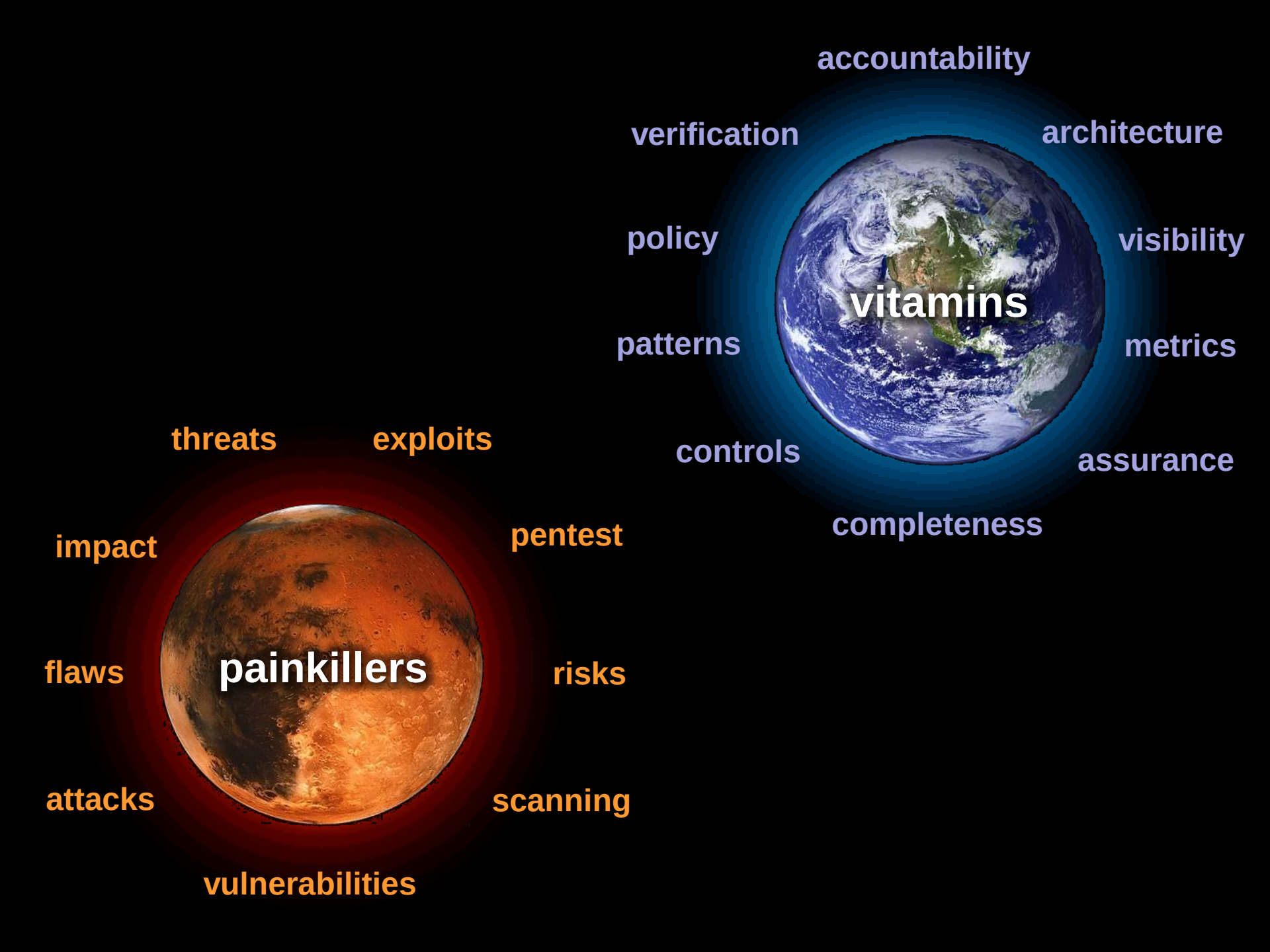
ASPECT SECURITY
application security specialists

The OWASP Enterprise Security API (ESAPI) Project and other musings

Dave Wichers
Aspect Security COO
Volunteer Conferences Chair of OWASP
Member of OWASP Board
dave.wichers@aspectsecurity.com
443-745-6268


accountability
verification
architecture
policy
visibility
vitamins
patterns
metrics
controls
assurance
completeness
threats
exploits
impact
pentest
flaws
painkillers
risks
attacks
scanning
vulnerabilities

Imagine a World with . . .

- **Standard security controls**
  - **that enable developers to build secure apps**
- **A straightforward way of comparing application security verification services**
  - **which truly verify security, not just report bugs**
- **Security analysis tools that report**
  - **positive results they identified**
  - **not just what flaws they uncovered**

Open Web Application Security Project

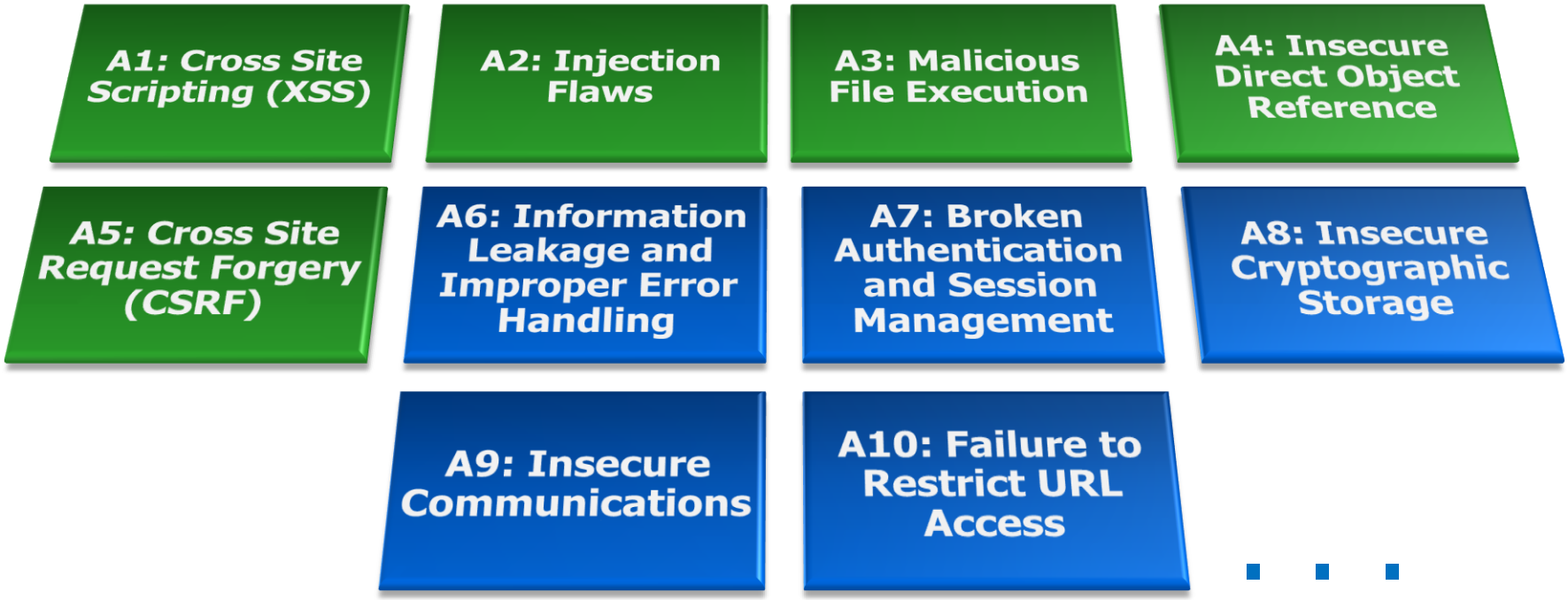
Participants

Chapters

http://www.owasp.org/

The Problem – How do you deal with this?

