



Software Security Initiatives for Information Security Officers

Marco Morana
OWASP Cincinnati Chapter

OWASP

**ISSA Cincinnati
Chapter Meeting
July 14th 2010**

Copyright © 2010 - The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License.

The OWASP Foundation
<http://www.owasp.org>

Who Am I ?

- Graduated from University of Padova, Italy in 1987 with Dr. Engineering Degree
- Worked as aerospace engineer in Italy between 1990-1994
- Graduated with Master in Computer System Engineering from NPU, California in 1996
- Worked as Software Engineer in Silicon Valley
- Started my security career working at a secure email project for NASA where I developed one of the first commercial applications based upon S/MIME
- As software engineer I developed commercial security tools for ISS (SafeSuite Decisions) in 1998-2000 and for Sybase in 2002-2003
- Project managed a join-venture security start-up in Italy, Thyreaus (2001)
- Founded my own consulting company, CerbTech LLC in (2002), architected security applications for VISA (2004) and CompuCredit (2005)
- Worked as Sr. Security Software Consultant and software security instructor for McAfee/Fondstone (2005-2006)
- Joined Citigroup in 2007 as Technology Information Security Officer and founded the Cincinnati OWASP Chapter

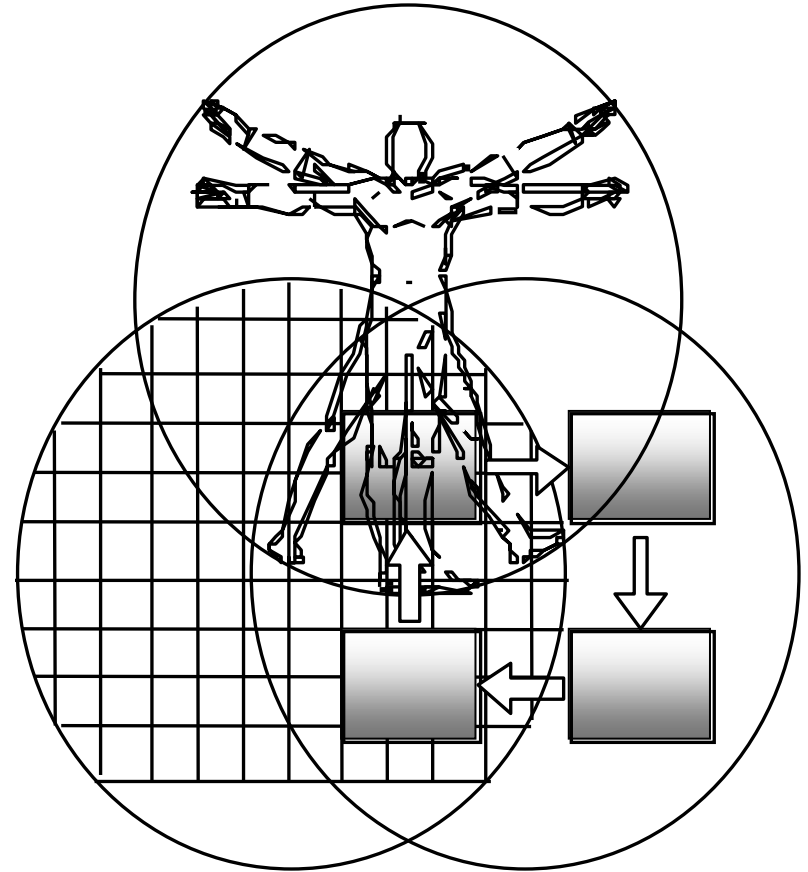
Agenda For Today's Meeting

1. Introduction to Software Security Initiatives
2. Building the Business Cases For Software Security
3. The Roadmap Toward Software Security
4. How to Integrate Security into the SDLC
5. Metrics and Measurements

Introduction to Software Security Initiatives

Software Security Initiative: People, Process, Technology

- People: Who manages software security risks
- Process: What where and how security can be build in the SDLC
- Tools: How processes can be automated



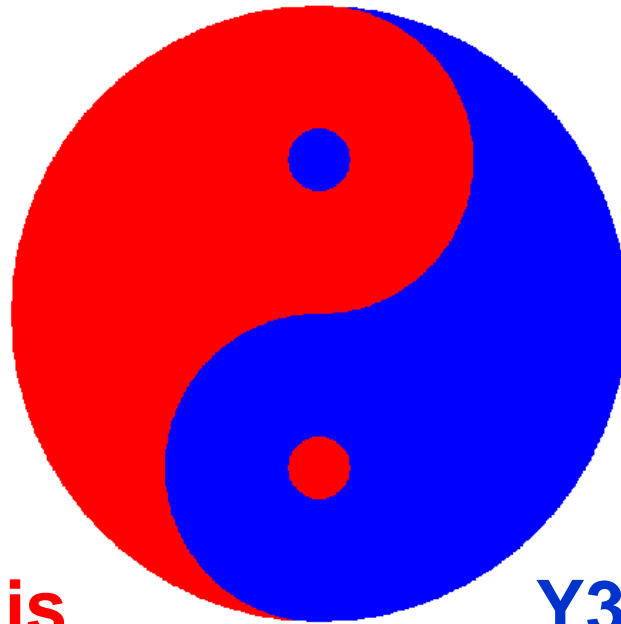
$$\textit{Security} = \textit{Commitment} * (\textit{People} + \textit{Tools} + \textit{Process}^2)$$

Application Security and Software Security

Y1: Security applied later by patching applications

Y2: Security that looks at external symptoms

Y3: Security that is reactive using SIRT or in response to audit and compliance



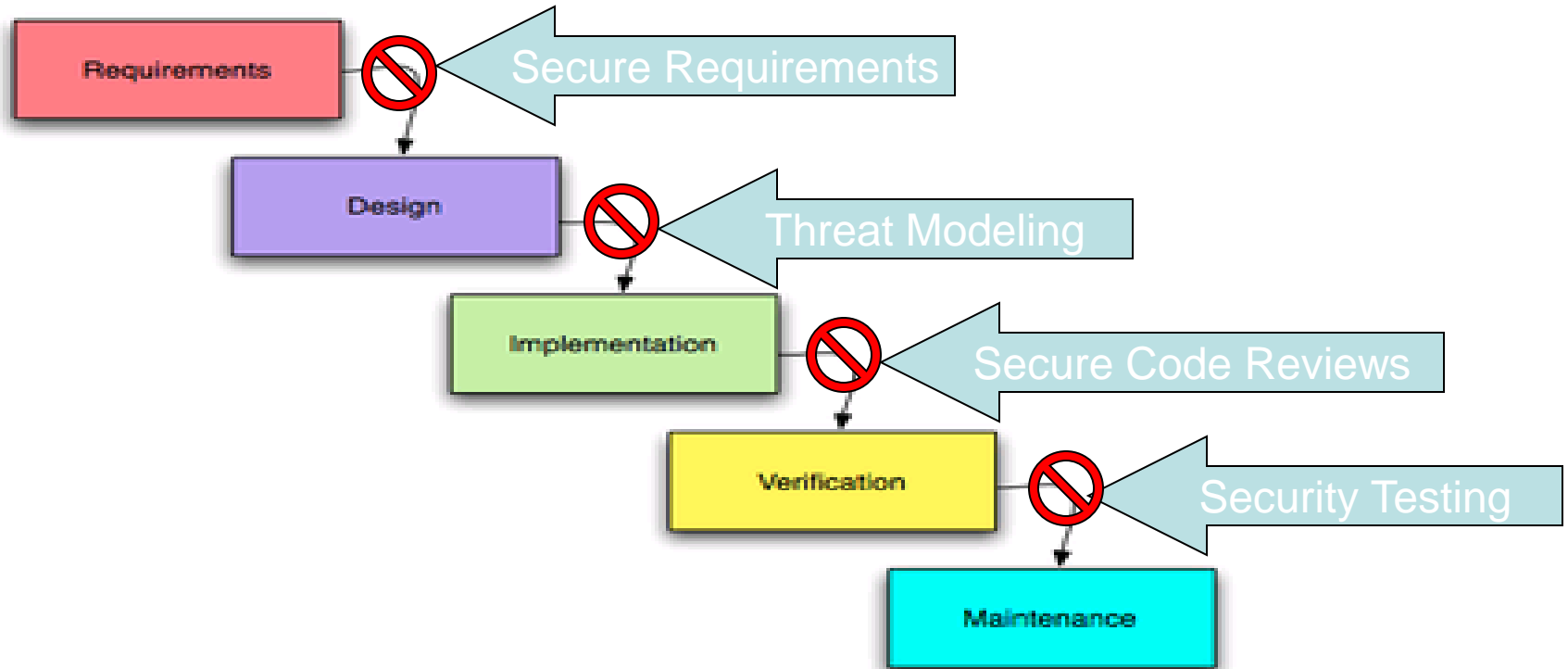
Y1: Security built into each phase of the project/SDLC

