Simone Riccetti
Massimo Caprinali

La soluzione IBM per la Sicurezza delle applicazioni software

Innovate2010
The Rational Software Conference

Let's build a smarter planet.

Agenda

- Perchè la sicurezza del codice?
- Secure Engineering
- Security & SDLC
- Live Demo

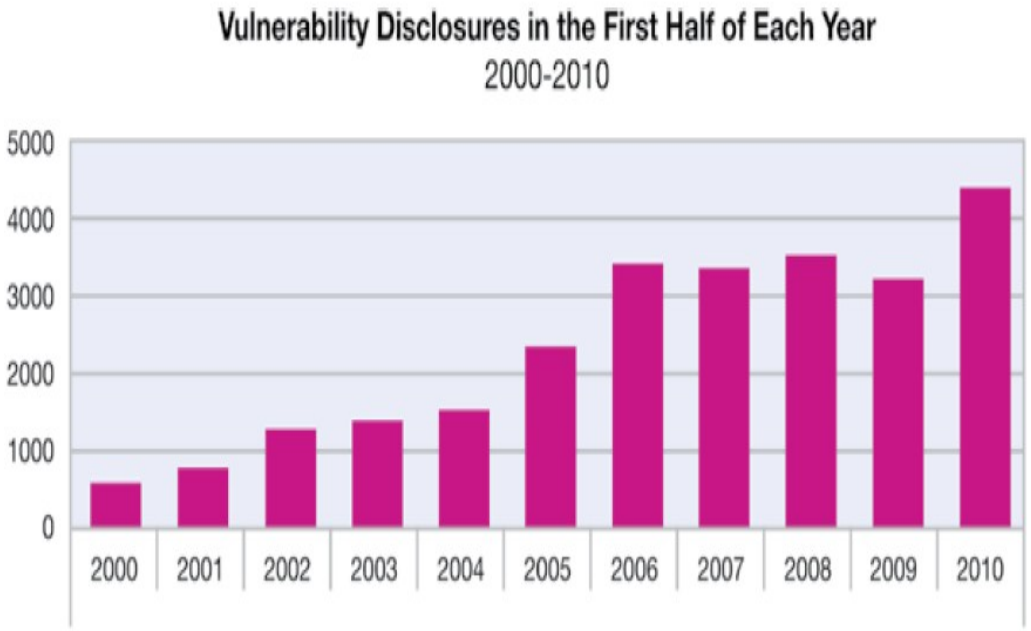

Quali sono i razionali?

Vulnerabilità riportate dai vendor

Rispetto all'H1 2009 c'è stato un incremento delle vulnerabilità del **36%**.

Le vulnerabilità più critiche scoperte in H1 2010 sono sfruttabili da remoto.

Diverse vulnerabilità critiche sono state pubblicate prima della disponibilità di patch.