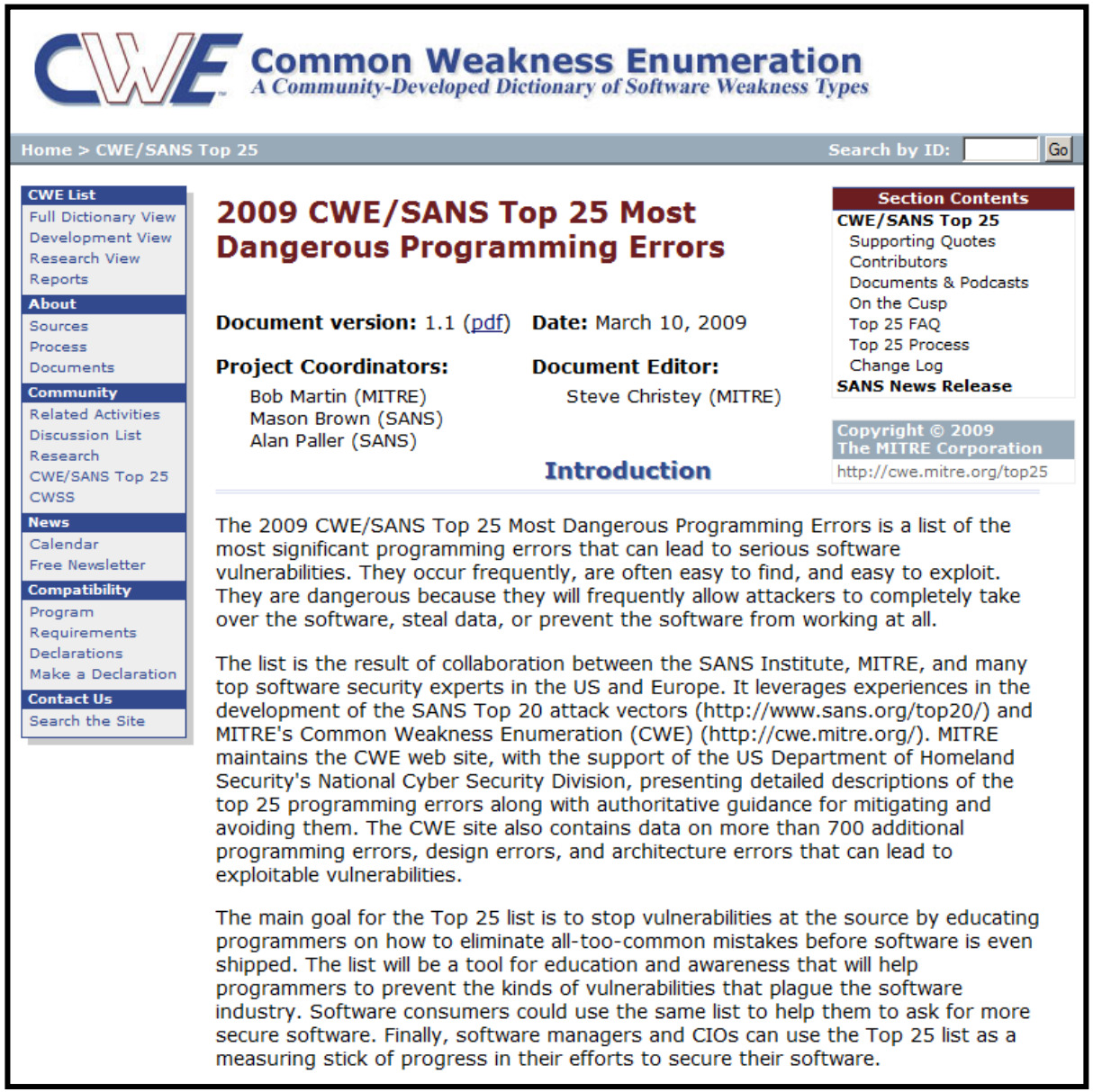
OWASP Top Ten (2007 Edition)



OWASP
The Open Web Application Security Project
http://www.owasp.org

http://www.owasp.org/index.php/Top_10

Or this …



CWE Common Weakness Enumeration
A Community-Developed Dictionary of Software Weakness Types

Home > CWE/SANS Top 25 | Search by ID: Go

CWE List
Full Dictionary View
Development View
Research View
Reports
About
Sources
Process
Documents
Community
Related Activities
Discussion List
Research
CWE/SANS Top 25
CWSS
News
Calendar
Free Newsletter
Compatibility
Program Requirements
Declarations
Make a Declaration
Contact Us
Search the Site

2009 CWE/SANS Top 25 Most Dangerous Programming Errors

Document version: 1.1 (pdf) **Date:** March 10, 2009

Project Coordinators:
Bob Martin (MITRE)
Mason Brown (SANS)
Alan Paller (SANS)

Document Editor:
Steve Christey (MITRE)

Section Contents
CWE/SANS Top 25
Supporting Quotes
Contributors
Documents & Podcasts
On the Cusp
Top 25 FAQ
Top 25 Process
Change Log
SANS News Release

Copyright © 2009 The MITRE Corporation
http://cwe.mitre.org/top25

Introduction

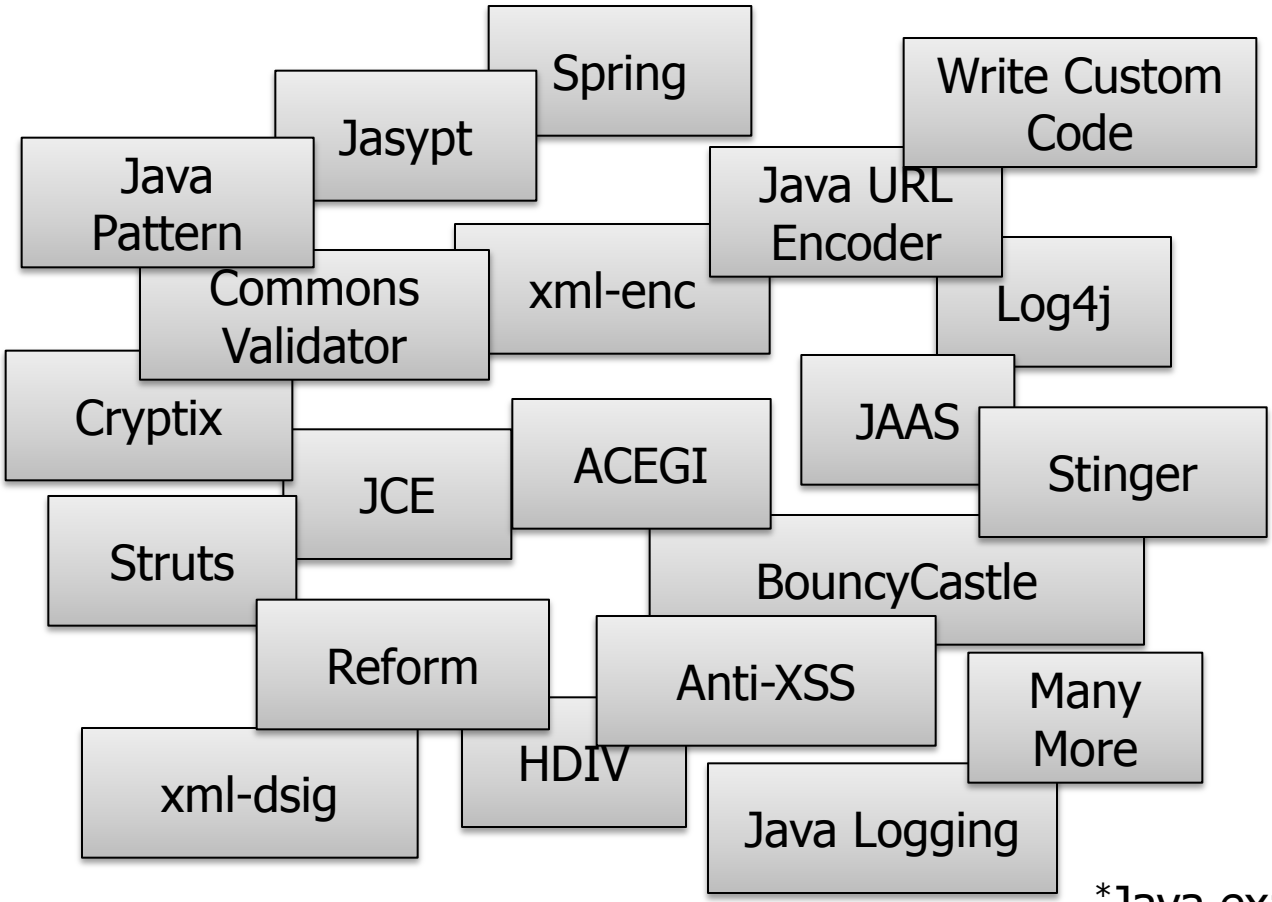
The 2009 CWE/SANS Top 25 Most Dangerous Programming Errors is a list of the most significant programming errors that can lead to serious software vulnerabilities. They occur frequently, are often easy to find, and easy to exploit. They are dangerous because they will frequently allow attackers to completely take over the software, steal data, or prevent the software from working at all.

The list is the result of collaboration between the SANS Institute, MITRE, and many top software security experts in the US and Europe. It leverages experiences in the development of the SANS Top 20 attack vectors (http://www.sans.org/top20/) and MITRE's Common Weakness Enumeration (CWE) (http://cwe.mitre.org/). MITRE maintains the CWE web site, with the support of the US Department of Homeland Security's National Cyber Security Division, presenting detailed descriptions of the top 25 programming errors along with authoritative guidance for mitigating and avoiding them. The CWE site also contains data on more than 700 additional programming errors, design errors, and architecture errors that can lead to exploitable vulnerabilities.

The main goal for the Top 25 list is to stop vulnerabilities at the source by educating programmers on how to eliminate all-too-common mistakes before software is even shipped. The list will be a tool for education and awareness that will help programmers to prevent the kinds of vulnerabilities that plague the software industry. Software consumers could use the same list to help them to ask for more secure software. Finally, software managers and CIOs can use the Top 25 list as a measuring stick of progress in their efforts to secure their software.

