex(openssl_random_pseudo_bytes(32)); ?>
<form action="/transfer-money.php" method="post">
    ....
    <input type="hidden" name="_token" value="<?php $_SESSION['csrf_token'] ?>" />
</form>
```

## ❑ Kiểm tra token khi xử lý truy vấn

```
if($_POST['_token'] === $_SESSION['csrf_token'])
    do_transfer(...);
else
    die("Posiable CSRF attack!");
```



# Phòng chống CSRF với Laravel

- Laravel tự động tạo CSRF token cho mỗi session
- Laravel tự động kiểm tra CSRF token cho mọi truy vấn, trừ GET
  - ==> Buộc phải chèn CSRF token, nếu không sẽ thất bại
- Cách chèn CSRF token
  - Sử dụng chỉ thị **@csrf** để tạo thẻ hidden input trong form
  - Tự chèn bằng cách gọi helper `csrf_token()` để lấy giá trị của token
  - Tự động chèn khi gọi **Form::open()** (laravelcollective/html).





# Phòng chống CSRF với Laravel

## ❑ Cách 1: Sử dụng chỉ thị @csrf

```
<form action="/transfer-money" method="post">  
    @csrf  
    ....  
</form>
```

## ❑ Cách 2: gọi helper csrf\_token()

```
<form action="/transfer-money" method="post">  
    <input type="hidden" name="_token" value="{{csrf_token()}}"/>  
    ...  
</form>
```



# Phòng chống CSRF với Laravel

## ❑ Cách 3: tạo form với laravelcollective

```
{!! Form::open(['url' => '/transfer-money', 'method'=>'post']) !!}
```

```
....
```

```
{!! Form::close() !!}
```



## Thảo luận:

- Chống command injection?
- Chống path traversal?
- ...





**Homework:**

**"Tìm hiểu cách chống race condition với ngôn ngữ  
hoặc framework tùy chọn"**

1

Mô hình hóa hiểm họa

2

Mẫu thiết kế an toàn

3

Lập trình an toàn

4

**Kiểm thử an toàn**

5

Triển khai an toàn

# Kiểm thử an toàn phần mềm

- **Application Security Testing (AST)** là việc phân tích phần mềm nhằm tìm ra các điểm yếu, lỗ hổng để khắc phục.
- **Phân loại** xét từ quan điểm của nhà phát triển phần mềm:
  - Kiểm thử tĩnh (SAST = Static AST)
  - Kiểm thử động (DAST = Dynamic AST)
  - Kiểm thử tương tác (IAST = Interactive AST)



# Kiểm thử an toàn phần mềm

- **Application Security Testing (AST)** là việc phân tích phần mềm nhằm tìm ra các điểm yếu, lỗ hổng để khắc phục.
- **Phân loại** xét từ quan điểm của nhà phát triển phần mềm:
  - Kiểm thử tĩnh (SAST = Static AST)
  - Kiểm thử động (DAST = Dynamic AST)
  - Kiểm thử tương tác (IAST = Interactive AST)

- Không thực thi phần mềm
- Chủ yếu là kiểm thử mã nguồn
- **Ưu** : Nhanh, Độ phủ 100% (không gồm lib), Chi tiết lỗi
- **Nhược**: Không phát hiện lỗi logic, Nhiều cảnh báo giả



# Kiểm thử an toàn phần mềm

- **Application Security Testing (AST)** là việc phân tích phần mềm nhằm tìm ra các điểm yếu, lỗ hổng để khắc phục.
- **Phân loại** xét từ quan điểm của nhà phát triển phần mềm:
  - Kiểm thử tĩnh (SAST = Static AST)
  - **Kiểm thử động (DAST = Dynamic AST)**
  - Kiểm thử tương tác (IAST = Interactive AST)

- Thực thi phần mềm
- Thường là black/gray box, out of process
- **Ưu** : Phát hiện lỗi logic; Ít cảnh báo giả
- **Nhược**: Chậm; Độ phủ  $\leq 100\%$





# Kiểm thử an toàn phần mềm

- **Application Security Testing (AST)** là việc phân tích phần mềm nhằm tìm ra các điểm yếu, lỗ hổng để khắc phục.