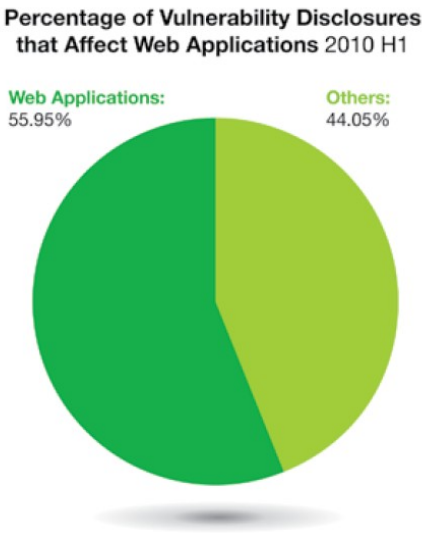


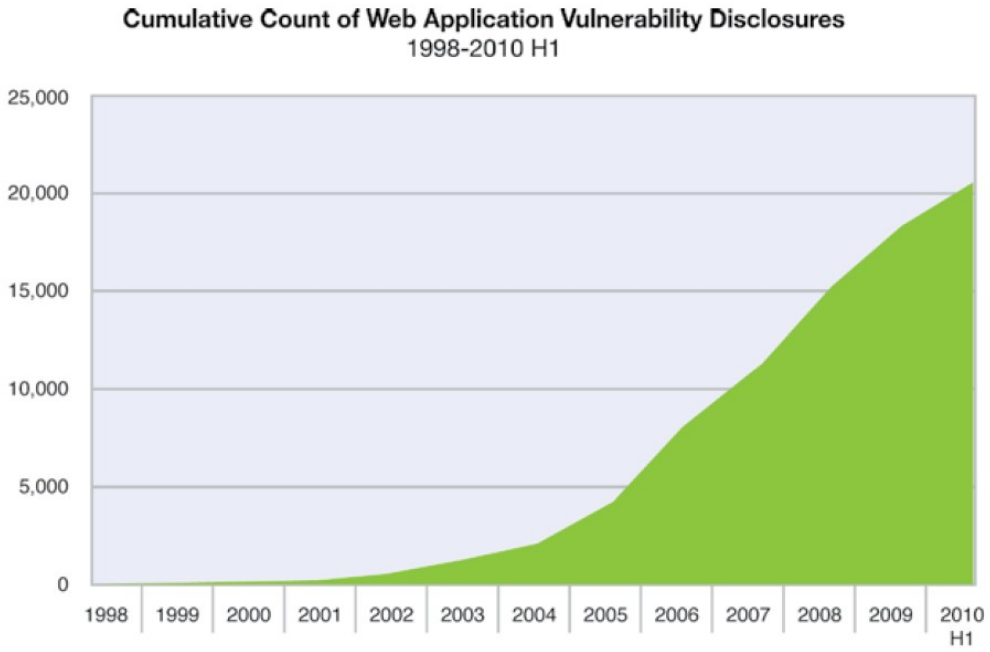
Vulnerabilità riportate dai vendor

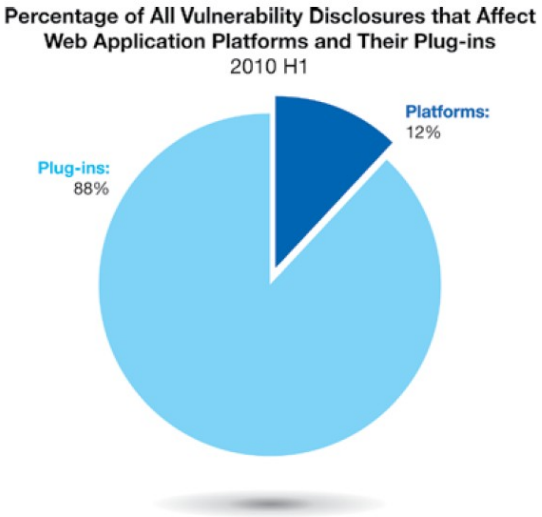
Il **55%** di tutte le vulnerabilità riguarda le applicazioni web.

Le vulnerabilità che permettono il Cross-Site Scripting & SQL injection sono le più diffuse.

I' **88%** delle vulnerabilità di applicazioni web riguarda i plug-in e non la piattaforma base.







Perchè il problema continua a crescere?

Connectivity:

Internet
L'incremento del numero e delle tipologie d vettori di attacco è proporzionale all'incremento di connettività.

SOA/Web Services
Applicazioni legacy non progettate per essere in rete, sono visibili come servizi.

Sistemi Legacy
Non sempre supportano le moderne funzionalità di sicurezza (es. Autenticazione)

Estensibilità:

Il software è sempre più "estensibile" , es browser plug-in, dynamic loadable device driver

L'estensibilità dei software rende difficile prevederne la superficie di attacco nel tempo

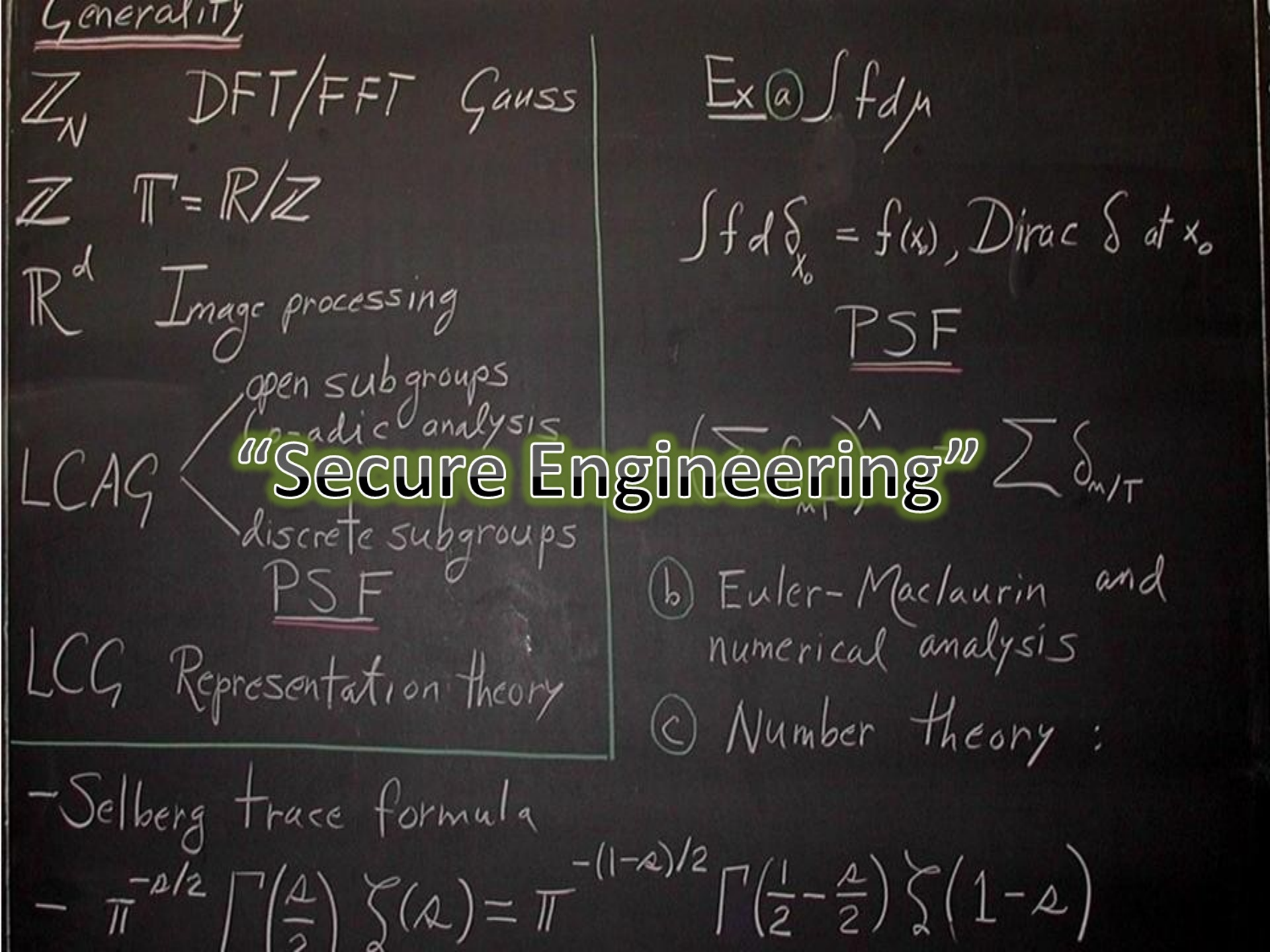
Complessità:

La complessità delle architetture software è sempre maggiore

Nel 1990, Windows 3.1 aveva 2,500,000 linee di codice.

Windows XP ha 40,000,000 linee di codice.