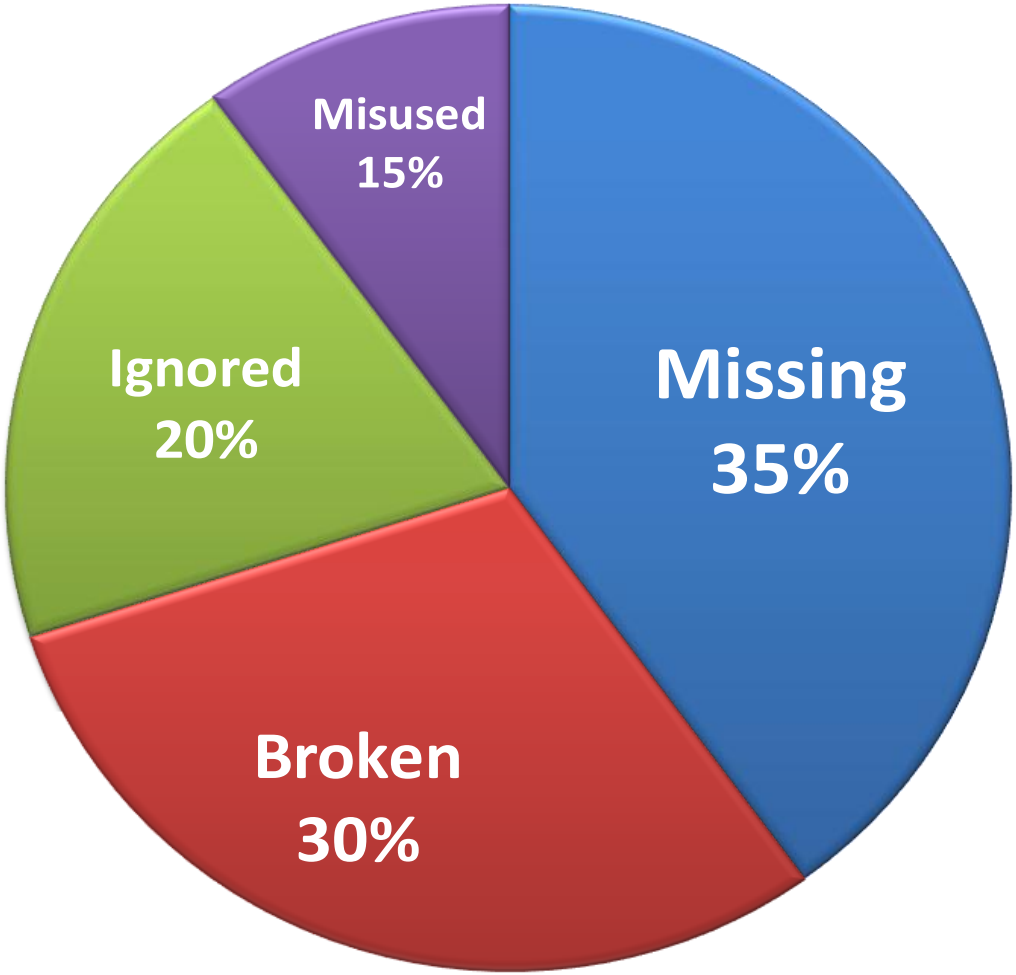
http://cwe.mitre.org/top25/

When you are using this?

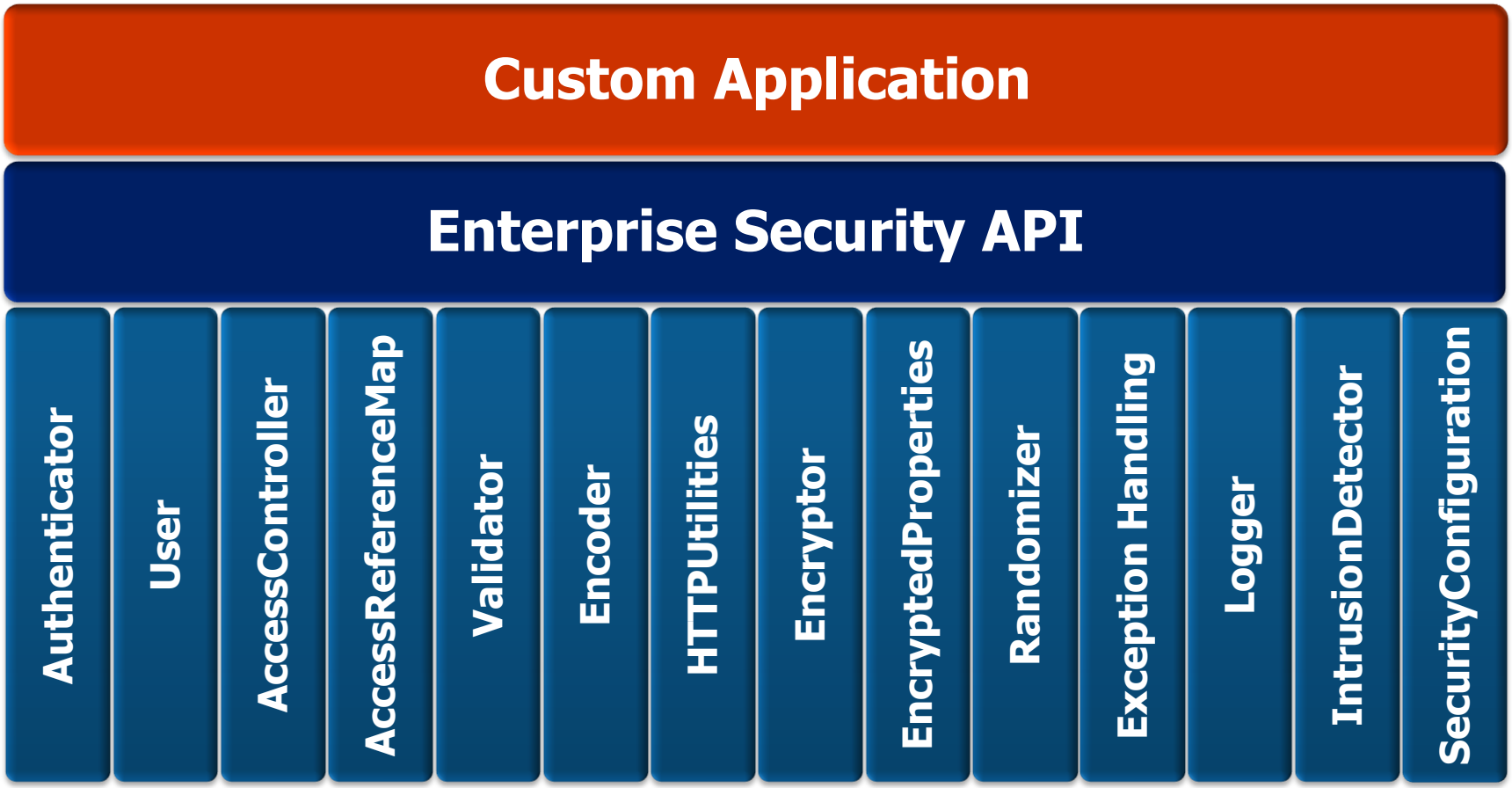


*Java examples

Vulnerabilities and Security Controls



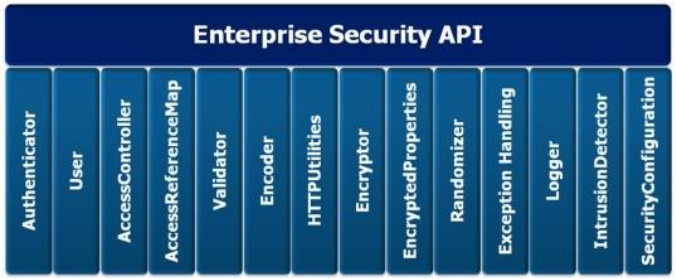
What Methods Do Developers Need?



Standardize and Isolate

Expected ESAPI Influence

OWASP (Refined and Proven)



Organization Influence

Org 1 Enterprise Security API

. . .

Org N Enterprise Security API

Language Influence

Microsoft .NET

Java

C/C++

Haskell
A Purely Functional Language

python

Microsoft ASP

macromedia COLDFUSION

php

Framework Influence

Jakarta Struts

Spring

YAHOO!