Y2: Security that looks at root causes

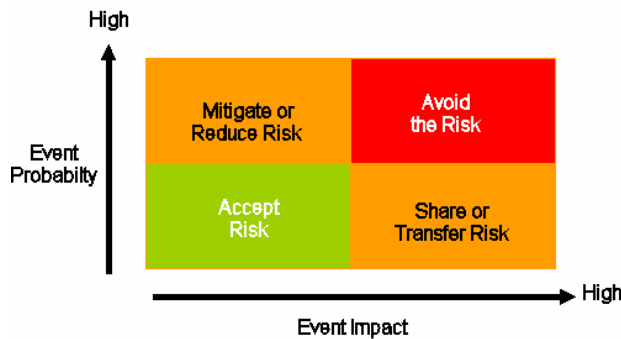
Y3: Security that is proactive using design reviews, threat analysis, defensive coding



Assurance of Software Security During the SDLC: Security Toll Gates



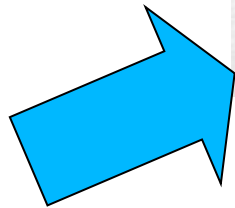
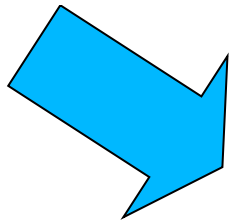
Software Security Frameworks



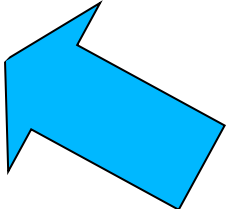
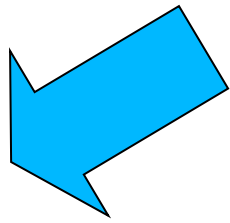
SDLC Phases	Requirements	Design	Development	Testing	Deployment and Operations		
Secure Software Best Practices	Preliminary Software Risk Analysis	Security Requirements Engineering	Security Risk-Driven Design	Secure Code Implementation	Security Tests	Security Configuration & Deployment	Secure Operations
Ongoing S-SDLC Activities Metrics and Measurements, Training, and Awareness							
S-SDLC Activities	Define Use & Misuse Cases	Define Security Requirements	Secure Architecture & Design Patterns Threat Modeling Security Test Planning Security Architecture Review	Peer Code Review Automated Static and Dynamic Code Review Security Unit Tests	Functional Test Risk Driven Tests Systems Tests White Box Testing Black Box Testing	Secure Configuration Secure Deployment	
Other Disciplines	High-Level Risk Assessments		Technical Risk Assessment				Incident Management Patch Management

Building the Business Cases For Software Security Initiatives

Four (4) Effective Business Cases Around Secure Software



```
End Sub  
End Sub  
Private Sub tbToolBar_ButtonClick  
On Error Resume Next  
timTimer.Enabled = True  
Select Case Button.Key  
Case "Back"  
    brwWebBrowser.Go  
Case "Forward"  
    brwWebBrowser.  
Case "Refresh"  
    brwWebBrows  
Case "Home"  
    brwWebBro
```



The Case #1 is about compliance with standards such as with the PCI-DSS

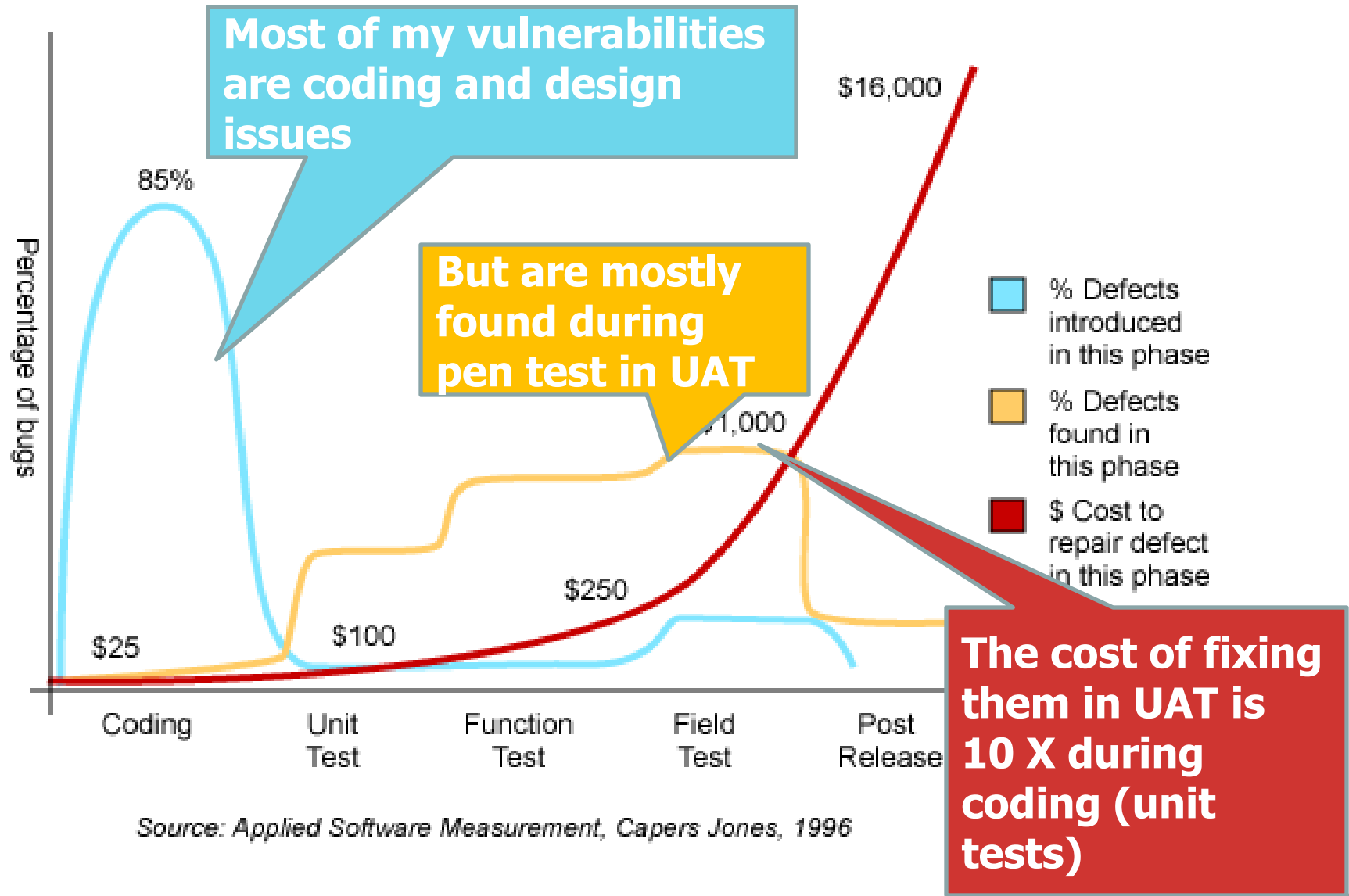
■ [PCI-DSS] 6 Develop and Maintain Secure Systems and Applications