Il numero di bugs è proporzionale al numero di linee di codice



"Secure Engineering"

Software sicuro

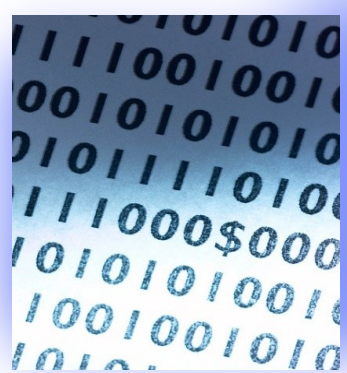
Che caratteristiche deve avere il "Software Sicuro"?

- **Affidabilità**: le funzioni implementate dal software sono quelle attese, in tutte le condizioni di funzionamento
- **Attendibilità**: il software non deve contenere vulnerabilità che possono comprometterne l'Affidabilità
- **Capacità di recupero**: il software deve essere in grado di tollerare eventuali attacchi e recuperare velocemente le funzionalità per cui è stato progettato

Quanto costa correggere una vulnerabilità?

80% dei costi di sviluppo sono spesi per identificare e correggere gli errori

National Institute of Standards & Technology

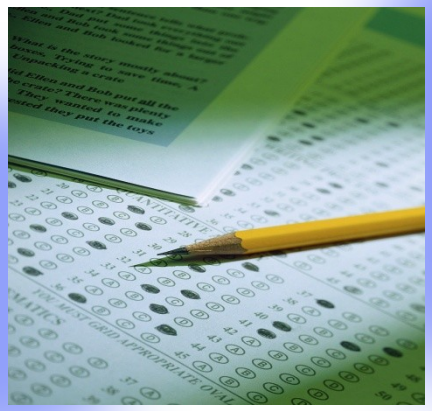


Durante la fase di coding

$25/difetto

Durante la fase di build

$100/difetto



Durante la fase di test

$450/difetto

Quando il prodotto è stato rilasciato

$16,000/difetto

+

Perdita della fiducia del cliente, danni di immagine, etc.

Caper Jones, Applied Software Measurement, 1996

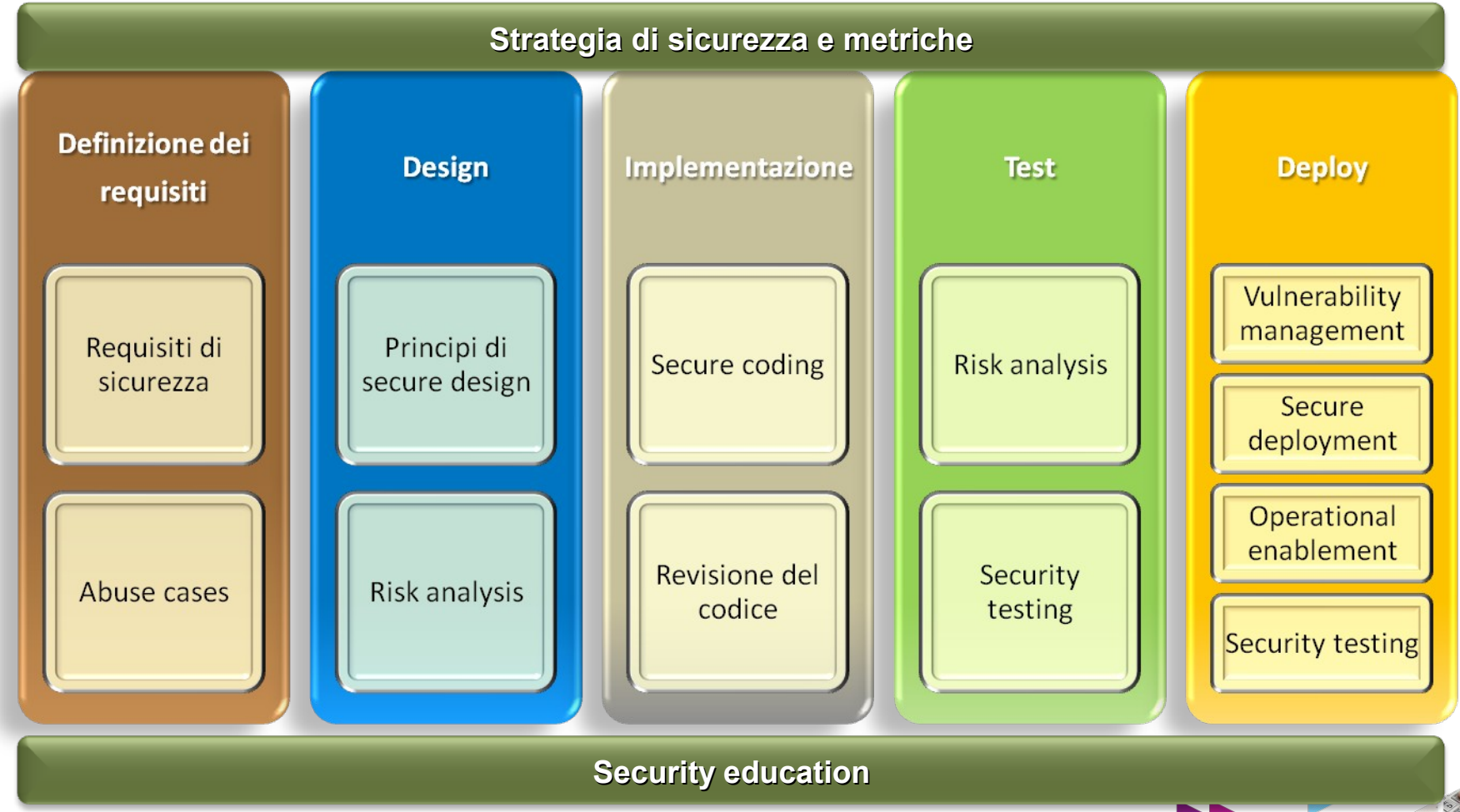
** Source: 2008 GBS Industry standard study. Defect cost derived in assuming it takes 8 hrs to find, fix and repair a defect when found in code and unit test. Defect FFR cost for other phases calculated by using the multiplier on a blended rate of $80/hr.*