- **Phân loại** xét từ quan điểm của nhà phát triển phần mềm:
  - Kiểm thử tĩnh (SAST = Static AST)
  - Kiểm thử động (DAST = Dynamic AST)
  - Kiểm thử tương tác (IAST = Interactive AST)

- Là một dạng lai giữa DAST và IAST
- Luôn là in-process (có một sensor được nhúng vào mã của chương trình được kiểm thử)
- Được kích hoạt mỗi khi có sự tương tác (interactive)



# Kiểm thử an toàn phần mềm


- **Application Security Testing (AST)** là việc phân tích phần mềm nhằm tìm ra các điểm yếu, lỗ hổng để khắc phục.
- **Phân loại** xét từ quan điểm của nhà phát triển phần mềm:
  - Kiểm thử tĩnh (SAST = Static AST)
  - Kiểm thử động (DAST = Dynamic AST)
  - Kiểm thử tương tác (IAST = Interactive AST)

Có nhiều công cụ hỗ trợ!



# So sánh SAST, DAST và IAST

## SAST, DAST, and IAST Value Trade-Off

	Coverage	Low False Positives	Exploitability	Code Visibility	SDLC Integration
SAST					
DAST					
IAST					

## Công cụ SAST

**sonarqube** 

**FORTIFY**<sup>®</sup>  
Static Code Analyzer



**HCL**   
**AppScan**

 **Checkmarx**

**VERACODE**

# Công cụ SAST – Tiêu chí đánh giá

- Chi phí
- Ngôn ngữ được hỗ trợ
- Quản lý dự án, quản lý người dùng
- Khả năng phát hiện điểm yếu, lỗ hổng
- Mức độ chi tiết của báo cáo về điểm yếu, lỗ hổng
  - Báo cáo tổng quan; Vị trí điểm yếu, lỗ hổng
  - Truy vết quá trình phát sinh điểm yếu, lỗ hổng