- ▶ All vulnerabilities must be corrected.
- ▶ The application must be re-evaluated after the corrections.
- ▶ Requirement 6.6 options:
 - Manual review of application source code
 - Proper use of automated source code analyzer (scanning) tools
 - Manual web application security vulnerability assessments Proper use of automated web application security vulnerability assessment (scanning)
 - Web Application Firewall (WAF)

■ [PCI-DSS] 11 Regularly Test Security Systems and Processes

- ▶ Requirement 11.3.2: External application layer penetration test. For web applications, the tests should include, at a minimum, testing for OWASP T10 vulnerabilities

The Case #2 is about reducing the cost to manage security defects



The Case # 3 is about cybercrime attacks that exploit software vulnerabilities

The screenshot shows the Wikipedia article for Albert Gonzalez. The browser title is "Albert Gonzalez - Wikipedia, the free encyclopedia - Mozilla Firefox". The address bar shows "http://en.wikipedia.org/wiki/Albert_Gonzalez". The article text includes: "Albert Gonzalez (born 1981) is a computer hacker and computer criminal who is accused of masterminding the combined credit card theft and subsequent reselling of more than 170 million card and ATM numbers from 2005 through 2007—the biggest such fraud in history." and "Gonzalez and his accomplices used SQL injection and packet sniffer malware software to create backdoors to several corporate systems... steal computer data". A photo of Albert Gonzalez is shown on the right with the caption "Photo of Albert Gonzalez by U.S. Secret Service".

170 million card and ATM numbers

Exploited application vulnerabilities such as SQL injection and uploaded sniffers

The Case #4 is about following what the analysts say about software security



- 1) "75% of security breaches happen at the application"-
- 2) "Over 70 percent of security vulnerabilities exist at the application layer, not the network layer"
- 3) "If only 50 percent of software vulnerabilities were removed prior to production ... costs would be reduced by 75 percent"

1,2,3 Sources: Gartner



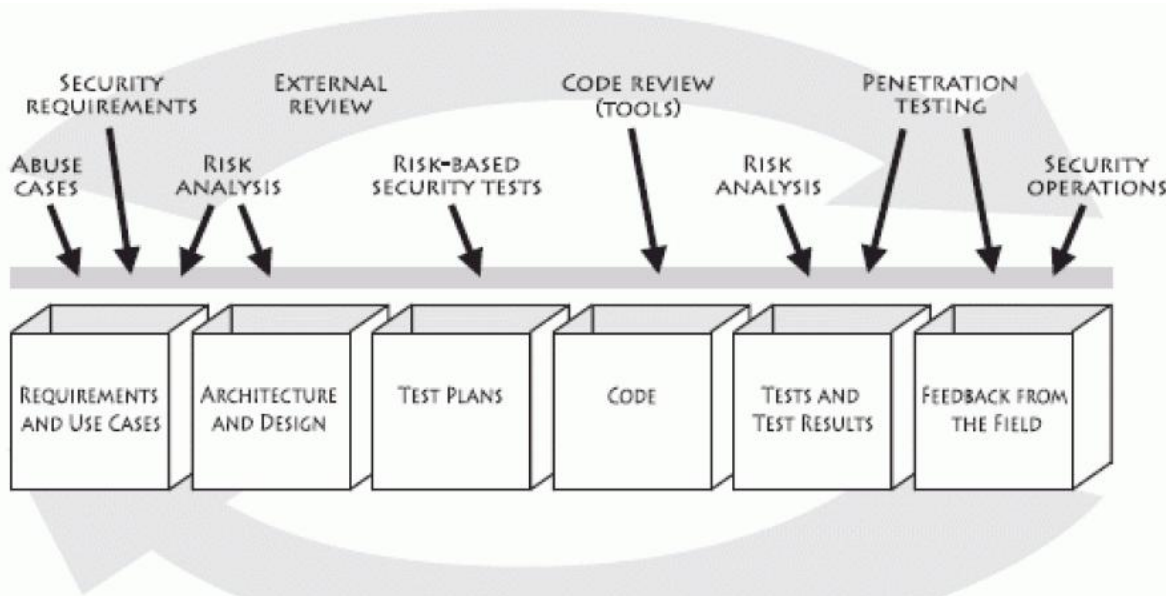
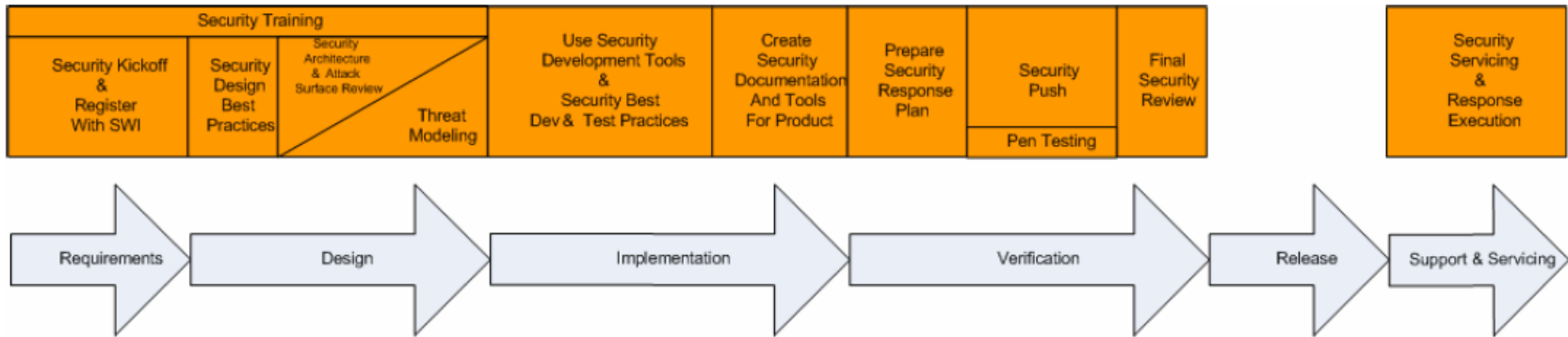
The Roadmap Toward Software Security

A Feasible Plan For Software Security

Initiative in 4 steps:

- 1. Assess the maturity level** of the software security processes within your organization/company
- 2. Start by introducing software security activities as part of the SDLC**
 1. Security Requirements
 2. Secure Design Reviews and Threat Modeling
 3. Static Code Analysis and Secure Code Reviews
 4. Security Testing
- 3. Measure and manage vulnerabilities and software security risks**
- 4. Integrate software security processes with other information security and risk management processes**

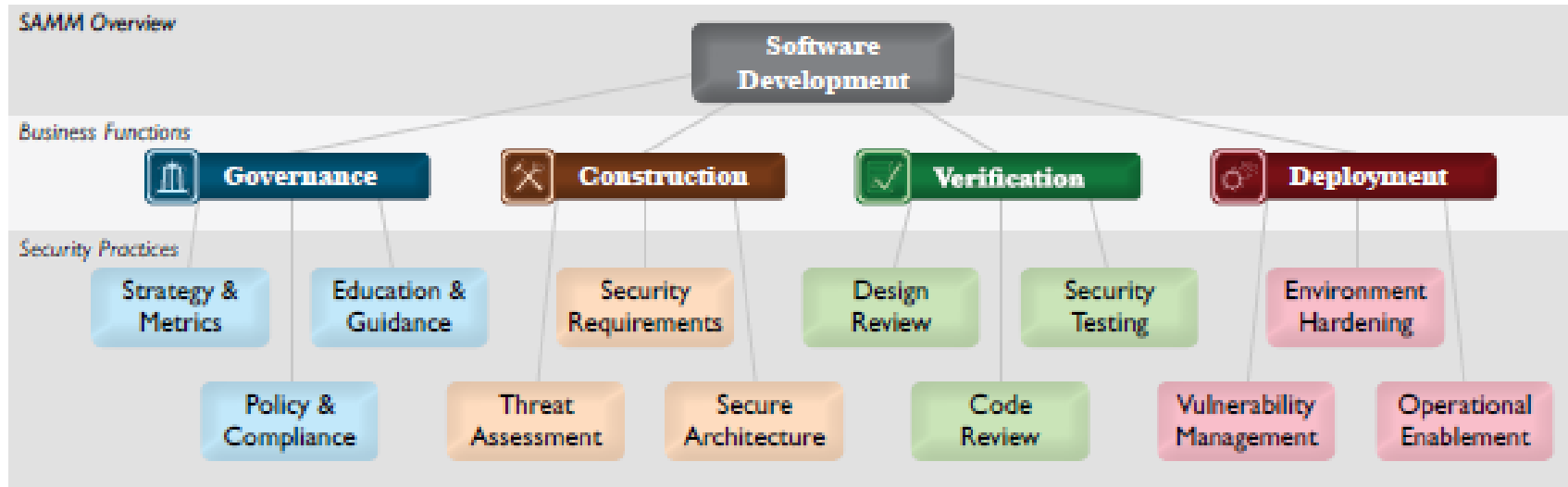
Old School Security-enhanced lifecycle process (S-SDLC): MS-SDL, Digital TP and CLASP