RAILS

Google Gears

ADOBE AIR

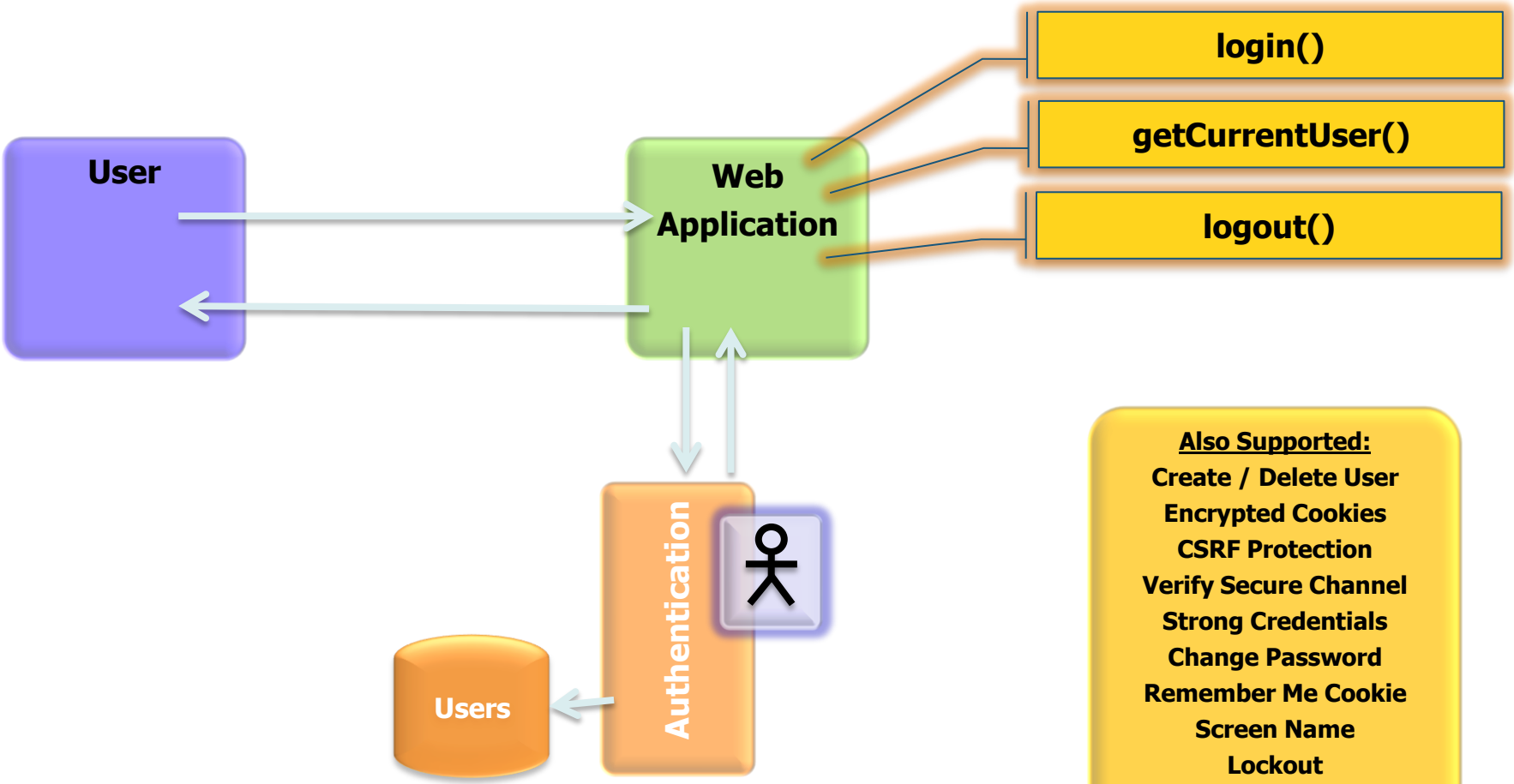
Deceptively Tricky Problems for Developers

1. **Buffer Overflows**
   - **So we fixed some of the languages**
2. **Input Validation and Output Encoding**
3. **Authentication and Identity**
4. **URL Access Control**
5. **Business Function Access Control**
6. **Data Layer Access Control**
7. **Presentation Layer Access Control**
8. **Errors, Logging, and Intrusion Detection**
9. **Encryption, Hashing, and Randomness**
   - **So we standardized the algorithms, but not how to use them, manage keys, etc.**