  - Hướng khắc phục

**LIVE DEMO**

## Công cụ DAST



## Công cụ IAST





**LIVE DEMO**

1

Mô hình hóa hiểm họa

2

Mẫu thiết kế an toàn

3

Lập trình an toàn

4

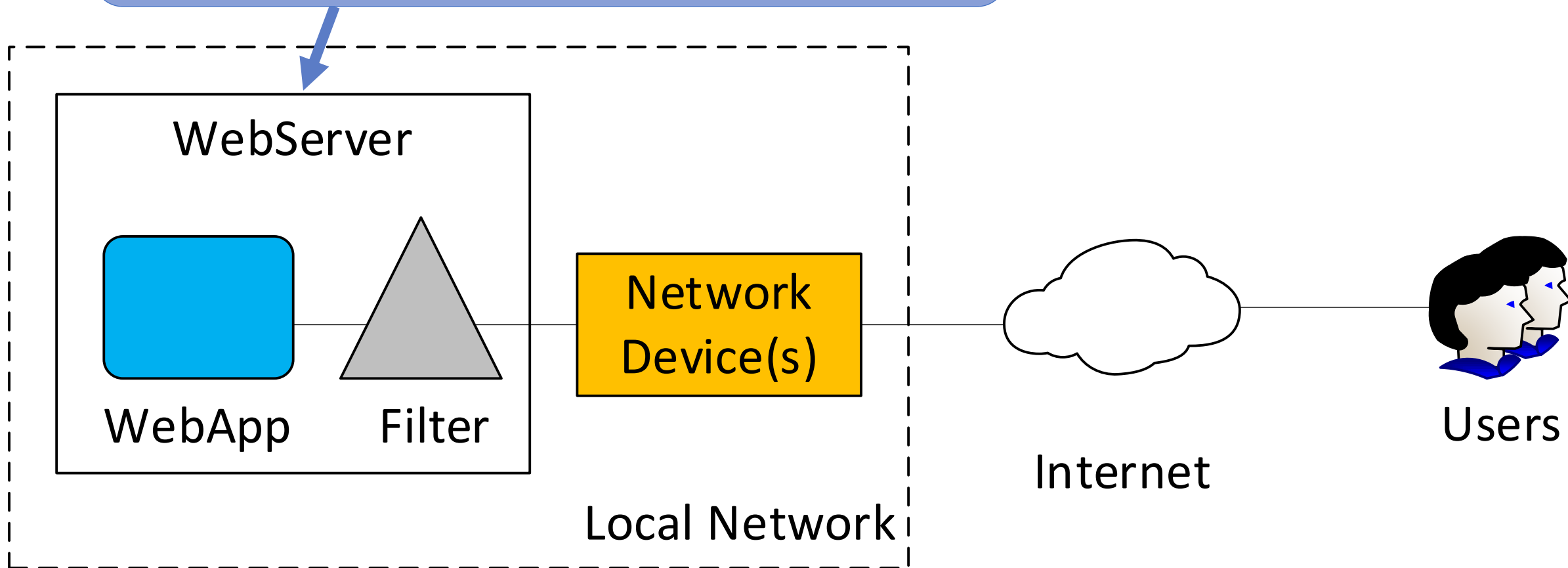
Kiểm thử an toàn

5

**Triển khai an toàn**

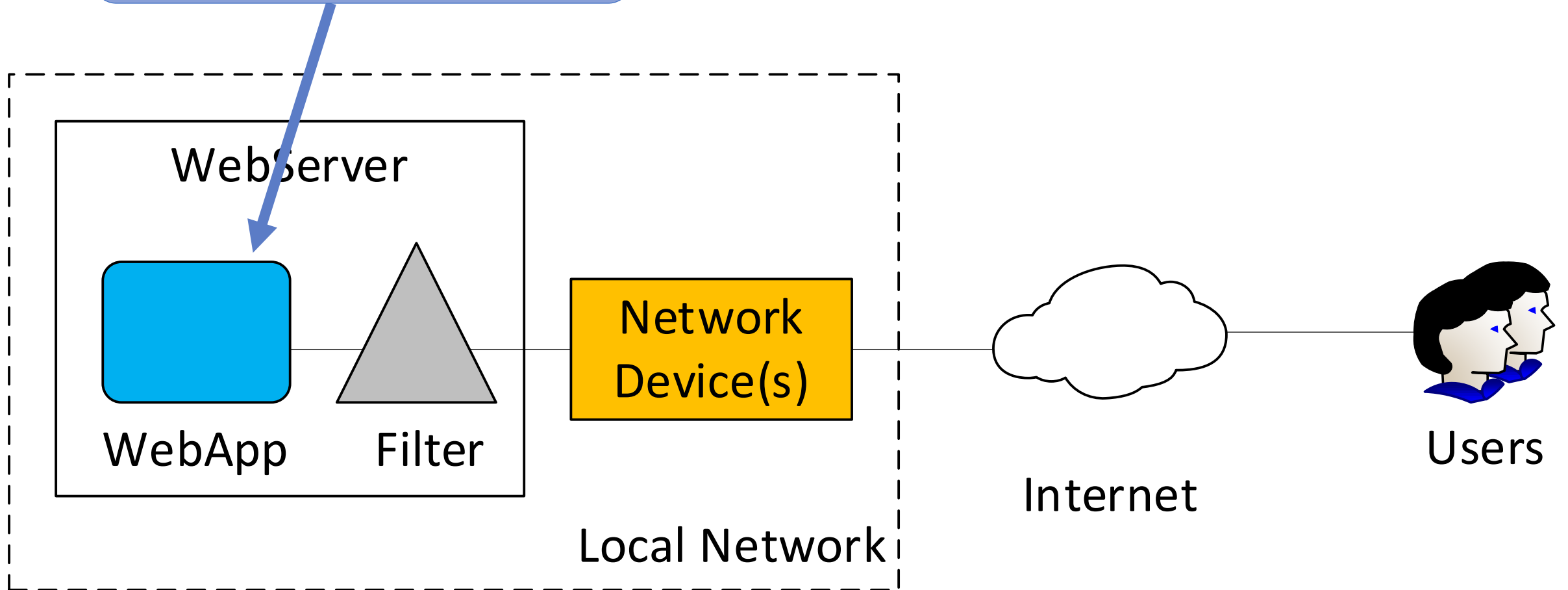
# Triển khai ứng dụng web an toàn

- Cấu hình an toàn (bao gồm TLS)
- Cập nhật HĐH và các thành phần khác;
- Gỡ bỏ những thành phần không cần thiết



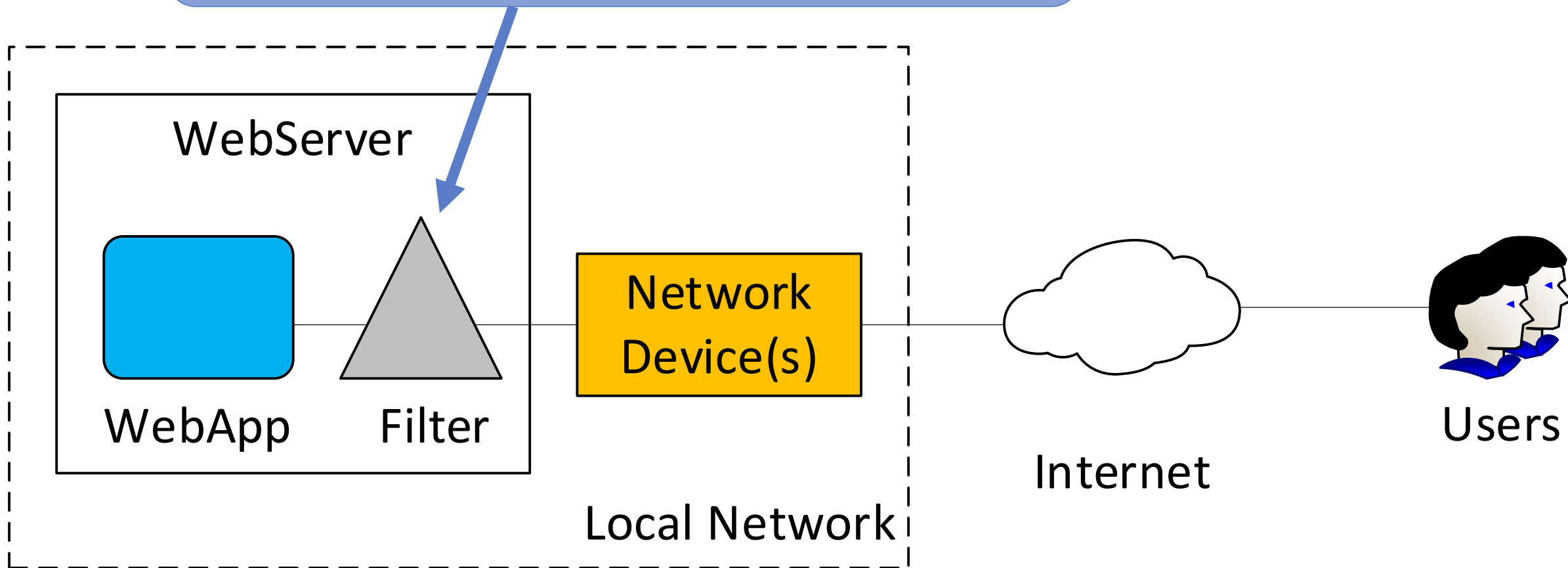
# Triển khai ứng dụng web an toàn

Cấu hình WebApp với các tùy chọn an toàn



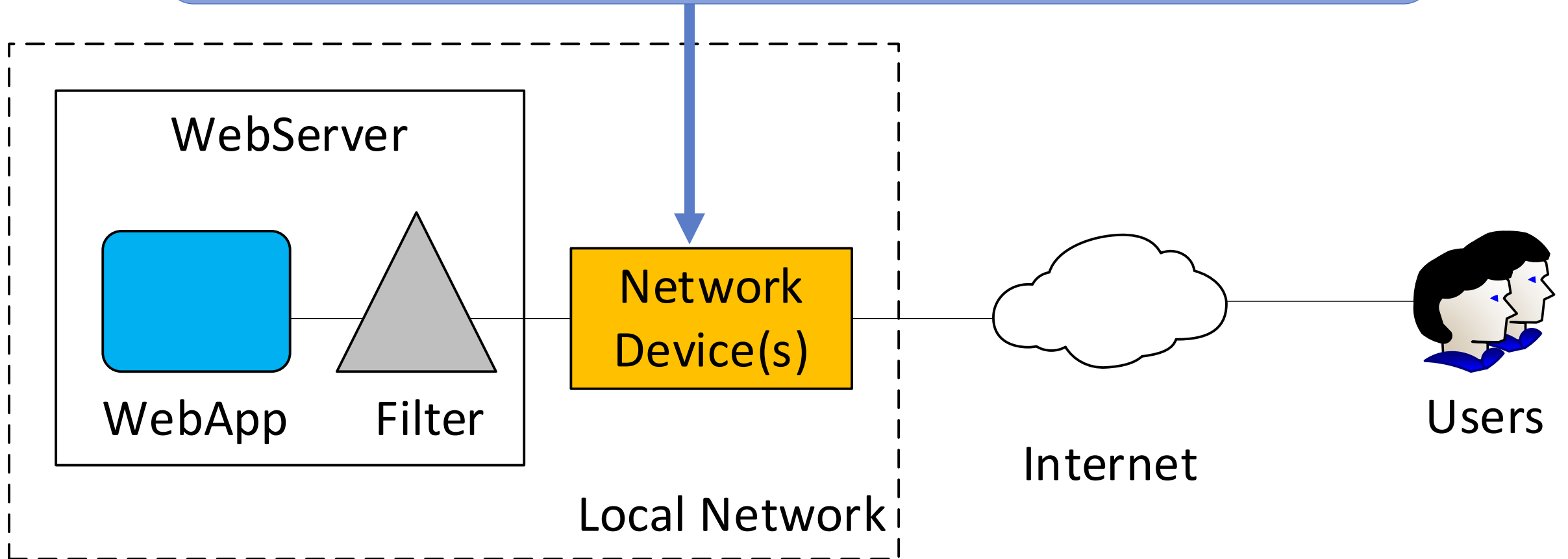
# Triển khai ứng dụng web an toàn

Thiết lập các cơ chế an toàn trên WebServer  