Secure Engineering

- Una serie di concetti, principi e best practice con l'obiettivo di integrare la sicurezza nel software progettato.
- Riguarda tutte le fasi del ciclo di sviluppo del software
  - Definizione dei requisiti
  - Design dell'architettura
  - Implementazione
  - Test
  - Deploy
  - *Post deployment*

Secure Engineering non riguarda solo la fase di implementazione, ma è un approccio end-to-end

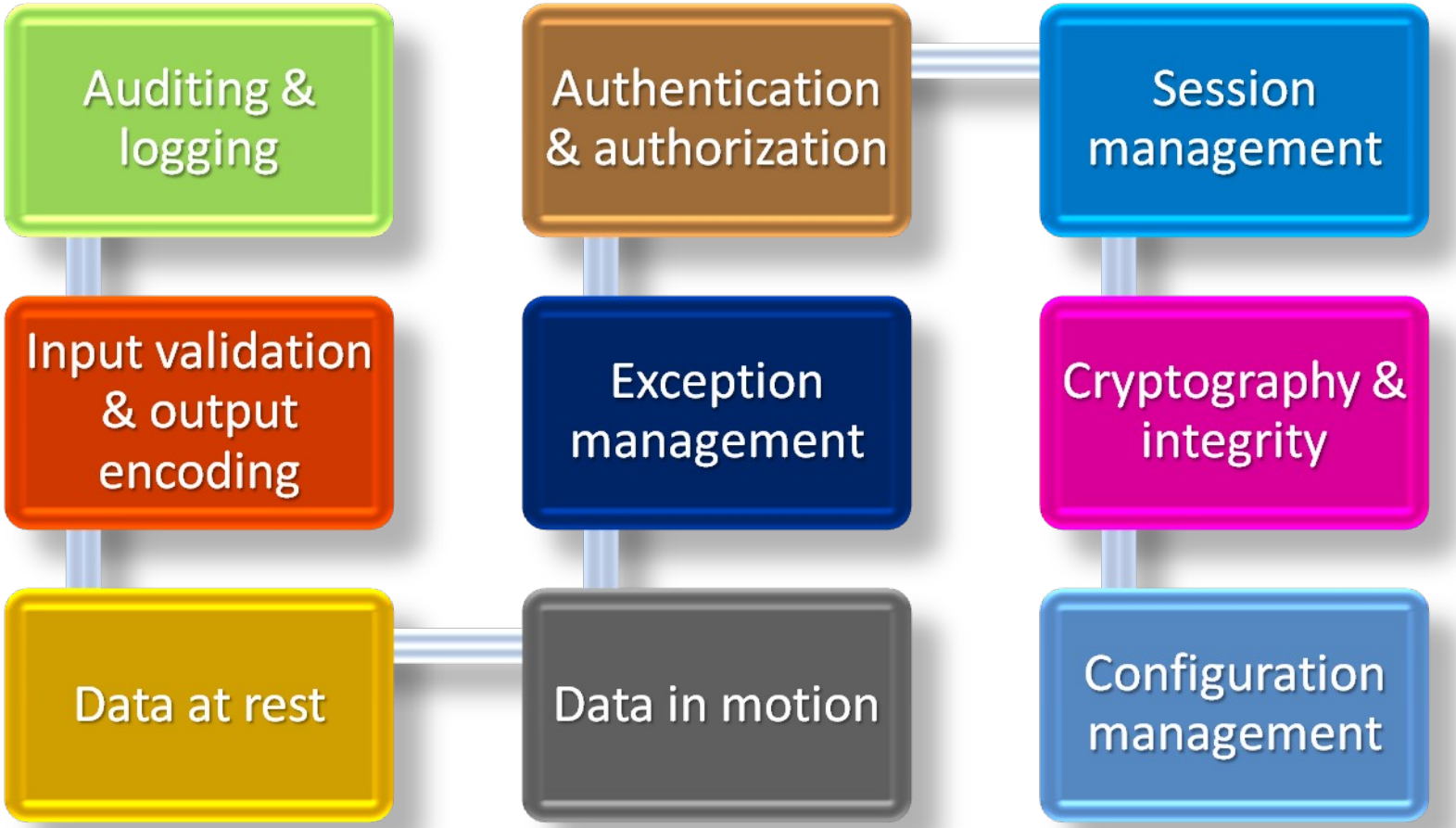
Security & SDLC - Overview



Fase 1:

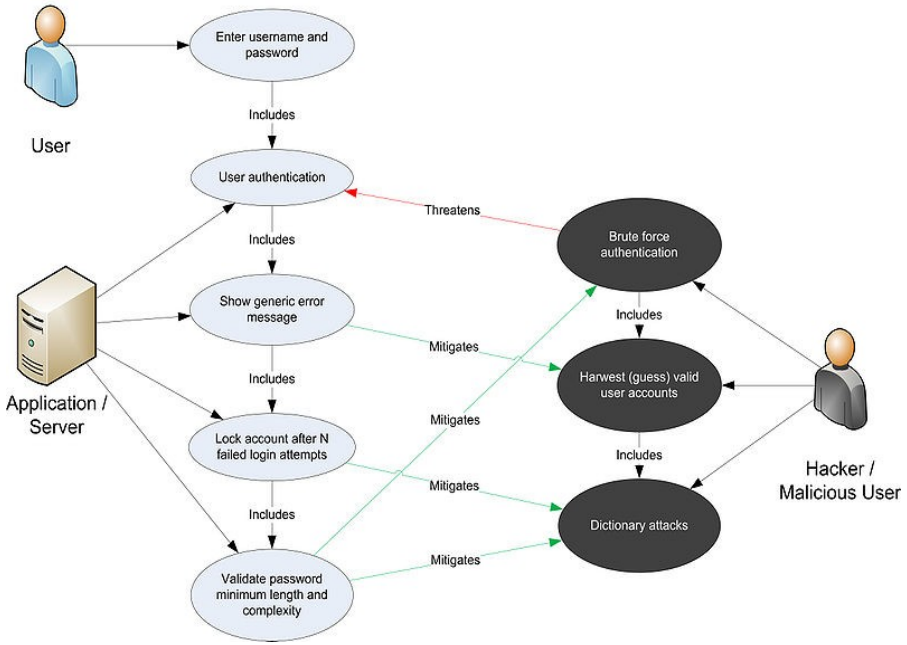
Definizione dei requisiti

Le macro categorie di requisiti di sicurezza



Use Cases e Misuse Cases

- Oltre agli Use Case, che definiscono gli aspetti funzionali dell'applicazione, è necessario identificare i possibili scenari di attacco o Misuse Cases.