CLASP BEST PRACTICES

- 1) Institute awareness programs
- 2) Perform application assessments
- 3) Capture security requirements
- 4) Implement secure development practices
- 5) Build vulnerability remediation procedures
- 6) Define & monitor metrics
- 7) Publish operational security guidelines

New School Standard Software Security Maturity Models: SAMM, BSIMM



The Software Security Framework (SSF)			
Governance	Intelligence	SSDL Touchpoints	Deployment
Strategy and Metrics	Attack Models	Architecture Analysis	Penetration Testing
Compliance and Policy	Security Features and Design	Code Review	Software Environment
Training	Standards and Requirements	Security Testing	Configuration Management and Vulnerability Management

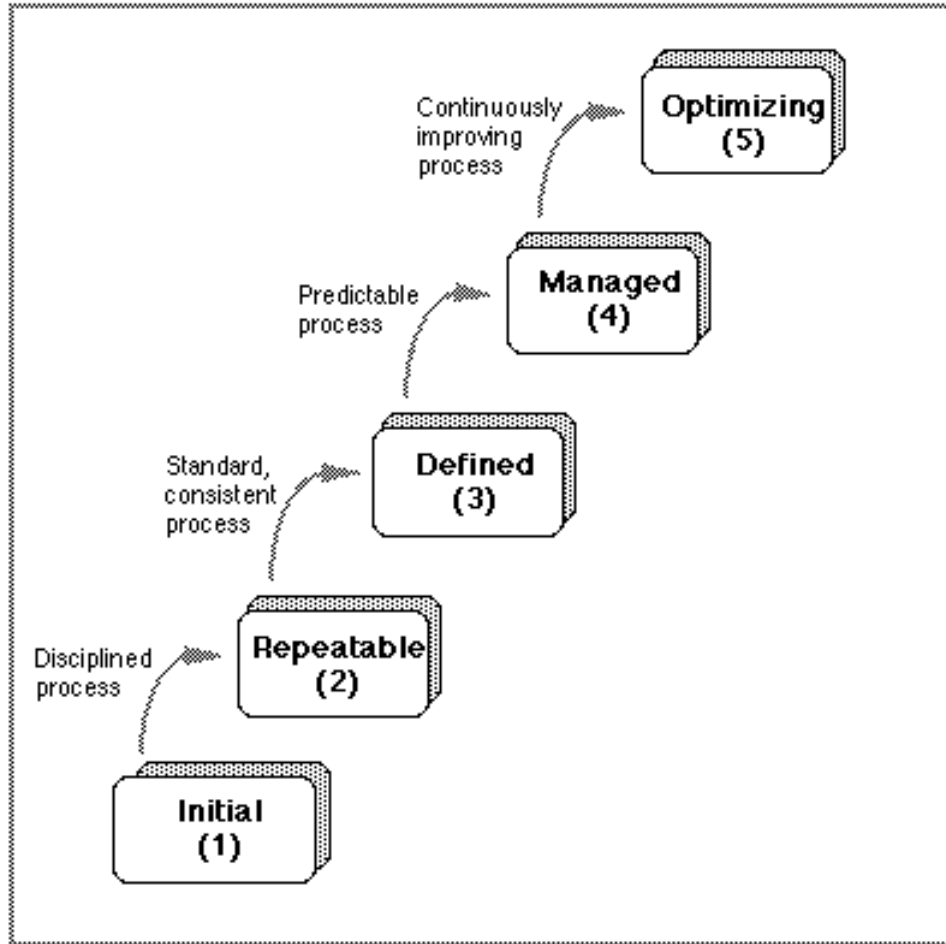


Code Review Activities And Capability Levels: BSIMM

SSDL TOUCHPOINTS: CODE REVIEW			
Use of code review tools, development of customized rules, profiles for tool use by different roles, manual analysis, ranking/measuring results.			
	Objective	Activity	Level
CR1.1	know which bugs matter to you	create top N bugs list (real data preferred) (T: training)	1
CR1.2	review high-risk applications opportunistically	have SSG perform ad hoc review	
CR1.3	spread software security around without any process	establish coding labs or of review	
CR2.1	drive efficiency/consistency with automation	use automated tools along with manual review	2
CR2.2	drive behavior objectively	enforce coding standards	
CR2.3	find bugs earlier	make code review mandatory for all projects	
CR2.4	know which bugs matter (for training)	use centralized reporting (close knowledge loop, drive training) (T: strategy/metrics)	
CR2.5	make most efficient use of tools	assign tool mentors	
CR3.1	drive efficiency/reduce false positives	use automated tools with tailored rules	3
CR3.2	combine assessment techniques	build a factory	
CR3.3	handle new bug classes in an already scanned codebase	build capability for eradicating specific bugs from entire codebase	

I am here

Capability Maturity Model Levels



Software Security Maturity Stages and Levels

■ **Maturity Innocence (CMM 0-1)**

- ▶ No formal security requirements
- ▶ Issues addressed with penetration testing and incidents
- ▶ Penetrate and patch and reactive approach

■ **Maturity Awareness (CMM 2-3)**

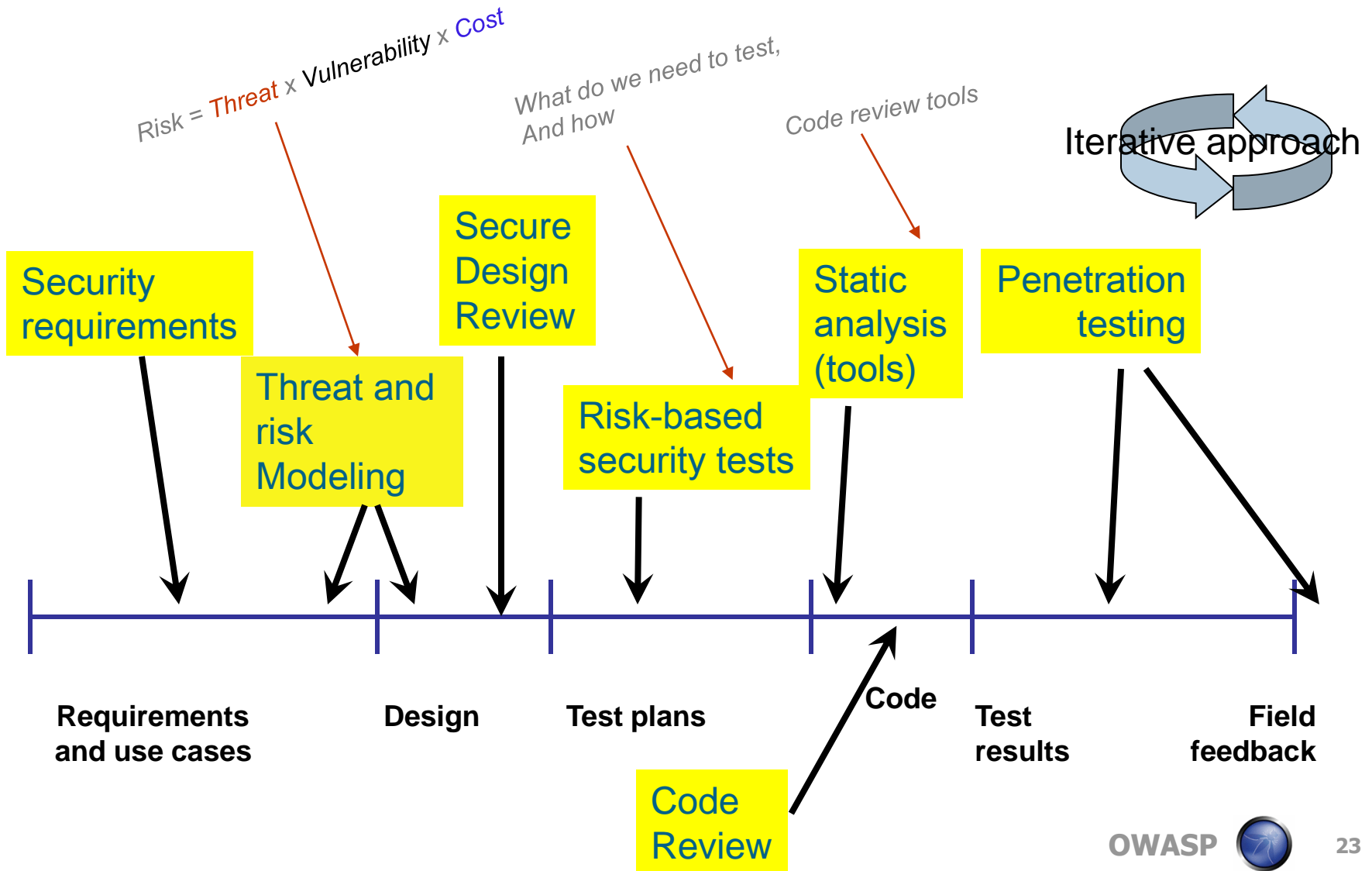
- ▶ All applications have penetration tests done before going into production
- ▶ Secure coding standards are adopted as well as source code reviews