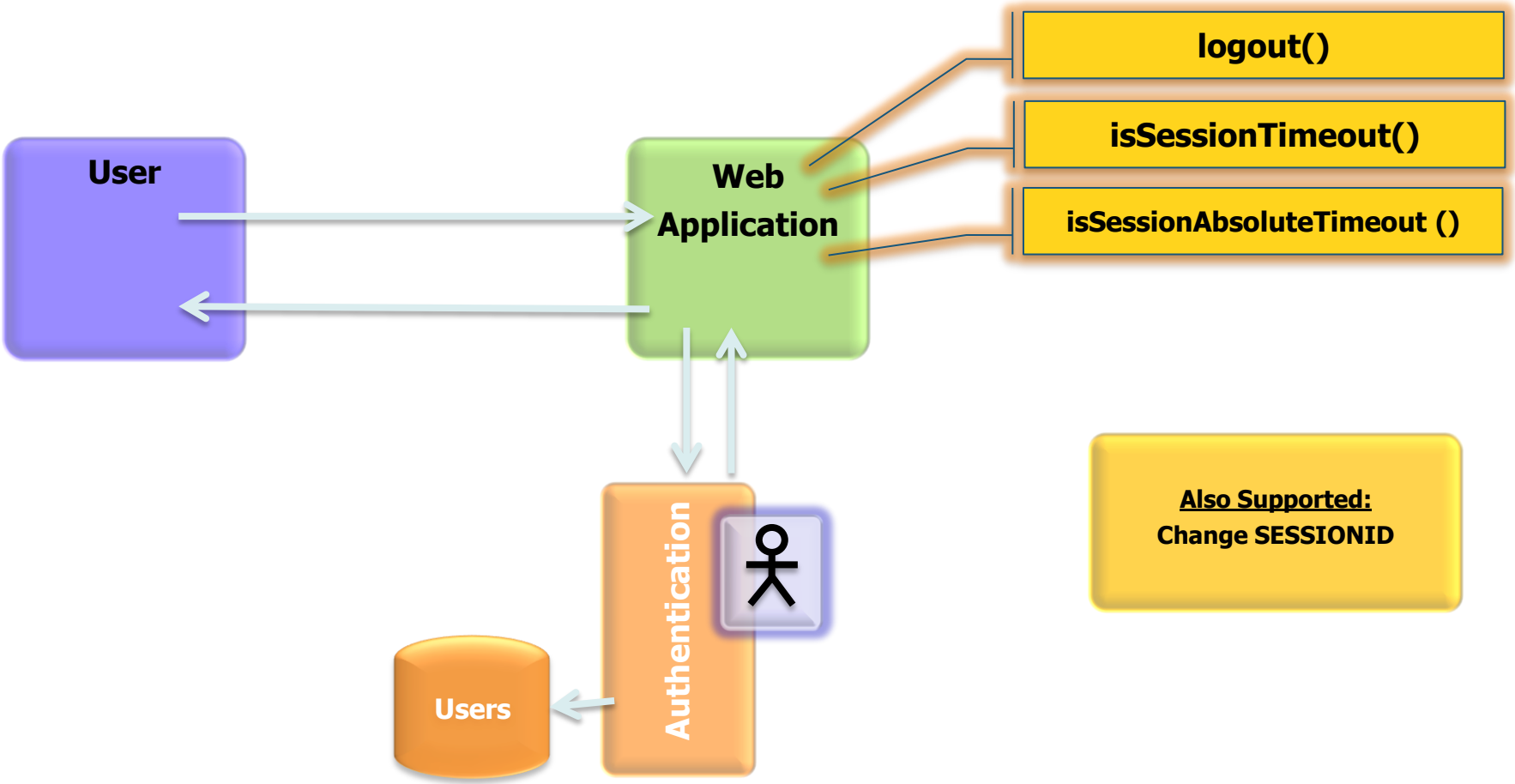
Lots more…

Applications Enjoy Attacks

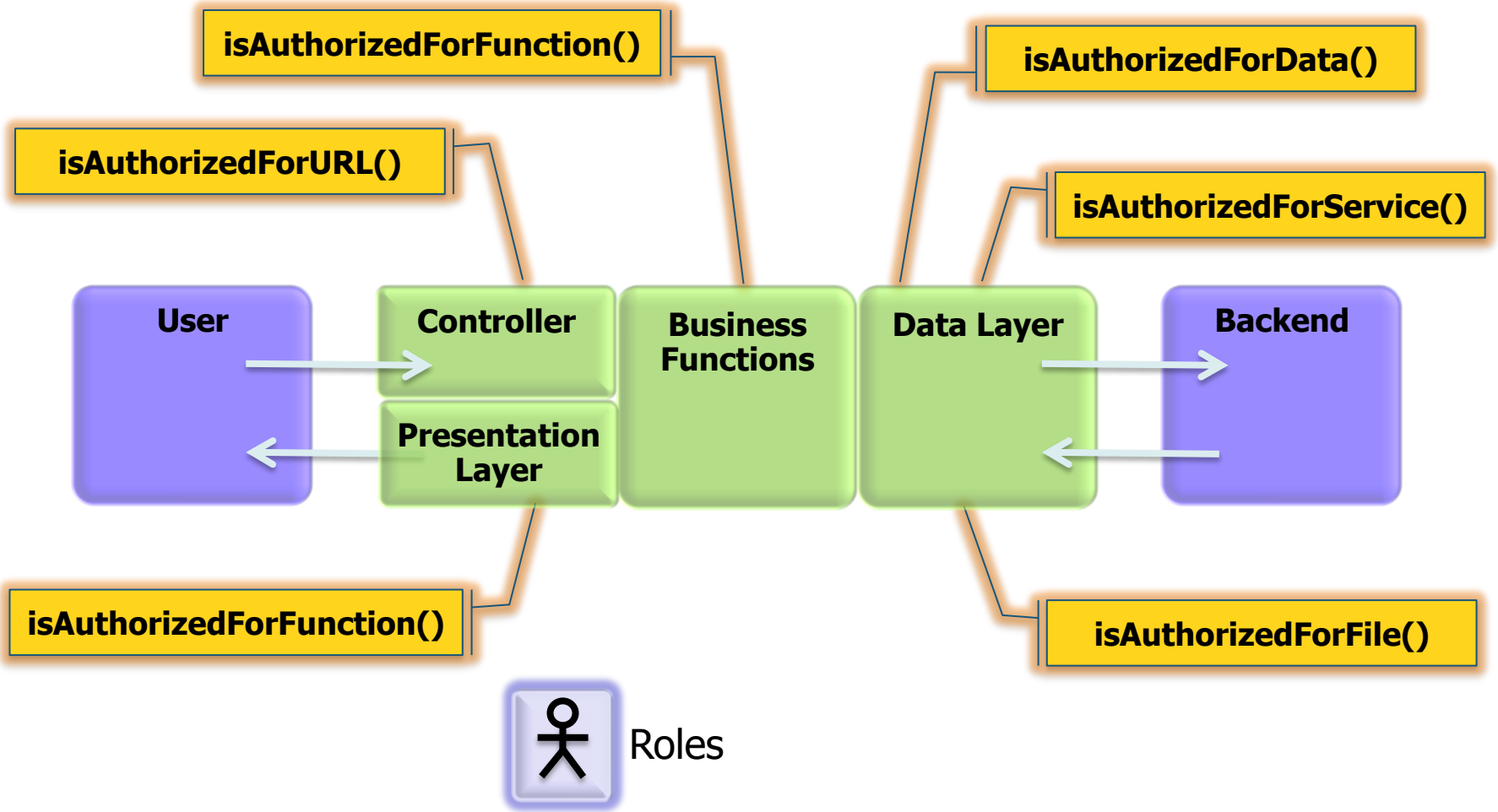
ESAPI Authentication



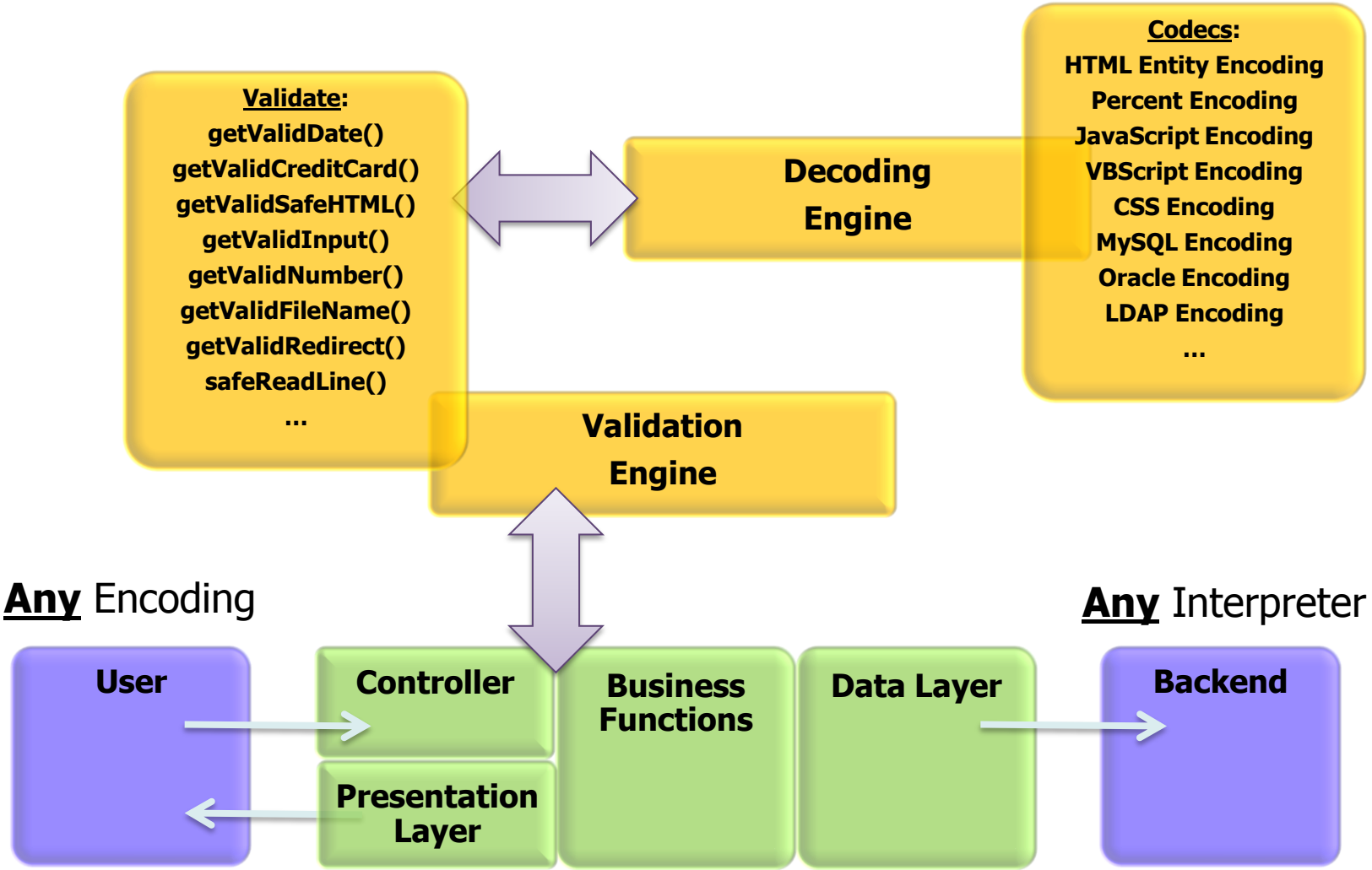
ESAPI Session Management



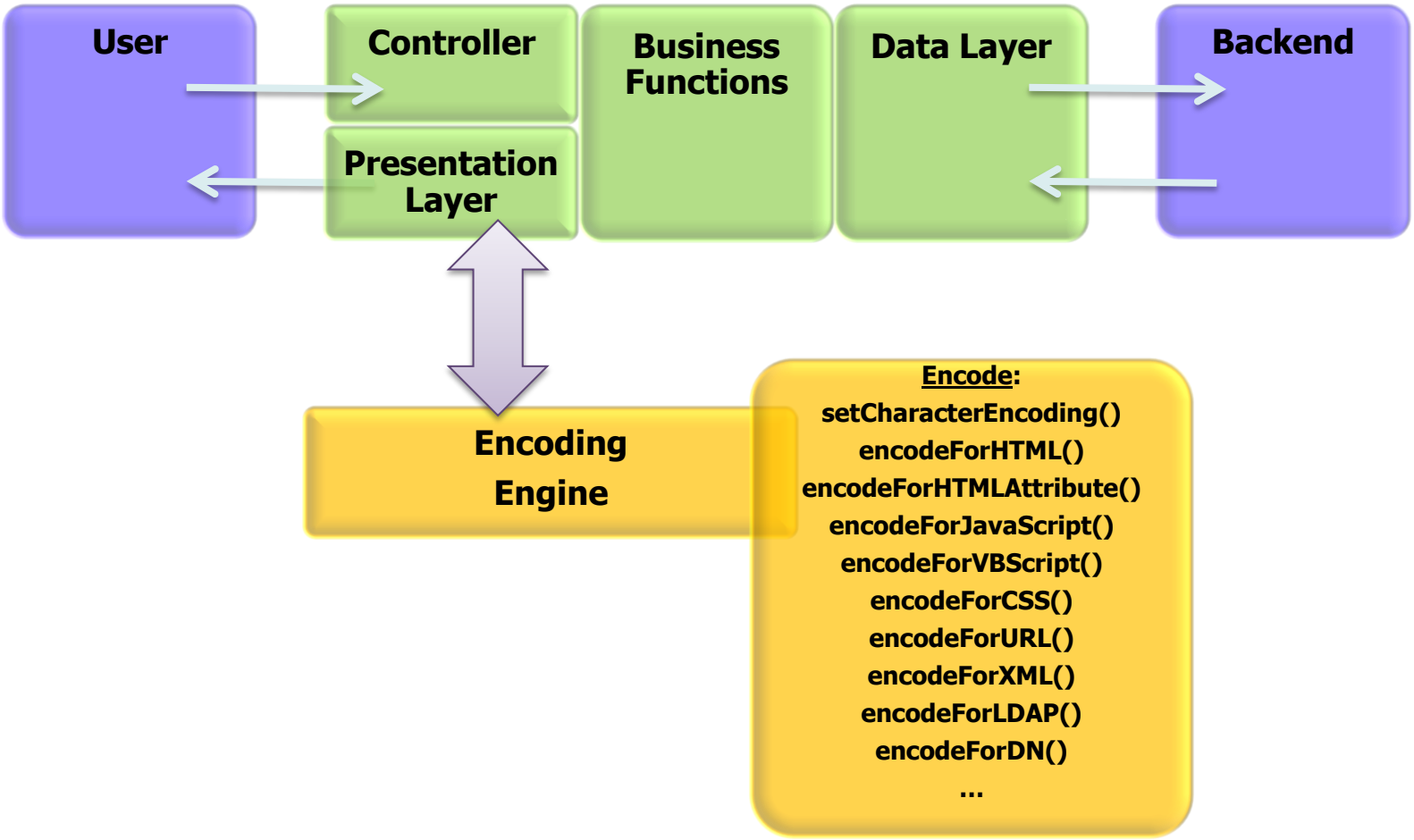
ESAPI Access Control



