

# Why Code Reviews And Pen-Tests Are Not Enough

FEBRUARY 2015 OWASP BELGIUM CHAPTER MEETING

JIM DELGROSSO

@JIMDELGROSSO  
DELNET2013(AT)CIGITAL.COM

# Introduction

## Jim DelGrosso

- Spend a great deal of time working with companies to find security design flaws
- Run Cigital's Architecture Analysis practice
- 20+ years in software development in many different domains
- ~15 years focusing on software security
- Executive Director of IEEE CS CSD initiative



@jimdelgrosso



# Digital Touchpoints



# Bugs And Flaws

# The Defect Universe – Bugs And Flaws



Cross Site Scripting  
Buffer Overflow



Weak/Missing/Wrong  
Security Control

(Implementation) BUGS

(Design) FLAWS

Code Review

Penetration Testing

Architecture Analysis

# Bugs And Flaws Comparison

# Authentication Defects

Description	Bug	Flaw
LDAP Injection		
Two-step authentication process with hidden user account, performed on client side		

# Logging Defects

Description	Bug	Flaw
Allow logs to be altered without detection		
Writing sensitive data to 'normal' application logs		
Log Injection		
Not tokenizing data for easy log aggregation		

# Cryptography Defects

Description	Bug	Flaw
Use a weak IV or key with a crypto primitive		
Use a confidentiality control where an integrity control is necessary		
Hardcoded key in source code		

# Examples Of Bugs And Flaws

## Implementation **BUGS**

- SQL Injection
- XML/XPath/\* Injection
- Cross-Site Scripting
- Buffer Overflow
- Unsafe system calls
- Predictable Identifiers
- Hardcoding secrets in code

## Design **FLAWS**

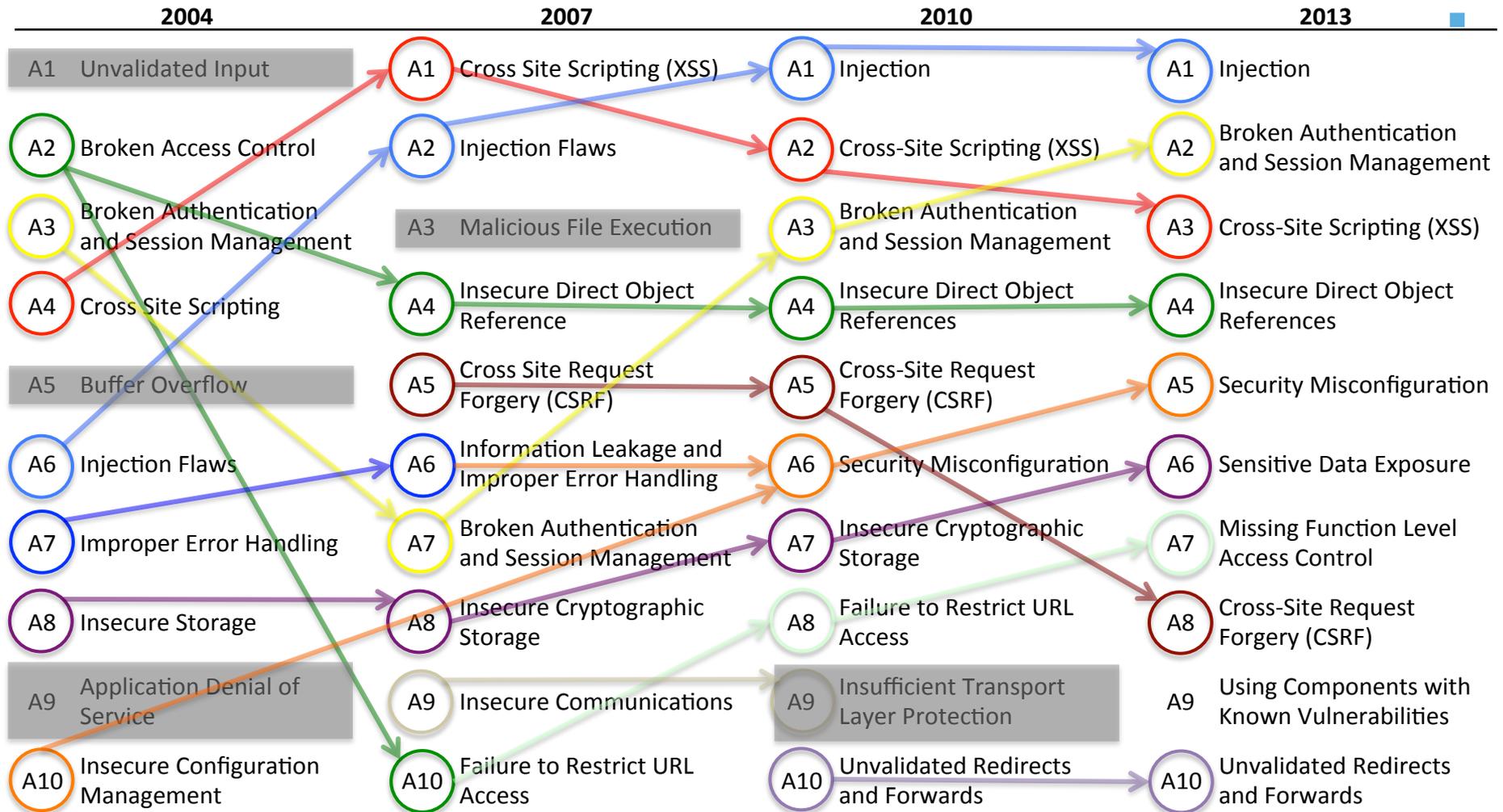
- Misuse of cryptography
- Broad trust between components
- Client-side trust
- Broken or illogical access control (RBAC over tiers)
- Missing defense for replay attacks
- Insecure auditing