■ **Maturity Enlightenment (CCM 4-5)**

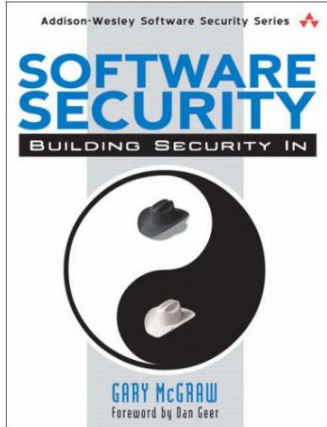
- ▶ Threat analysis in each phase of the SDLC
- ▶ Risk metrics and vulnerability measurements are used for security activity decision making

How to Integrate Security Activities into the SDLC

S-SDLC Security Tollgates



A Prerequisite For A Successful Software Security Program is to Acquire People with the Right Skills



The Initial Step Toward Software Security : From Black Box To White Box Testing

**Manual
Penetration
Testing**

**Manual
Code
Review**

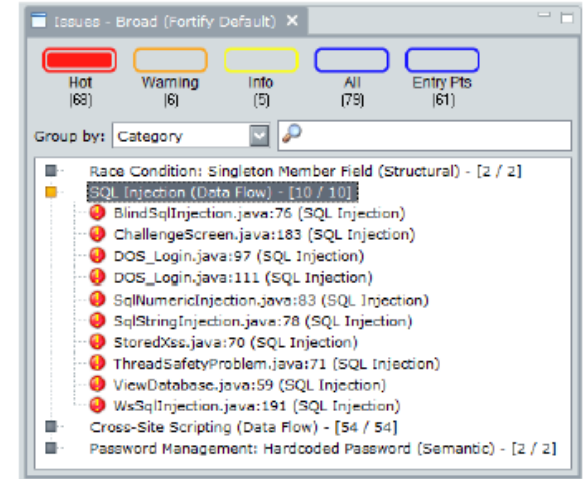
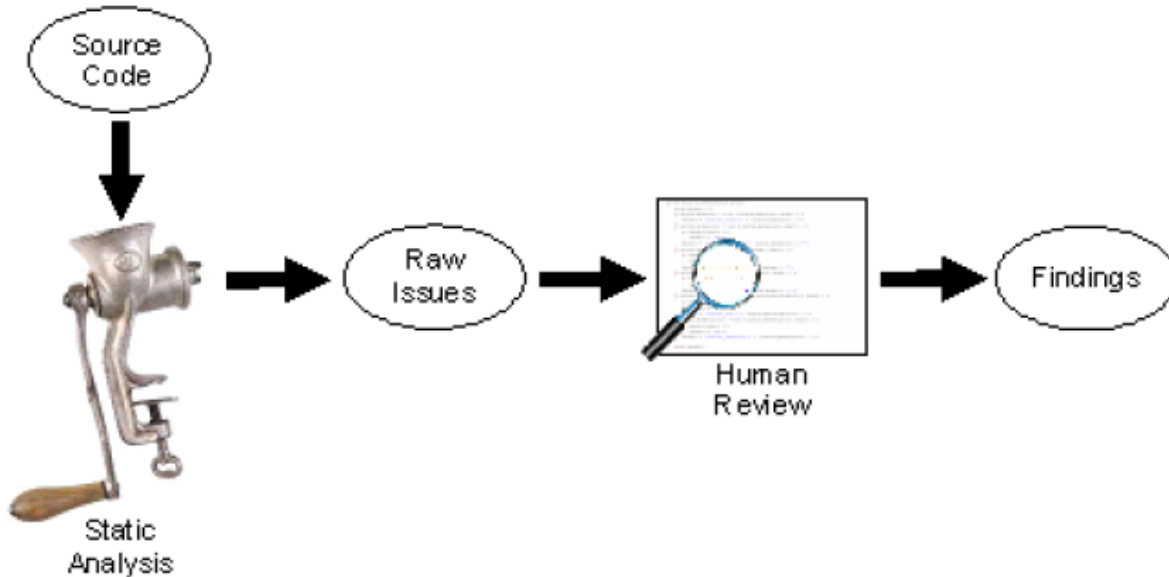
**Automated
Vulnerability
Scanning**

```
166 // check if the user wants to remember username
167 String rememberUsername = hreq.getParameter("rememberUsername");
168 if (rememberUsername != null) {
169     // set a cookie with the username in it
170     Cookie userNameCookie = new Cookie("Cookie", rememberUsername);
171     // set cookie to last for one month
172     userNameCookie.setMaxAge(2678400);
173     hres.addCookie(userNameCookie);
174 } else {
175     // see if the cookie exists and remove it
176     Cookie[] cookies = hreq.getCookies();
177     if (cookies != null) {
178         for (int loop=0; loop < cookies.length; loop++) {
179             if (cookies[loop].getName().equals("Cookie")) {
180                 cookies[loop].setMaxAge(0);
181                 hres.addCookie(cookies[loop]);
182             }
183         }
184     }
185 }
186 // validate against the registered users
187 boolean authenticated = false;
188 SignOnLocal signOn = getSignOnEjb();
189 boolean authenticated = signOn.authenticate(userName, password);
190 if (authenticated) {
191     // place a true boolean in the session
192     if (hreq.getSession().getAttribute("authenticated") == null) {
193         hreq.getSession().setAttribute("authenticated", true);
194     }
195     hreq.getSession().setAttribute("USER_NAME", userName);
196     // remove the sign on user last before
197     signOn.removeSignOnUser(userName);
198 }
```