(Antivirus, Personal Firewall, WAF, IDPS sensor...)



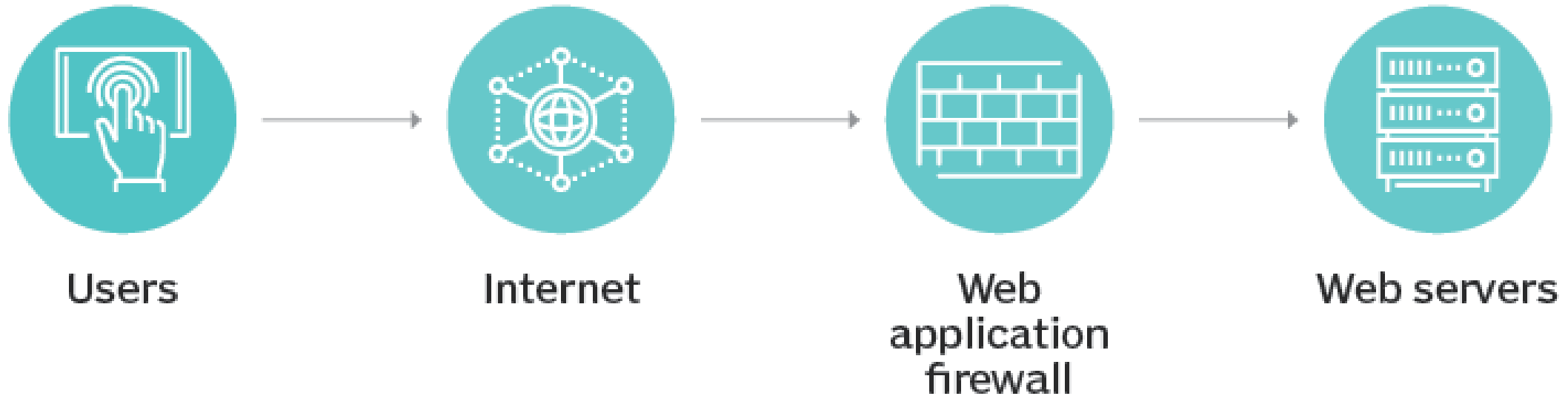
# Triển khai ứng dụng web an toàn

Triển khai, cấu hình các thiết bị an toàn mạng  
(Network Firewall, Network IDS/IPS, Reverse Proxy, Load Balancer...)



# Web Application Firewall

- **Host-based WAF:** ModSecurity, WebKnight
- **Network-based WAF:** Barracuda, FortiWeb,...
- **Cloud-based WAF:** Cloudflare, Sucuri, Apptrana, Imperva...



# Chuẩn bị thực hành

---

1. Thực hành lập trình ứng dụng web an toàn (chống lại XSS, SQL Injection, CSRF)
2. Thực hành kiểm thử an toàn ứng dụng web (SAST, DAST và IAST)
3. Thực hành triển khai WAF