# So How Are We Doing? (regarding software security)

# OWASP Top Ten

2004		2007		2010		2013	
A1	Unvalidated Input	A1	Cross Site Scripting (XSS)	A1	Injection	A1	Injection
A2	Broken Access Control	A2	Injection Flaws	A2	Cross-Site Scripting (XSS)	A2	Broken Authentication and Session Management
A3	Broken Authentication and Session Management	A3	Malicious File Execution	A3	Broken Authentication and Session Management	A3	Cross-Site Scripting (XSS)
A4	Cross Site Scripting	A4	Insecure Direct Object Reference	A4	Insecure Direct Object References	A4	Insecure Direct Object References
A5	Buffer Overflow	A5	Cross Site Request Forgery (CSRF)	A5	Cross-Site Request Forgery (CSRF)	A5	Security Misconfiguration
A6	Injection Flaws	A6	Information Leakage and Improper Error Handling	A6	Security Misconfiguration	A6	Sensitive Data Exposure
A7	Improper Error Handling	A7	Broken Authentication and Session Management	A7	Insecure Cryptographic Storage	A7	Missing Function Level Access Control
A8	Insecure Storage	A8	Insecure Cryptographic Storage	A8	Failure to Restrict URL Access	A8	Cross-Site Request Forgery (CSRF)
A9	Application Denial of Service	A9	Insecure Communications	A9	Insufficient Transport Layer Protection	A9	Using Components with Known Vulnerabilities
A10	Insecure Configuration Management	A10	Failure to Restrict URL Access	A10	Unvalidated Redirects and Forwards	A10	Unvalidated Redirects and Forwards

# Once Again – With Some Color



# Finding Flaws

# How To Find Flaws?

- Code review unlikely to find much
- Pen-testing unlikely to find much without deep system knowledge and a lot of time
- Need something else...
  - A type of analysis focusing on how we design a system
  - A different set of checklists

**\*\*\* Not replacing PT or SCR \*\*\***

# Using Architecture Analysis To Find Flaws

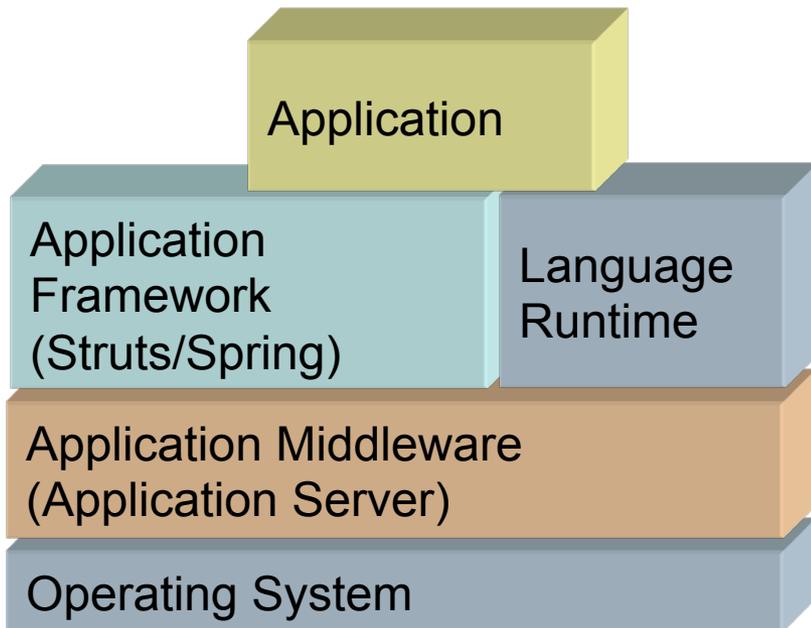
- **Dependency Analysis**
- **Known Attack Analysis**
- **System Specific Analysis**