**Automated
Static Code
Analysis**



Automated Source Code Analysis



```
66 try
67 {
68     Statement answer_statement = connection.createStatement( ResultSet.TYPE_SCROLL_INSENSITIVE,
69     ResultSet.CONCUR_READ_ONLY );
70     ResultSet answer_results = answer_statement.executeQuery( answer_query );
71     answer_results.first();
72     if( accountNumber.toString().equals( answer_results.getString(1) ) ) {
73         makeSuccess( s );
74     } else {
75         Statement statement = connection.createStatement( ResultSet.TYPE_SCROLL_INSENSITIVE,
76         ResultSet.CONCUR_READ_ONLY );
77         ResultSet results = statement.executeQuery( query );
78
79         if ( ( results != null ) && ( results.first() ) == true ) {
80             ec.addElement( new P().addElement("Account number is valid"));
81         } else {
82             ec.addElement( new P().addElement("Invalid account number"));
83         }
84     }
85 }
86 }
87 }
```

```
Summary - Issues X
ABSTRACT
Constructing a dynamic SQL statement with user input may allow an attacker to modify the statement's
meaning or to execute arbitrary SQL commands.

EXPLANATION
SQL Injection errors occur when:

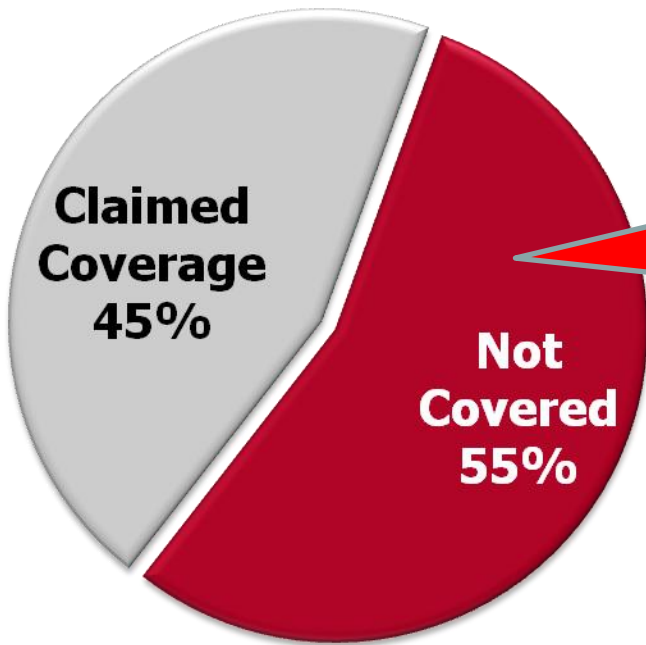
1. Data enters a program from an untrusted source.
In this case the data enters at getParameter(values) in src/session/ParameterParser.java at line 590.

2. The data is used to dynamically construct a SQL query.
In this case the data is passed to executeQuery() in src/lessons/BlindSqlInjection.java at line 76.

Example 1: The following code dynamically constructs and executes a SQL query that searches for items
matching a specified name. The query restricts the items displayed to those where the owner matches the
user name of the currently-authenticated user.

String username = ctx.getAuthenticatedUsername();
String itemname = request.getParameter("itemname");
String query = "SELECT * FROM items WHERE owner = "
+ username + " AND itemname = "
+ itemname + ";";
ResultSet rs = stmt.executeQuery(query);
...
}
```

Security Tools Coverage

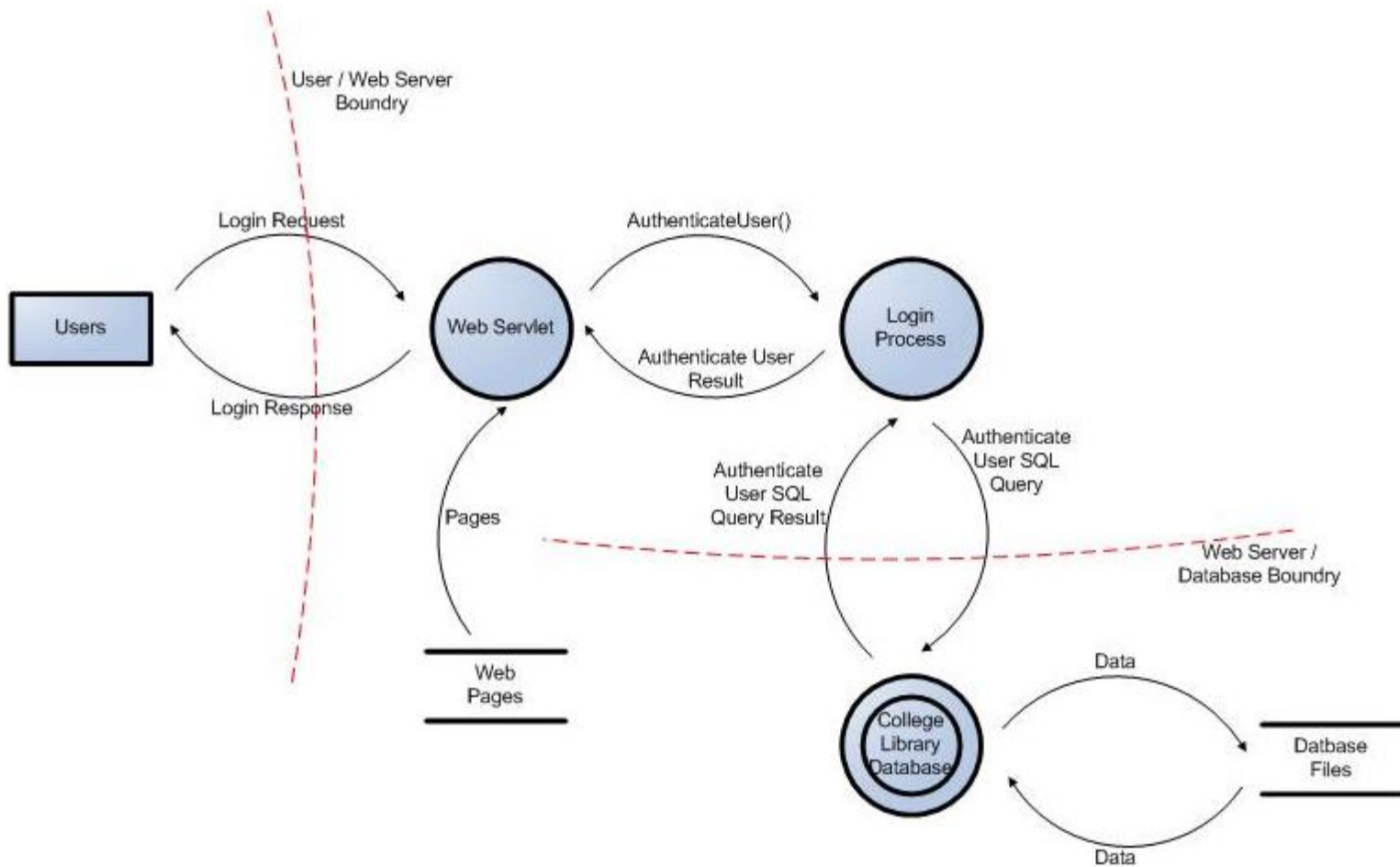


Beware of the silver bullet security mentality and false sense of security given by tools !

- They found very little overlap between tools, so to get 45% you need them all (assuming their claims are true)