E' importante avere la prospettiva dell'attacker

Ref: http://www.owasp.org/index.php/Testing_Guide_Introduction

Fase 2: Design

Principi di Secure Design

Definire le priorità: Risk Analysis

- Analizzare software complessi può risultare un'impresa molto lunga e costosa
- E' necessario focalizzare l'attenzione sul codice più a rischio (es. componente che gestisce il login degli utenti, componente accessibile via rete, codice legacy etc.)

- L'analisi dei rischi permette di:
  - Definire le priorità di analisi in funzione della criticità e della funzioni di business
  - Definire l'approccio all'analisi delle vulnerabilità
  - "misurare" il livello di sicurezza del software prodotto (KPI)
  - Individuare le contromisure più efficaci

Fase 3: Implementazione

(Coding e Build)

Secure Coding

- Quali sono le principali cause di vulnerabilità del codice?

1) Requisiti di sicurezza non definiti correttamente

2) Durante la fase di implementazione vengono inavvertitamente introdotte vulnerabilità nel codice

3) Nella fase di deployment e configurazione non rispecchia i requisiti e le modalità operative previste

Il Secure coding risolvere il problema #2

Es. Vulnerabilità comuni (SANS/MITRE Top 25)

- **Improper Input Validation**
- Improper Encoding or Escaping of Output
- **Failure to Preserve SQL Query Structure (aka 'SQL Injection')**
- **Failure to Preserve Web Page Structure (aka XSS)**
- **Failure to Preserve OS Command Structure**
- Clear text Transmission of Sensitive Information
- Cross-Site Request Forgery (CSRF)
- Race Condition
- Error Message Information Leak
- **Failure to Constrain Operations within the Bounds of a Memory Buffer**
- External Control of Critical State Data
- **External Control of File Name or Path**
- Untrusted Search Path

Es. Vulnerabilità comuni (SANS/MITRE Top 25)

- Failure to Control Generation of Code (aka 'Code Injection')
- Download of Code Without Integrity Check
- Improper Resource Shutdown or Release
- Improper Initialization
- Incorrect Calculation
- Improper Access Control (Authorization)
- **Use of a Broken or Risky Cryptographic Algorithm**
- Hard-Coded Password
- Insecure Permission Assignment for Critical Resource
- Use of Insufficiently Random Values
- Execution with Unnecessary Privileges
- **Client-Side Enforcement of Server-Side Security**

Fase 4: Test

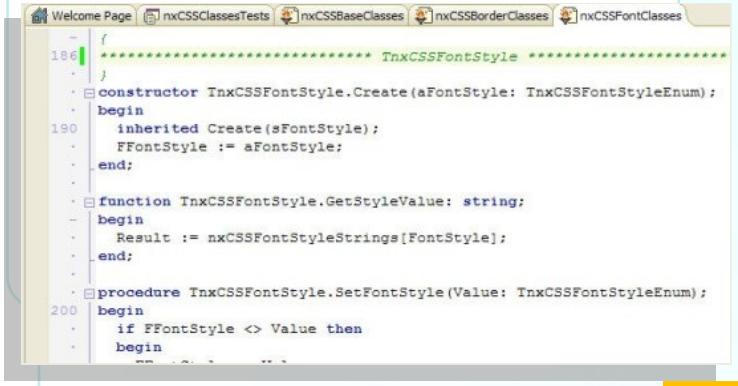
```
automatic code()
{
    testing tools;
}
```

Code Review/Testing Tools

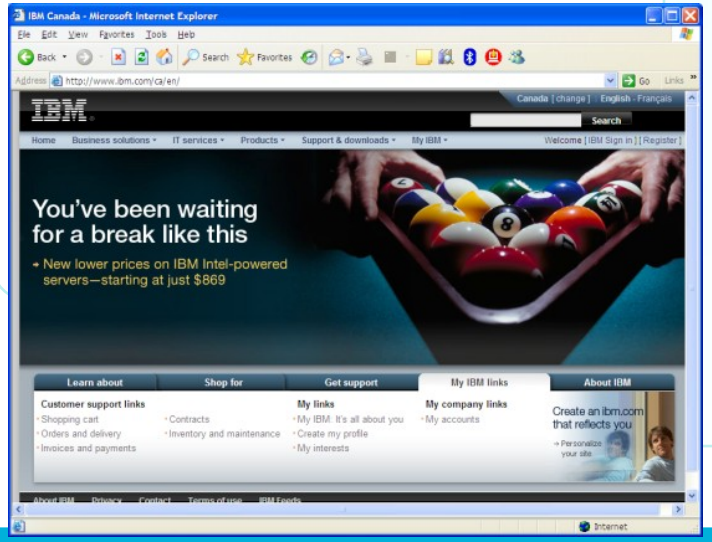
- Esistono tre approcci fondamentali per analizzare le vulnerabilità di un'applicazione:
  - **White Box**: viene analizzato il codice sorgente e l'applicazione NON viene eseguita
  - **Black Box**: NON si ha accesso al codice sorgente e l'applicazione viene eseguita
  - **Gray Box**: si ha parzialmente accesso al codice sorgente e l'applicazione viene eseguita
- L'analisi delle vulnerabilità può essere fatta a mano o con tool automatici
- La revisione manuale è efficace per porzioni limitate di applicazione e codice. Spesso non è la strada corretta
  - Richiede skill elevati di sicurezza
  - Richiede molto tempo (pensate cosa significa controllare 500K LoC…)