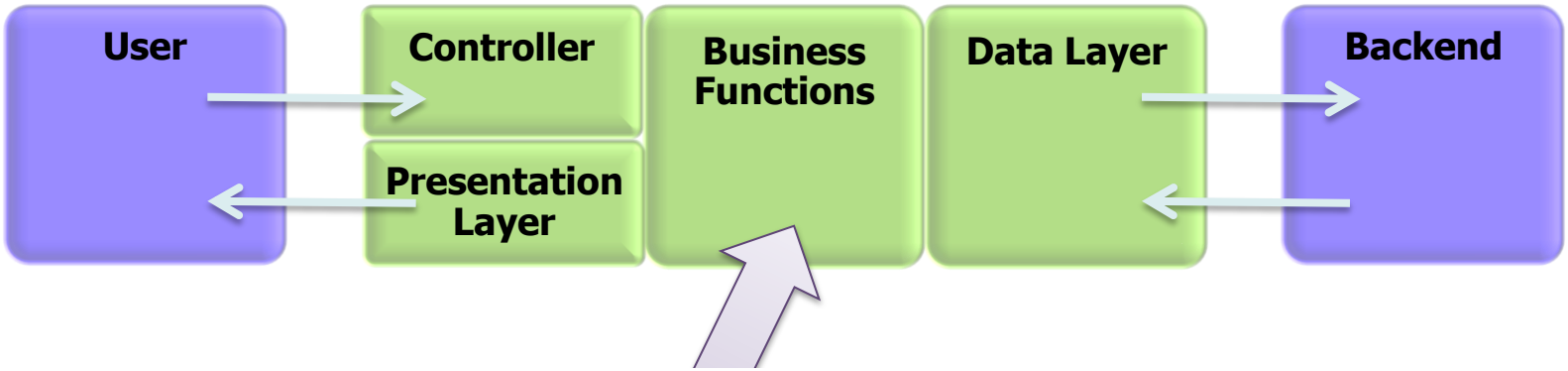
ESAPI Input Validation



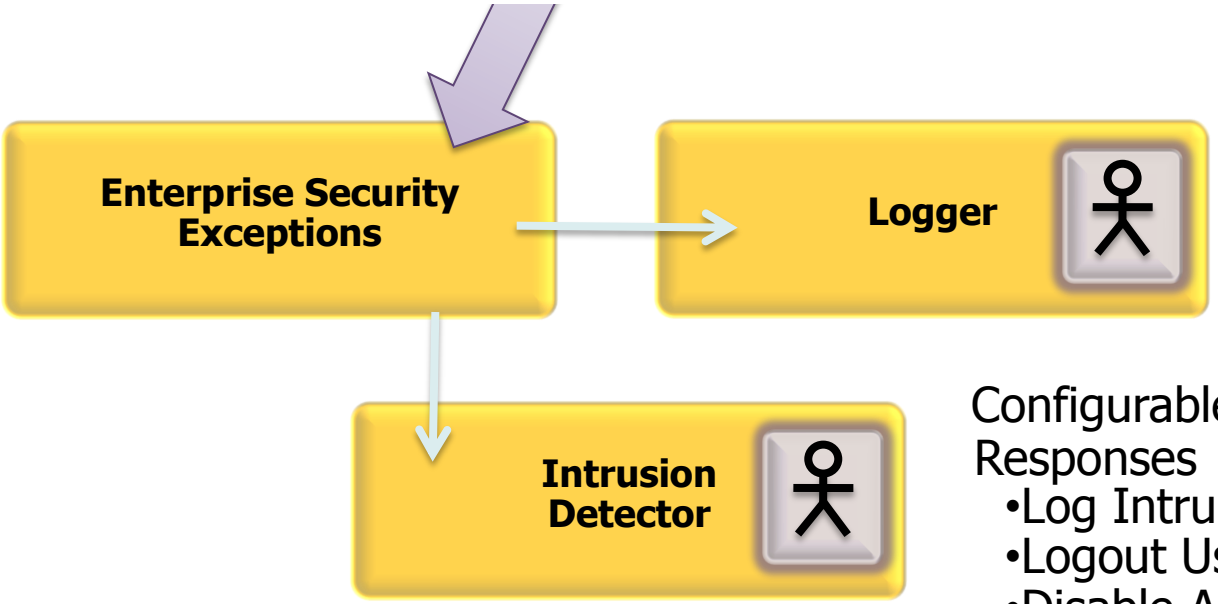
ESAPI Output Encoding



Errors, Logging, and Detection

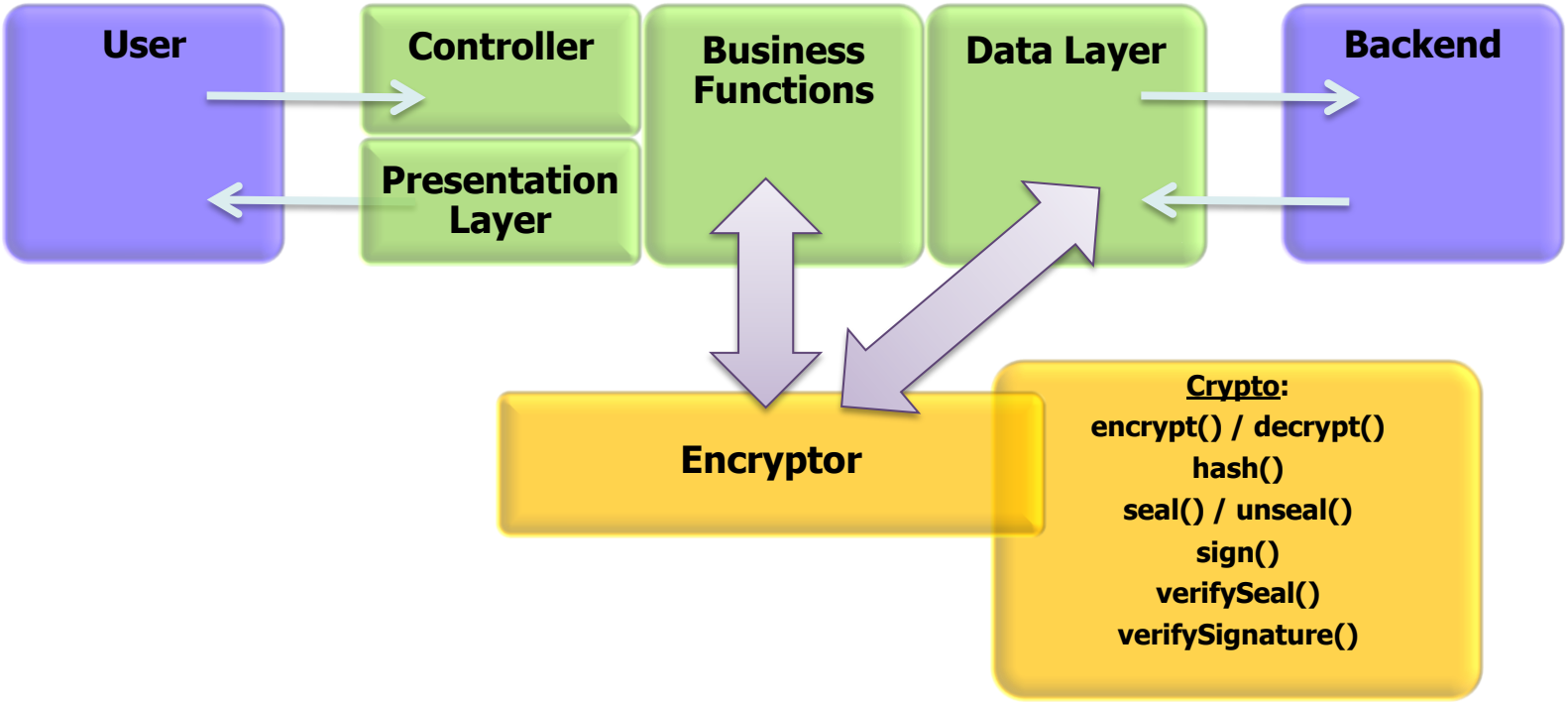


throw new ValidationException("User message", "Log message");