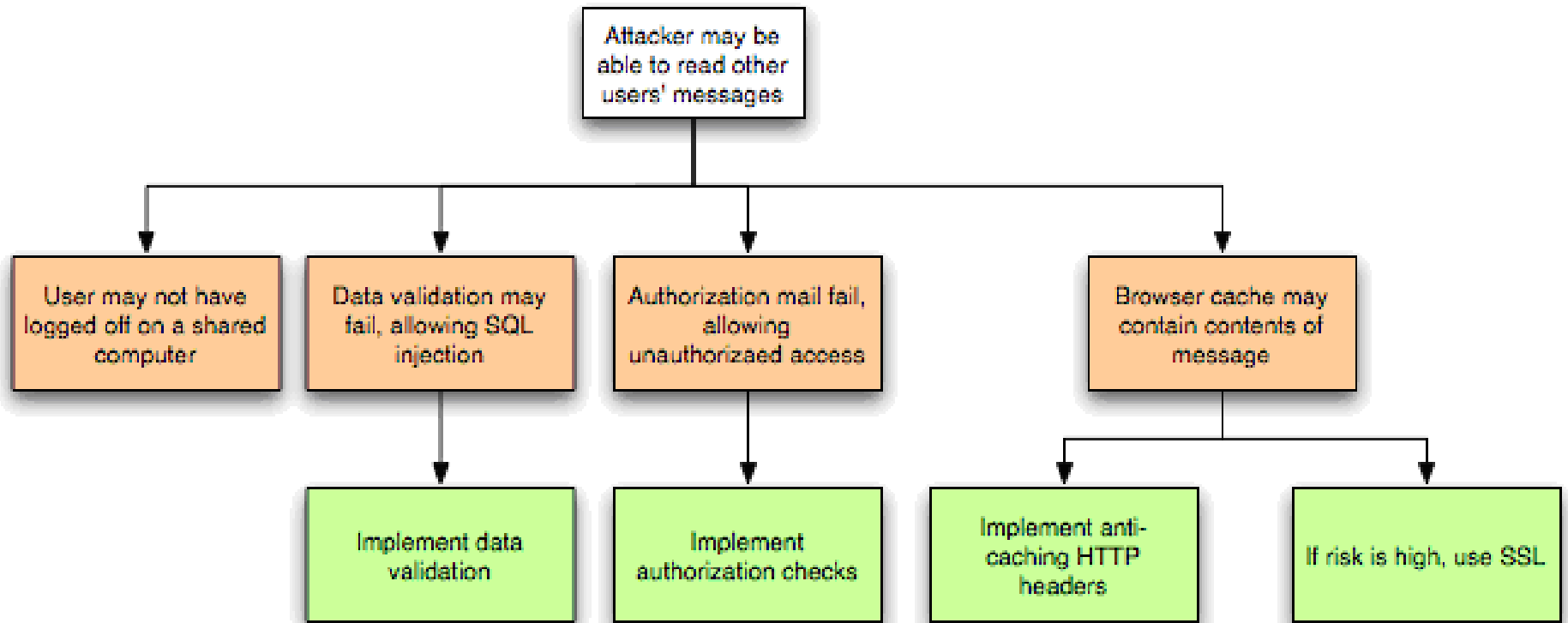


Application Threat Modeling: Data Flow Diagrams

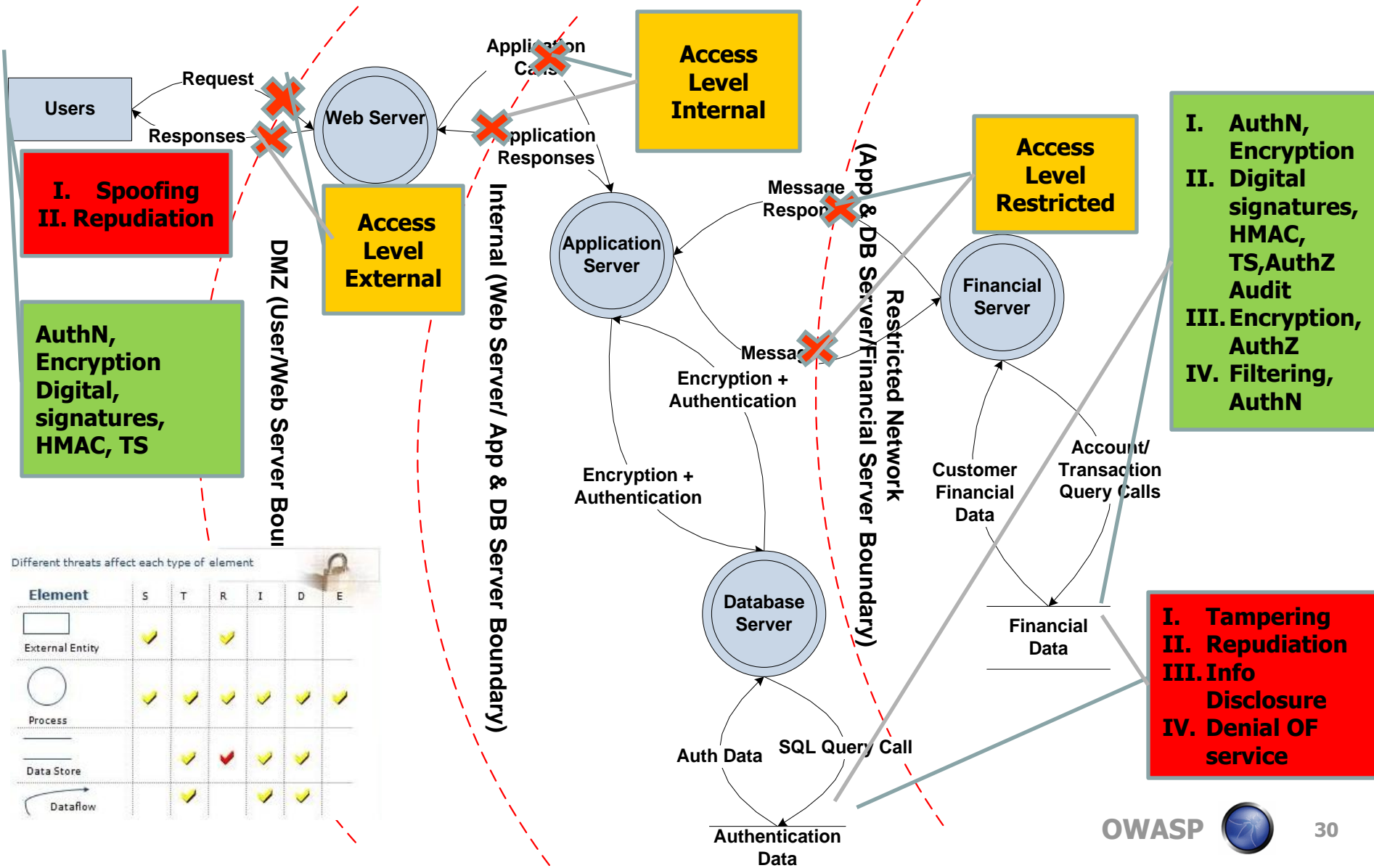


https://www.owasp.org/index.php/Application_Threat_Modeling

Threats, Vulnerabilities and Countermeasures



The Holistic Step: Application Threat Modeling



Software Risk Analysis

■ Evaluate The Risk Factors Of Software:

- Threat (e.g. the cause)
- Vulnerability (e.g. the application weakness)
- Technical Impact (e.g. the loss of service/data)
- Business Impact (e.g. financial loss, fraud, unlawful compliance etc)

■ Calculate The Overall Risk on Insecure Software:

- Qualitative: Likelihood x Impact (H, M, L)
- Quantitative: $ALE = SLE \times ARO$
- Threat Source (STRIDE) x Severity (DREAD)
- Threat X Vulnerability X Impact (OWASP)

Security Requirements Definition

- Include both *functional requirements* for security controls and *risk derived requirements* from the abuse case scenarios
- Define Security Requirements in Standards
 - ▶ Which controls are required (e.g. authentication, authorization, encryption etc)
 - ▶ Where should be implemented (e.g. design, source code, application, server)
 - ▶ Why are required
 - Compliance and auditing (e.g. FFIEC, PCI, SOX etc.)
 - Mitigation for known threats (e.g. STRIDE)
 - ▶ How should be implemented and tested