# Finding Flaws

## DEPENDENCY ANALYSIS

# Dependency Analysis

Software is built upon layers of other software



What kind of flaws are found?

- Known vulnerabilities in open-source or product versions
- Weak security controls provided with the framework
- Framework features that must be disabled or configured to their secure form

# Dependency Analysis



## National Vulnerability Database

automating vulnerability management, security measurement, and compliance checking

Vulnerabilities	Checklists	Products/800-53A	Product Dictionary	Impact Metrics	Data Feeds	Statistics	FAQs
Home	SCAP	Related Tools	SCAP Events	About	Contact	Vendor Comments	

**Mission and Overview**  
NVD is the U.S. government repository of standards based vulnerability management data. This data enables automation of vulnerability management, security measurement, and compliance (e.g. FISMA).

### Search Results (Refine Search)

There are **42** matching records.  
Displaying matches **1** through **20**.

#### Search Parameters:

- **Keyword (text search):** ruby rails
- **Search Type:** Search Last 3 Years
- **Contains Software Flaws (CVE)**

1 2 3 > >>

**Resource Status**  
**NVD contains:**  
68647 [CVE Vulnerabilities](#)  
278 [Checklists](#)  
248 [US-CERT Alerts](#)  
4326 [US-CERT Vuln Notes](#)  
10286 [OVAL Queries](#)  
100871 [CPE Names](#)  
**Last updated:** 2/9/2015 12:03:15 PM  
**CVE Publication rate:** 24.13

#### CVE-2014-7829

**Summary:** Directory traversal vulnerability in actionpack/lib/action\_dispatch/middleware/static.rb in Action Pack in Ruby on Rails 3.x before 3.2.21, 4.0.x before 4.0.12, 4.1.x before 4.1.8, and 4.2.x before 4.2.0.beta4, when serve\_static\_assets is enabled, allows remote attackers to determine the existence of files outside the application root via vectors involving a \ (backslash) character, a similar issue to CVE-2014-7818.

**Published:** 11/18/2014 6:59:03 PM

**CVSS Severity:** 5.0 MEDIUM

#### CVE-2014-7819

**Summary:** Multiple directory traversal vulnerabilities in server.rb in Sprockets before 2.0.5, 2.1.x before 2.1.4, 2.2.x before 2.2.3, 2.3.x before 2.3.3, 2.4.x before 2.4.0, 2.5.x before 2.5.1, 2.6.x and 2.7.x before 2.7.1, 2.8.x before 2.8.3, 2.9.x before 2.9.4, 2.10.x before 2.10.2, 2.11.x before 2.11.3, 2.12.x before 2.12.3, and 3.x before 3.0.0.beta.3, as distributed with Ruby on Rails 3.x and 4.x, allow remote attackers to determine the existence of files outside the application root via a ../ (dot dot slash) sequence with (1) double slashes or (2) URL encoding.

**Published:** 11/8/2014 6:55:03 AM

**CVSS Severity:** 5.0 MEDIUM

#### CVE-2014-7818

**Summary:** Directory traversal vulnerability in actionpack/lib/action\_dispatch/middleware/static.rb in Action Pack in Ruby on Rails 3.x before 3.2.20, 4.0.x before 4.0.11, 4.1.x before 4.1.7, and 4.2.x before 4.2.0.beta3, when serve\_static\_assets is enabled, allows remote attackers to determine the existence of files outside the application root via a /..%2F sequence.