Configurable Thresholds
Responses
- Log Intrusion
- Logout User
- Disable Account

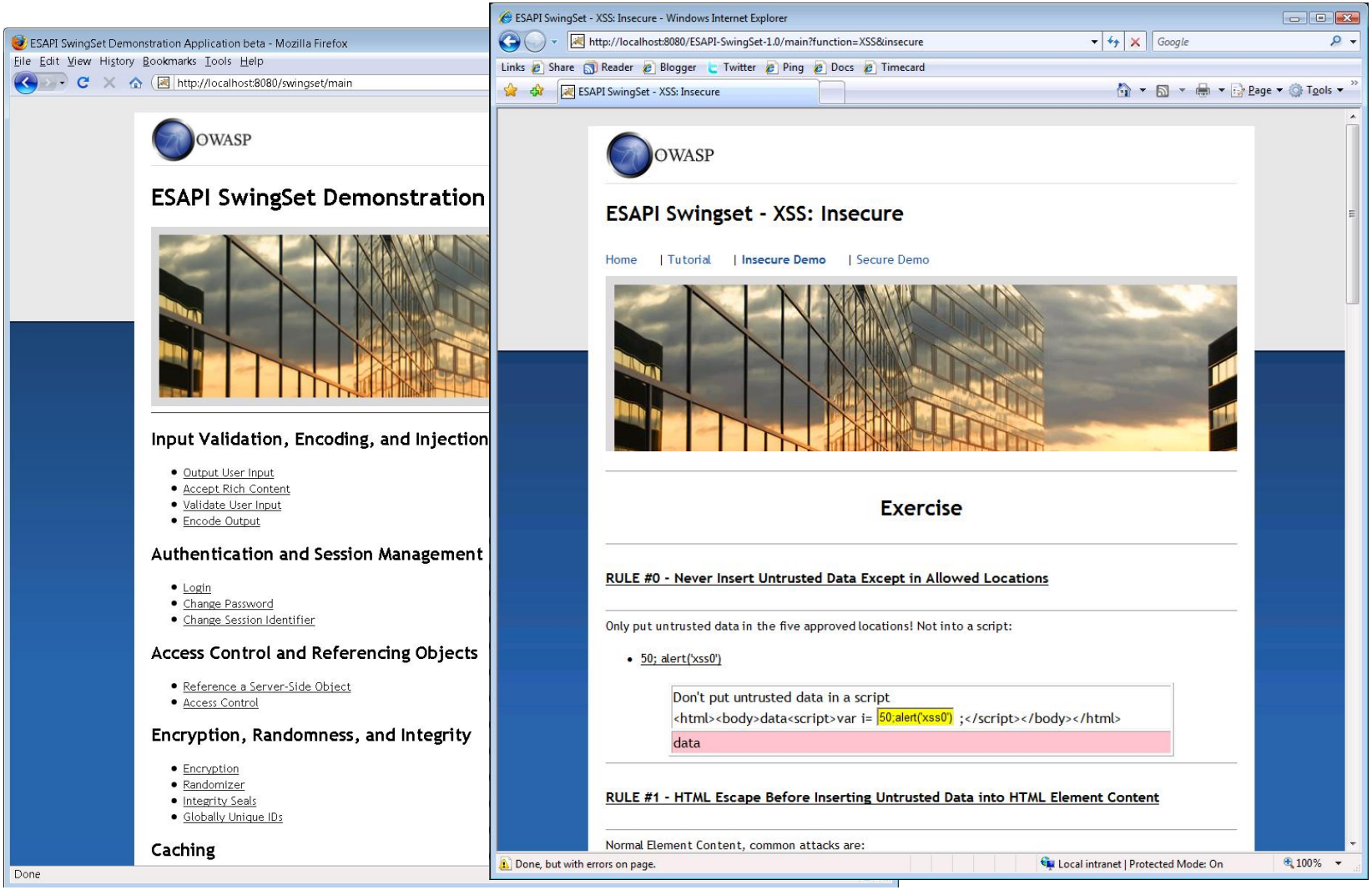
ESAPI Cryptography



Coverage

OWASP Top Ten	OWASP ESAPI
A1. Cross Site Scripting (XSS)	Validator, Encoder
A2. Injection Flaws	Encoder
A3. Malicious File Execution	HTTPUtilities (Safe Upload)
A4. Insecure Direct Object Reference	AccessReferenceMap, AccessController
A5. Cross Site Request Forgery (CSRF)	User (CSRF Token)
A6. Leakage and Improper Error Handling	EnterpriseSecurityException, HTTPUtils
A7. Broken Authentication and Sessions	Authenticator, User, HTTPUtils
A8. Insecure Cryptographic Storage	Encryptor
A9. Insecure Communications	HTTPUtilities (Secure Cookie, Channel)
A10. Failure to Restrict URL Access	AccessController

ESAPI Swingset – Sample Application



Assurance

- **Expert advisory/design/implementation team**
  - Includes security consultants, product vendors, software developers
  - Collectively reviewed over 100 million lines of code
  - Given guidance to static analysis tool vendors
  - Taught over 500 application security classes
  - Minimal and modular design/implementation

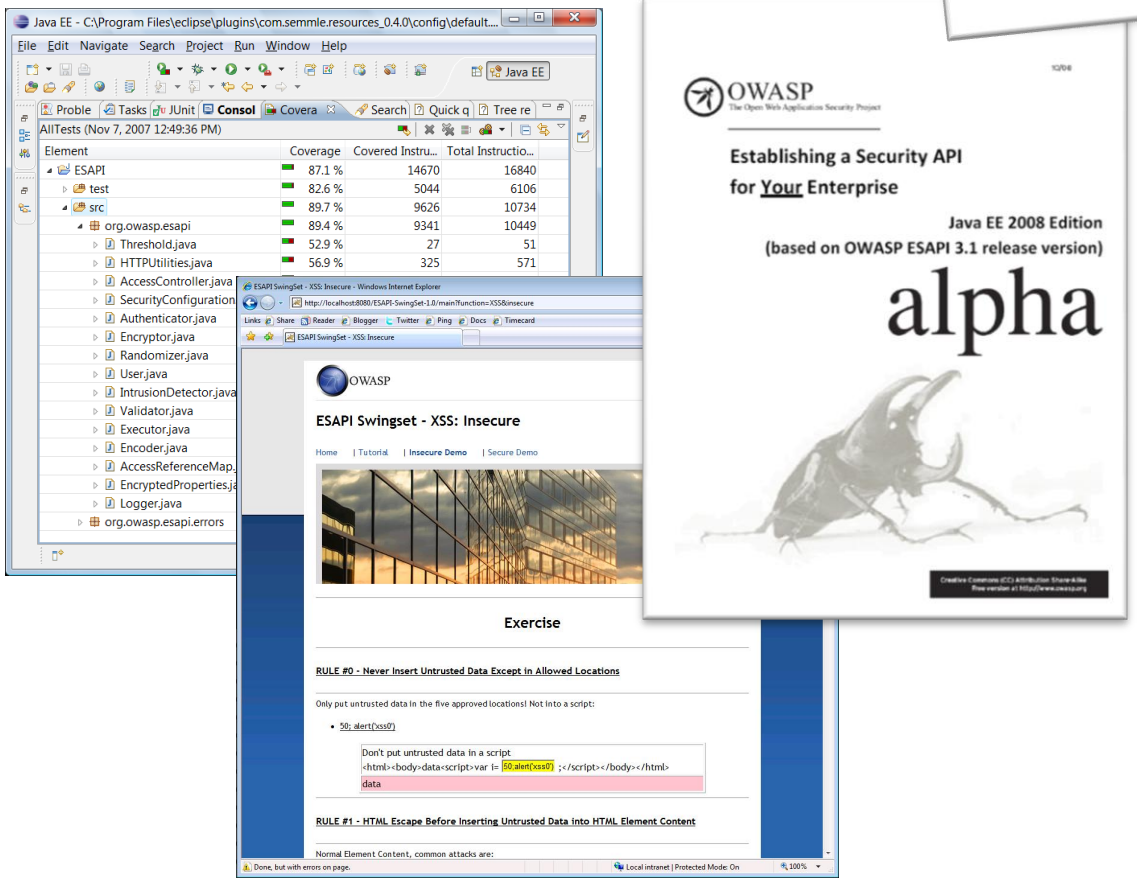
- **Tools and Testing**
  - ~700 JUnit test cases (89% coverage)
  - FindBugs, PMD, Ounce, Fortify clean
  - Code review by several Java security experts
  - Line by line review by Major Systems Integrator
  - Penetration test of sample applications
  - Full Javadoc for all functions

Major enterprises are using and evaluating ESAPI:

- Sun
- Oracle
- Dept. of Census
- Lockheed Martin
- World Bank
- American Express
- Nationwide Insurance
- …

- **Working closely with the Java Servlet Spec team at Sun**
  - They're adopting six new changes to Java EE based on ESAPI

OWASP ESAPI Project



Versions Underway

- Java EE
- .NET
- PHP
- Cold Fusion
- Classic ASP
- Haskell
- Python (starting this summer)

License

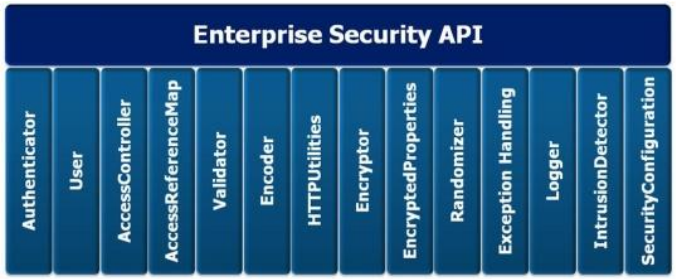
- Free and open source
- Software: BSD
- Doc: Creative Commons

Project Home Page: http://www.owasp.org/index.php/ESAPI
Code Repository: http://code.google.com/p/owasp-esapi-java

Another Opportunity for Influence

Standard Organizational Controls

OWASP (Refined and Proven)



Org 1 Enterprise Security API

. . .