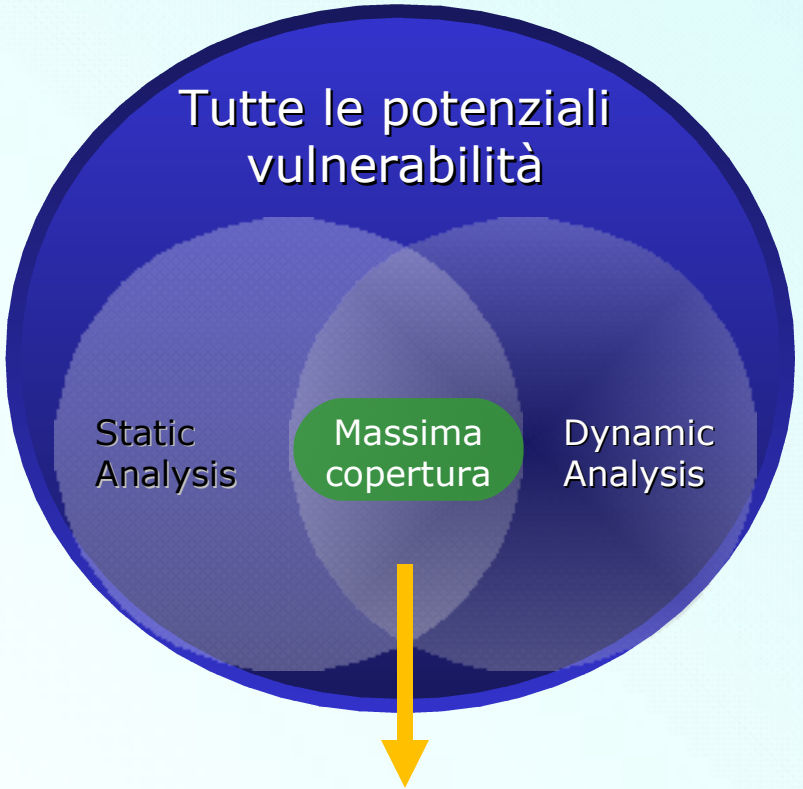
Tecnologie di security testing…

Static Code Analysis = Whitebox



Dynamic Analysis = Blackbox

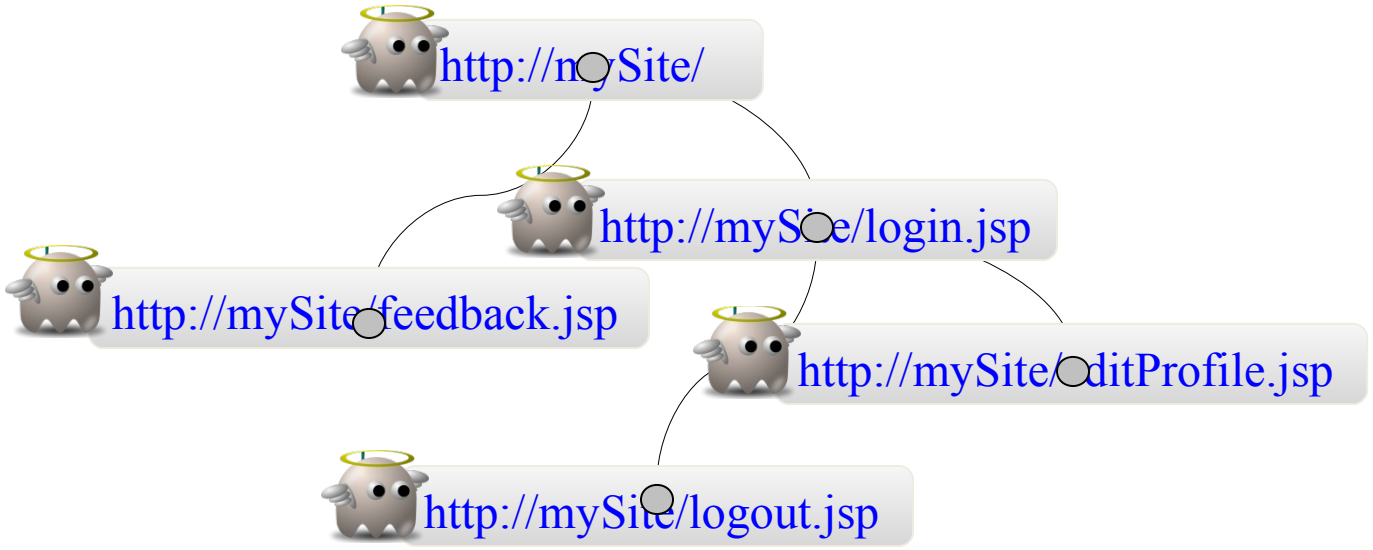




Gray Box Analysis

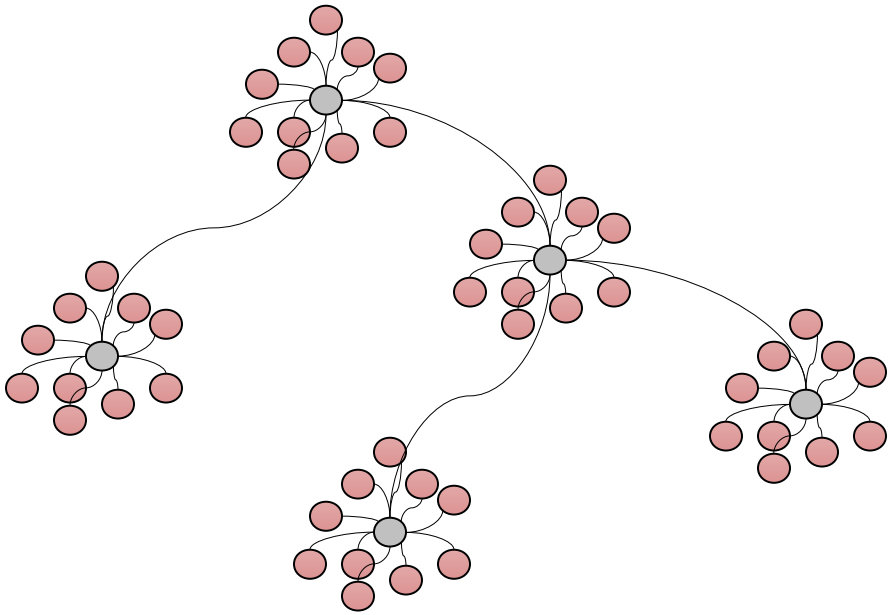
Come funzionano i Black Box scanner

- Stage 1: si effettua il crawling dell'applicazione

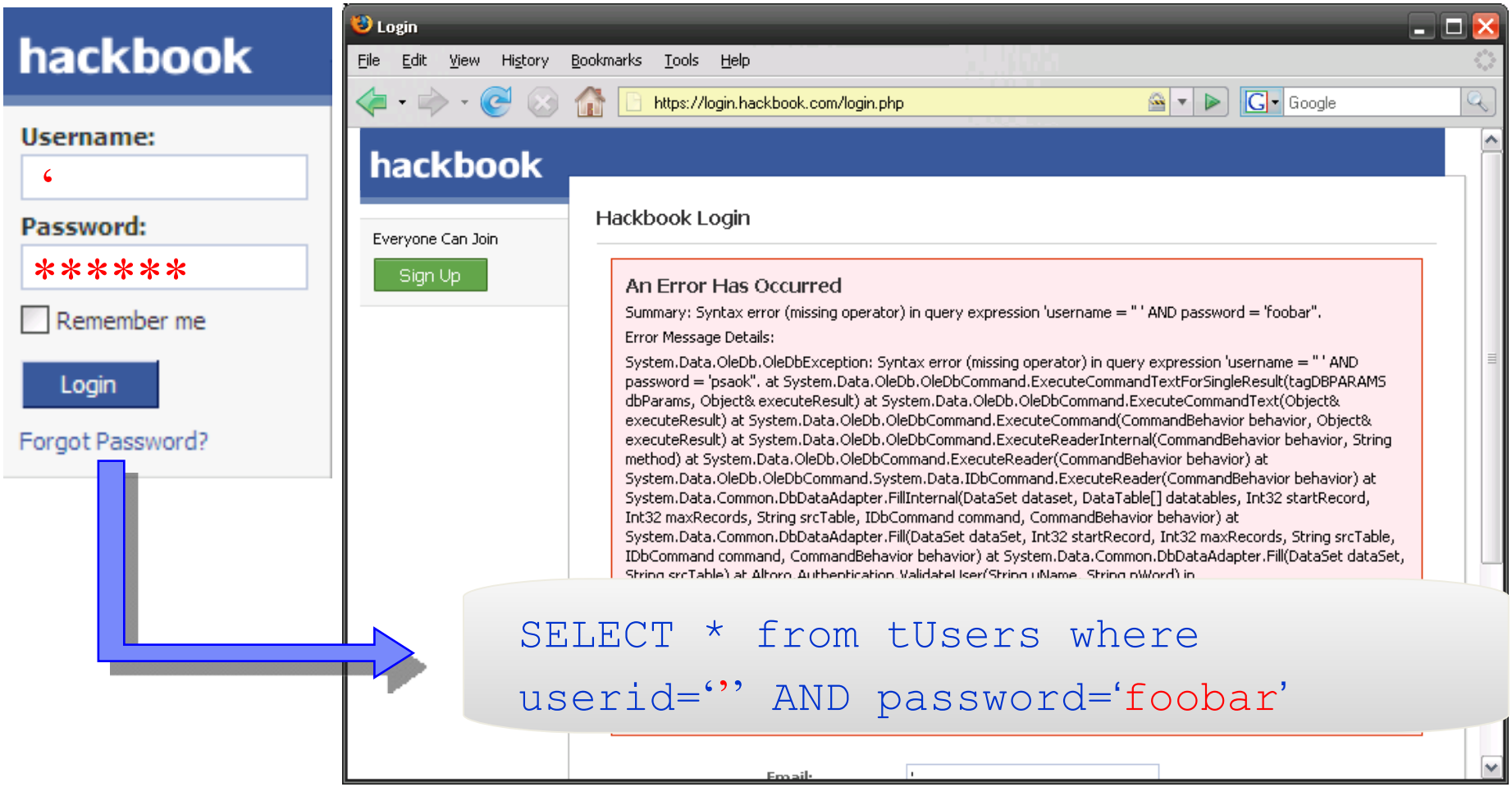


Come funzionano i Black Box scanner

- Stage 1: si effettua il crawling dell'applicazione
- Stage 2: Vengono creati i test specifici



Black-box (es. SQL Injection)



Sistemi White-Box (es. SQL Injection)

Source
Metodo che restituisce una tainted string

```
// ...
String username = request.getParameter("username");
String password = request.getParameter("password");

// ...
String query = "SELECT * from tUsers where " +
  "userid='" + username + "' " +
  "AND password='" + password + "'";

// ...
ResultSet rs = stmt.executeQuery(query);
```

java.io.BufferedReader
read

L'utente può inserire comandi arbitrari

Sink
Un metodo potenzialmente pericoloso

java.sql.Statement
executeQuery

Sistemi White-Box (es. SQL Injection)

da un Source …

```
String username = request.getParameter("username");
// ...
String username = request.getParameter("username");
String password = request.getParameter("password");

// ...
String query = "SELECT * from tUsers where " +'
  "userid='" + username + "' " +
String query = "SELECT …" + username
// ...
ResultSet rs = stmt.executeQuery(query);
ResultSet rs = stmt.executeQuery(query);
```