**Published:** 11/8/2014 6:55:02 AM

**Email List**  
NVD provides four mailing lists to the public. For information and subscription instructions please visit [NVD Mailing Lists](#)



# Finding Flaws

KNOWN ATTACK ANALYSIS

# Known Attack Analysis

Understanding known attacks provide insight

- Designers – what controls are needed to prevent them
- Attackers – what to try again



# Known Attack Analysis

## What defects show up “often”?

- Client-side trust
- Missing or weak control
  - XSS
  - CSRF
  - Logging and auditing
  - Click-jacking
- Session management

# Known Attack Analysis

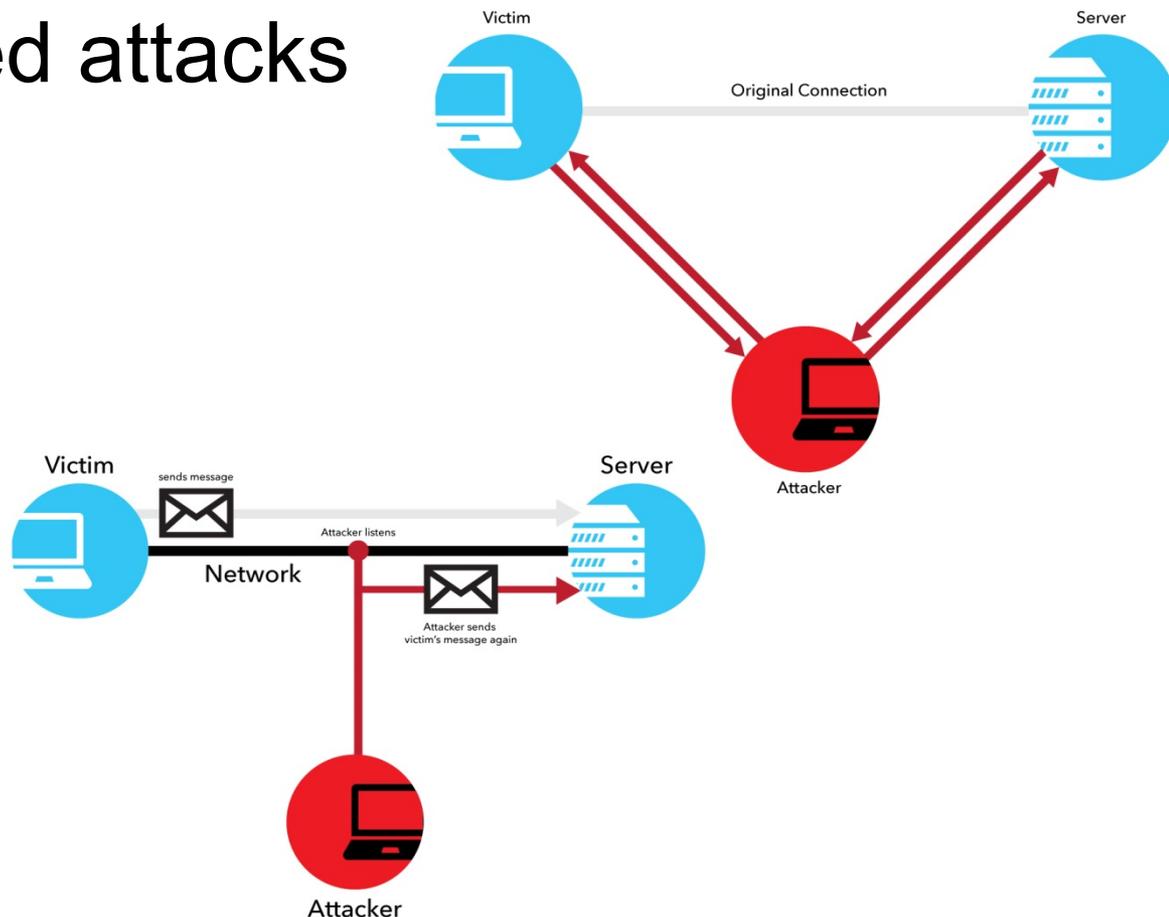
Identify design elements historically vulnerable to attack

- Distributed architecture
- Dynamic code generation and interpretation
- APIs across stateless protocols
- Client code – RIA, Mobile, ...
- Service-Oriented Architecture

# Distributed Architecture

- Distributed systems are susceptible to network-based attacks

- Eavesdrop
- Tamper
- Spoof
- Hijack
- Observe
- Replay



# Dynamic Code Generation and Interpretation

- Languages and programming environments are moving more decisions from design-time to run-time
- Many attacks involve misinterpretation of data as code in these environments
- When and how will user input be used by runtime language interpreters?