Risk Driven Security Requirements: Use and Misuse Cases



Requirements Driven Security Testing

■ The OWASP Testing Guide

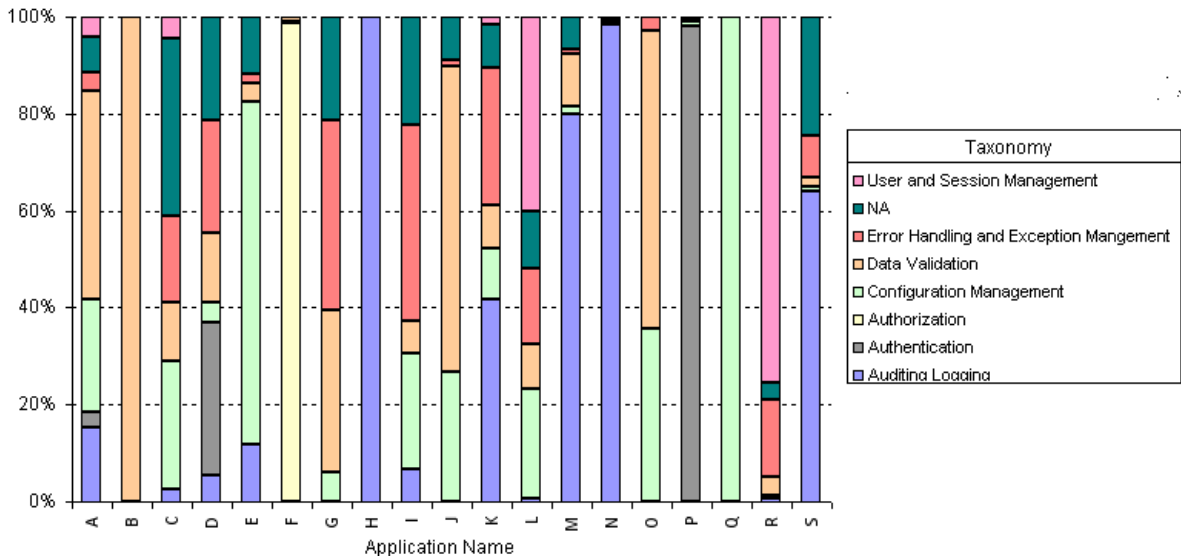
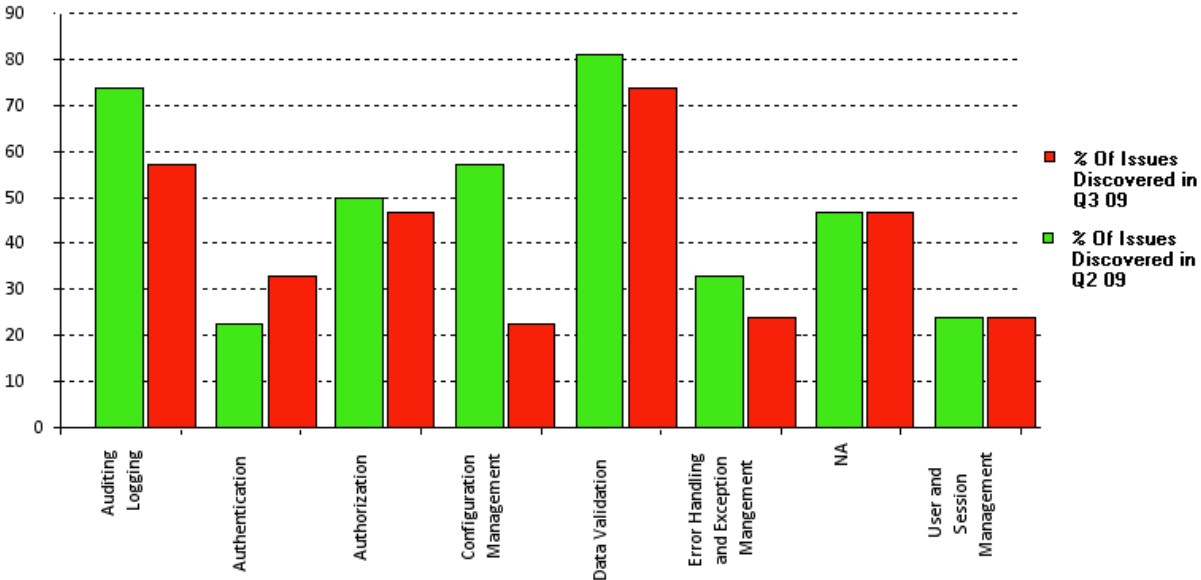
- Testing Principles
- Testing Process
- Custom Web Applications
 - Black Box Testing
 - Grey Box Testing
- Risk and Reporting
- Appendix: Testing Tools
- Appendix: Fuzz Vectors



- Information Gathering
- Business Logic Testing
- Authentication Testing
- Session Management Testing
- Data Validation Testing
- Denial of Service Testing
- Web Services Testing
- Ajax Testing

Metrics and Measurements

Vulnerability Management Metrics



Essential Software Security Metrics

■ Define **where**:

- ▶ Tracking security defects throughout the SDLC

■ Define **what qualitatively**:

- ▶ Root causes: requirements, design, code, application
- ▶ Type of the issues (e.g. bugs vs. flaws vs. configuration)
- ▶ Severity (Critical, High, Medium, Low)
- ▶ SDLC Lifecycle stage where most flaws originating in

■ Define **how quantitatively**:

- ▶ % of Critical, High, Medium, Lows for application
- ▶ % of vulnerabilities closed/open
- ▶ Vulnerability density (security bugs/LOC)

Defect Taxonomy in Support of Root Cause Analysis and Defect Containment Objectives

- Analysis to support focused remediation, risk prioritization and tracking:
 - ▶ **Security Design Flaws**
 - Introduced because of errors in design
 - Can be identified with threat modeling and manual code reviews
 - ▶ **Security Coding Bugs**
 - Coding errors that result in vulnerabilities
 - Can be identified with secure code reviews and/or tools
 - ▶ **Security Configuration Issues**
 - Introduced after tests because of a change in secure configuration of either the application, the server and the infrastructure components
 - Can be identified by testing the application close to production staging environment

Examples of Software Security Metrics

Process Metrics

- Evidence that security-check points are enforced
 - Secure code analysis
 - Vulnerability assessments
- Evidence that source code is validated against security standards (e.g. OWASP ASVS)?
- Evidence of security oversight by security officers, SME:
 - ▶ Security officers signing off design documents
 - ▶ SME participate to secure code review
 - ▶ Security officer complete risk assessments
- Training coverage on software security

Management Metrics

- % of security issues identified by lifecycle phase
- % of issues whose risk has been accepted vs. % of security issues being fixed
- % of issues per project over time (between quarter to quarter)
- % of type of issues per project over time
- Average time required to fix/close vulnerabilities during design, coding and testing
- Average time to fix issues by issue type
- Average time to fix issue by application size/code complexity



Security Metrics Goals The Good and The Bad

- **Good:** if goals when are "SMART" that is Specific, Measurable, Attainable, Realistic, Traceable and Appropriate
 - ▶ Example: reducing the overall number of vulnerabilities by 30% by fixing all low hanging fruits with source code analysis during construction
- **Bad:** if the goals justify the means to obtain the goals



Q & A

QUESTIONS
ANSWERS

Thanks for listening, further references

■ Gartner 2004 Press Release

- ▶ http://www.gartner.com/press_releases/asset_106327_1_1.html

■ Software Assurance Maturity Model

- ▶ <http://www.opensamm.org/>

■ The Software Security Framework (SSF)

- ▶ <http://www.bsi-mm.com/ssf/>

■ SEI Capability Maturity Model Integration CMMi

- ▶ <http://www.sei.cmu.edu/cmami/>

■ The Microsoft Security Development LifeCycle

- ▶ <http://msdn.microsoft.com/en-us/security/cc448177.aspx>

Further references con't

■ A CISO's Guide to Application Security

- ▶ http://www.nysforum.org/committees/security/051409_pdfs/A%20CISO'S%20Guide%20to%20Application%20Security.pdf

■ The Seven Touchpoints of Software Security

- ▶ <http://www.buildsecurityin.com/concepts/touchpoints/>

■ OWASP CLASP

- ▶ http://www.owasp.org/index.php/Category:OWASP_CLASP_Project

■ ITARC Software Security Assurance

- ▶ <http://iac.dtic.mil/iatac/download/security.pdf>

Further references con't

- OWASP Education Module Embed within SDLC
 - ▶ http://www.owasp.org/index.php/Education_Module_Embed_within_SDLC
- Producing Secure Software With Software Security Enhanced Processes
 - ▶ <http://www.net-security.org/dl/insecure/INSECURE-Mag-16.pdf>
- Security Flaws Identification and Technical Risk Analysis Through Threat Modeling
 - ▶ <http://www.net-security.org/dl/insecure/INSECURE-Mag-17.pdf>