… a un Sink

Un tipico Fix (non sempre il migliore per SQL Injection…)

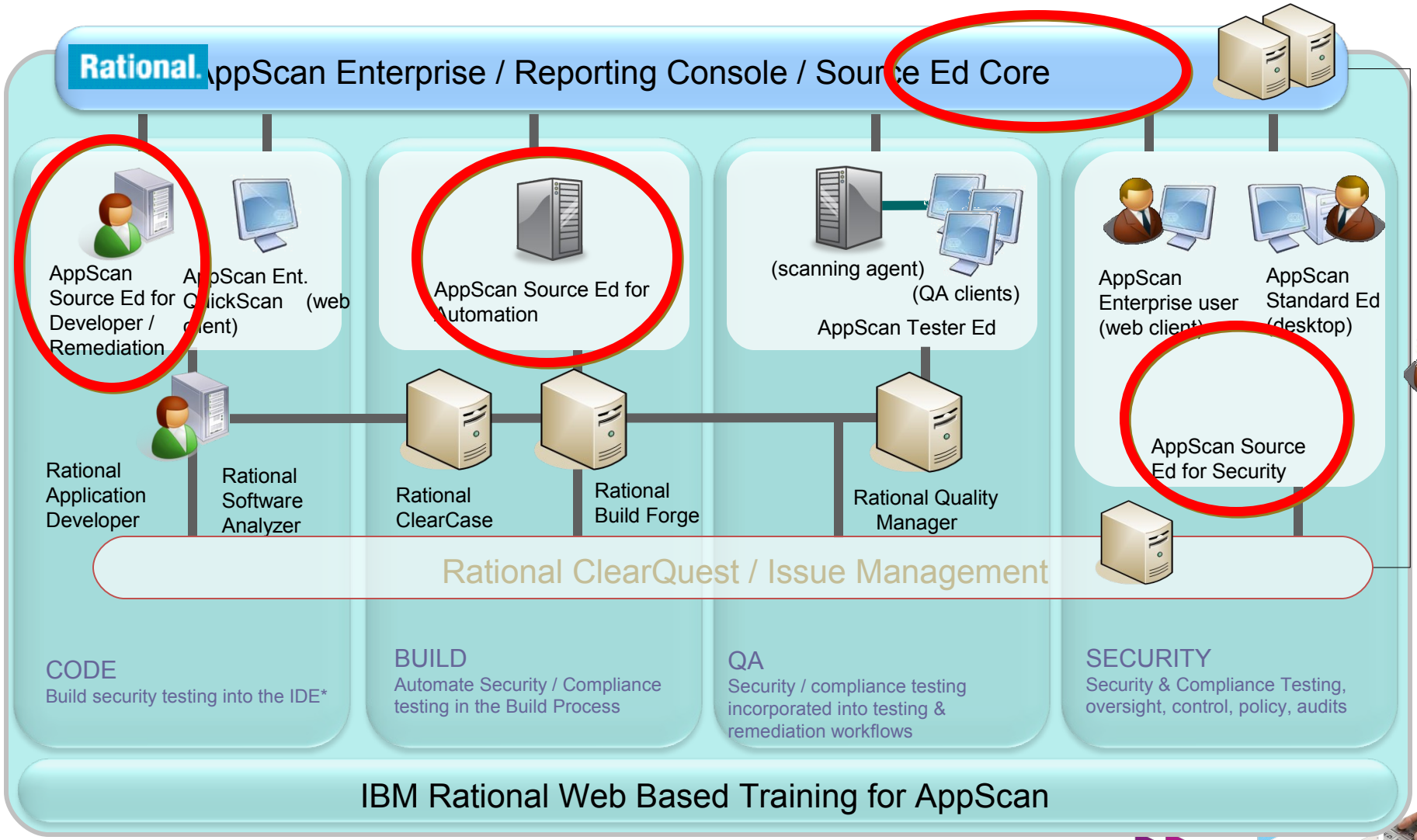
```
// ...
String username = request.getParameter("username");
String password = request.getParameter("password");

// ...
String query = "SELECT * from tUsers where " +
  "userid='" + Encode(username) + "' " +
  "AND password='" + Encode(password) + "'";

// ...
ResultSet rs = stmt.exec    eQuery(query);
```

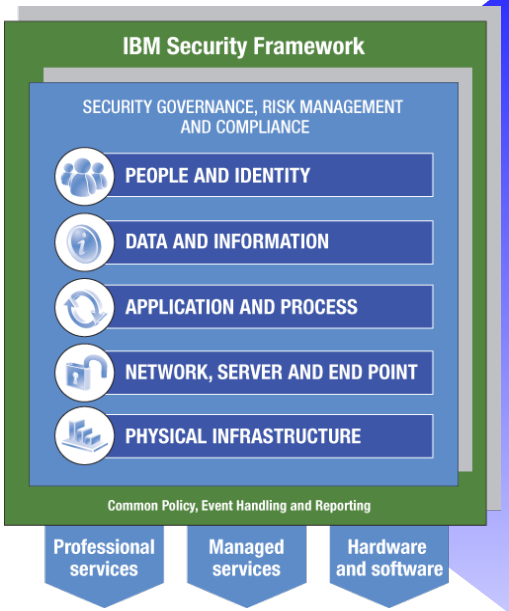
Sanitizer:
è un metodo che elimina la taint string

IBM Rational AppScan Ecosystem



IBM Security Services portfolio

= Professional services
= Managed services
= Cloud services



GRC

Security Governance and Compliance Services

Identity & Access Mgmt Services

Managed Identity and Access Management

Data Security Services

E-mail Security

Application Security Services

Threat Mitigation Services

Firewall

UTM

IDS/IPS

Protection Mgt

Vulnerability Assessment

Web/URL Filtering

Web/URL Filtering

Security Events

Security Logs

Threat Assessment

Physical Security Services

Grazie!