# APIs Across Stateless Protocols

- Identifiers representing state can be abused
  - Prediction
  - Capture
  - Fixation
- State sent to the client between requests is altered or replayed

# Client Code – RIA, Mobile, ...

- Processing moved to the client
  - RIA
  - Mobile
  - HTML5
- It is still a client
- It is still an untrusted platform
- An exposed server endpoint is exposed to everyone – not just for your purposes

# Service-Oriented Architecture (SOA)

- Security needed for SOA components
  - Web-services: SOAP/WSDL/UDDI
  - Message-oriented middleware
  - Enterprise Service Bus
- Common Problems
  - Exposing backend code to dynamic attacks
  - Channel versus message security

# Finding Flaws

SYSTEM SPECIFIC ANALYSIS

# System Specific Analysis Flaws

- Weakness in a custom protocol
- Reusing authentication credentials
- Not following good software security design principles

## Model the software by understanding

- Threat agent
- Asset
- Attack
- Attack surface
- Attack goal
- Security control

# Some Work Being Done By IEEE

# Why Does The IEEE CS CSD Exist?

- IEEE Computer Society wanted to expand their presence in security
  - Kathy Clark-Fisher is the program director of the Center for Secure Design initiative
- What problem is nobody solving?
  - The stuff that keeps happening  
... over and over again ...
- Focus on weak design

# Initial Workshop Attendees

Organization	Individual
Athens University of Economics and Business	Diomidis Spinellis
Cigital	Jim DelGrosso
Cigital	Gary McGraw
EMC	Izar Tarandach
George Washington University	Carl Landwehr
Google	Christoph Kern
Harvard University	Margo Seltzer
HP	Jacob West
McAfee, Part of Intel Security Group	Brook Schoenfield
RSA	Danny Dhillon
Sadosky Foundation	Iván Arc
Twitter	Neil Daswani
University of Washington	Tadayoshi Kohno

# Avoiding The Top Ten Security Flaws

- Earn or give, but never assume, trust
- Use an authentication mechanism that cannot be bypassed or tampered with
- Authorize after you authenticate
- Strictly separate data and control instructions, and never process control instructions received from untrusted sources
- Define an approach that ensures all data are explicitly validated
- Use cryptography correctly
- Identify sensitive data and how they should be handled
- Always consider the users
- Understand how integrating external components changes your attack surface
- Be flexible when considering future changes to objects and actors

**<http://cybersecurity.ieee.org/center-for-secure-design/>**

# Example 1: Avoiding Top Ten Security Flaws

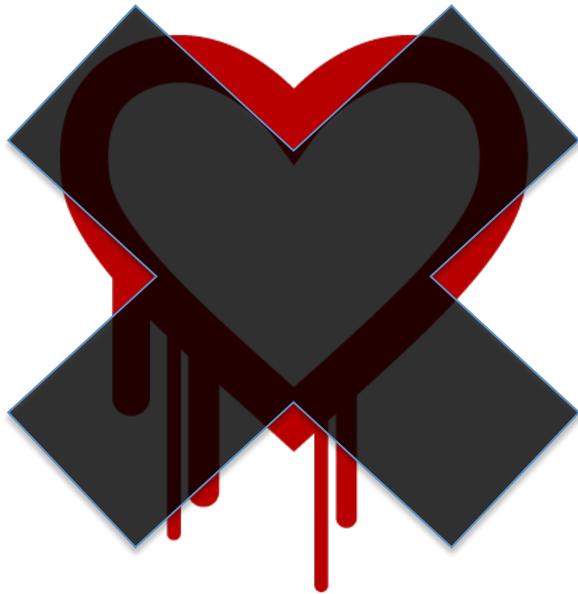
- Strictly separate data and control instructions, and never process control instructions received from untrusted sources

*[http://cacm.acm.org/magazines/  
2014/9/177924-securing-the-tangled-web/  
fulltext](http://cacm.acm.org/magazines/2014/9/177924-securing-the-tangled-web/fulltext)*

*by Christoph Kern (Google)*

# Example 2: Avoiding Top Ten Security Flaws

- Use cryptography correctly



BUG



FLAW

## Example 3: Avoiding Top Ten Security Flaws

- Understand how integrating external components changes your attack surface





# Thank You