Org N Enterprise Security API

Train Tool

Java

. . .

Microsoft .NET

C/C++

Standard Lang. Controls

Recognize Standard Security Controls

Code Analysis Tools

OUNCE LABS

FORTIFY SOFTWARE

VERACODE

coverity

GRAMMATECH

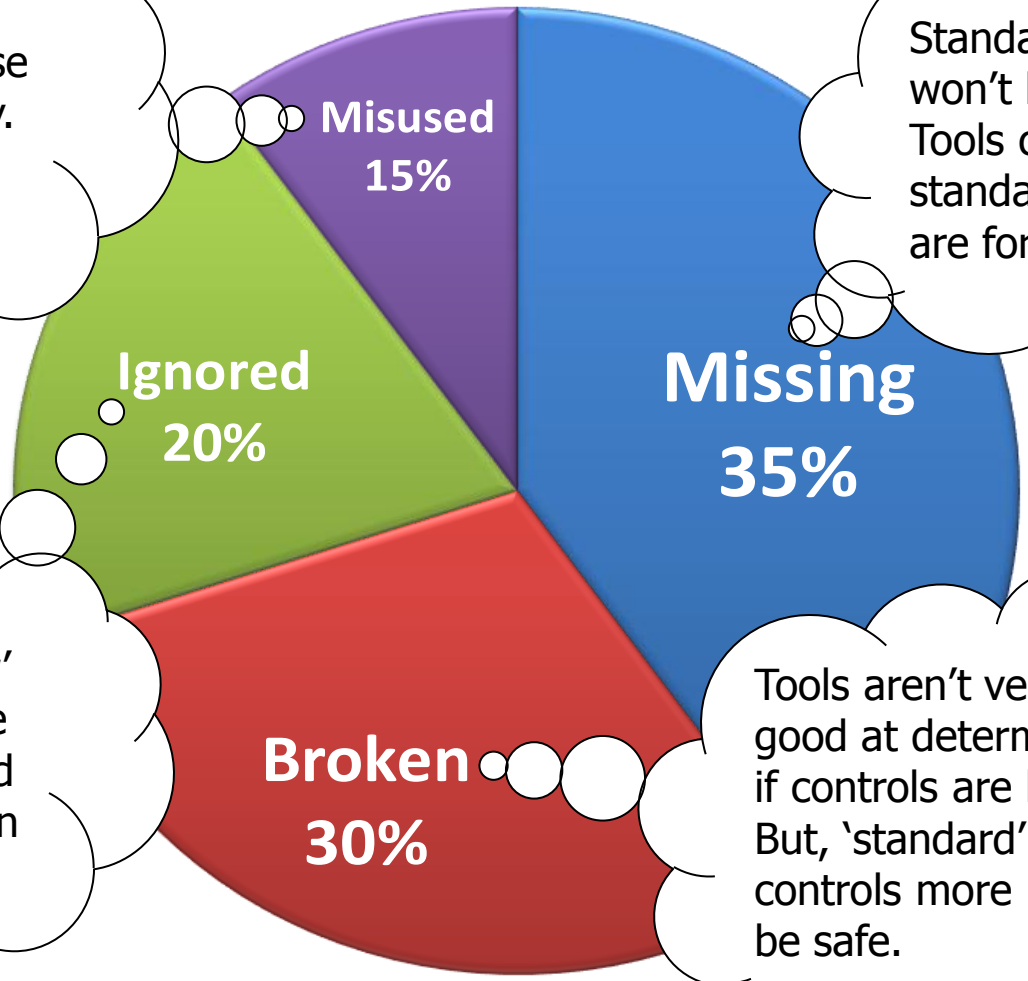
Recognize Standard Security Controls

Jakarta Struts

Spring

Standard Framework Controls

So how can an ESAPI help with Code Analysis?



Tools can be taught how to use controls correctly. They can recognize when they aren't.

Standard controls won't be 'missing'. Tools can learn what standard controls are for.

Once tools 'know' what controls are for, they are good at detecting when they aren't used.

Tools aren't very good at determining if controls are broken. But, 'standard' controls more likely to be safe.

A few 'hard' Code Analysis Problems

- **Access Control**
- **Authentication (and Session Management)**
- **Security Logging**

Access Control

- **Most (all?) code analysis tools blind to this problem. Why? Because …**
  - **they don't know what your policy is, or**
  - **the methods the app uses to enforce access control**
- **But, standard access control methods facilitate simple rules like**
  - **"Ensure the user is authorized before access to any of these resources is granted"**
    - Resources could be pages, functions, files, data, etc.
  - **Control flow analysis can verify there's a check before each access**
  - **But, tool probably can't determine if the check is correct**

Broken Authentication and Session Management

- This is a very common vulnerability area for web apps
- Almost all of the flaws are in the implementation of the security control
  - So, a standard control that is correct will simply eliminate most problems in this area
  - Control flow analysis can verify that any protected resources require authentication prior to being accessed

Security Logging

- **Most tools simply don't report that security logging is 'missing'**
  - **Today, they could detect logging, but can't tell the difference between security and functional logging**
- **ESAPI clarifies this with a method signature**

```
logger.error(Logger.SECURITY_FAILURE, "User not
 authorized to access file: " + fileName);
```

Security Event

```
logger.warning(Logger.SECURITY_SUCCESS, "Successful
 login");
```

- **We can also add a rule that indicates every `EnterpriseSecurityException()` thrown should cause a Security event to be logged**

Application Verification Challenges…

- **There is a huge range in coverage and rigor available in the application security verification market!**
- **Consumers have no way to tell the difference between someone**
  - **running a free scanner**
  - **running a high end commercial code analysis tool**
  - **doing painstaking code review and manual testing!**
  - **and anything in between!!**

OWASP Application Security Verification Standard (ASVS)

OWASP Summer of Code 2008

- **Defines 4 Verification Levels**
  - **Level 1: Automated Verification**
    - Level 1A: Dynamic Scan
    - Level 1B: Source Code Scan
  - **Level 2: Manual Verification**
    - Level 2A: Penetration Test
    - Level 2B: Code Review
  - **Level 3: Design Verification**
  - **Level 4: Internal Verification**
- **Requires Positive!! Reporting**

More Vitamins … Reporting Completeness

- **Imagine … instead of**
  - **Scan complete: No SQL Injection flaws found …**
- **It could say**
  - **147 Database calls found**
    - 62 Stored Procedural Calls
    - 81 Prepared Statements
    - 4 Dynamic Queries with all parameters escaped using: ESAPI.encodeForOracle()
- **And if they can't**
  - **how do they know the 147 calls they found are safe?**
  - **. . . I wonder how many calls they actually found??**

Software Security Facts

Reported Vulnerabilities
- 47 Total Reported
  - 15 Since version 3.0 released
  - 43 Closed

CSRF Protection
- 186 Update Requests detected
  - 152 protected with ESAPI.verifyCSRFToken()
  - 34 unprotected

Cross Site Scripting
- 1143 Locations of user input included in response
  - 262 inputs validated
  - 811 outputs HTML encoded
  - 70 Unvalidated parameters

Injection
- 147 Direct calls to Oracle found
  - 62 Stored Procedural Calls
  - 81 Prepared Statements
  - 4 Dynamic Queries with Escaping
- 68 Hibernate calls found
  - 60 Hibernate Prepared Statements
  - 8 Unprotected Dynamic Queries
- 2 Direct system calls found
  - 1 with Escaped user input
  - 1 with no user input

Developer Training	87%
Code Peer Reviewed	60%
Code Scanned	100%

Ingredients: 214,875 LOC Java, 26,350 JSPs, 37,653 HTML, Hibernate, Spring, ESAPI, 45 Oracle Stored Procedures