Why Code Reviews And Pen-Tests Are Not Enough

FEBRUARY 2015 OWASP BELGIUM CHAPTER MEETING
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
Cigital 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

<table>
<thead>
<tr>
<th>Description</th>
<th>Bug</th>
<th>Flaw</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAP Injection</td>
<td>✔</td>
<td></td>
</tr>
<tr>
<td>Two-step authentication process with hidden user account, performed on client side</td>
<td>✔</td>
<td></td>
</tr>
</tbody>
</table>
# Logging Defects

<table>
<thead>
<tr>
<th>Description</th>
<th>Bug</th>
<th>Flaw</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow logs to be altered without detection</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Writing sensitive data to 'normal' application logs</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Log Injection</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Not tokenizing data for easy log aggregation</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
# Cryptography Defects

<table>
<thead>
<tr>
<th>Description</th>
<th>Bug</th>
<th>Flaw</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use a weak IV or key with a crypto primitive</td>
<td>✔️</td>
<td></td>
</tr>
<tr>
<td>Use a confidentiality control where an integrity control is necessary</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Hardcoded key in source code</td>
<td>✔️</td>
<td>✔️</td>
</tr>
</tbody>
</table>
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

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

### 2007
- A1: Cross Site Scripting (XSS)
- A2: Injection Flaws
- A3: Malicious File Execution
- A4: Insecure Direct Object Reference
- A5: Cross Site Request Forgery (CSRF)
- A6: Information Leakage and Improper Error Handling
- A7: Broken Authentication and Session Management
- A8: Insecure Cryptographic Storage
- A9: Insecure Communications
- A10: Failure to Restrict URL Access

### 2010
- A1: Injection
- A2: Cross-Site Scripting (XSS)
- A3: Broken Authentication and Session Management
- A4: Insecure Direct Object References
- A5: Cross-Site Request Forgery (CSRF)
- A6: Security Misconfiguration
- A7: Insecure Cryptographic Storage
- A8: Failure to Restrict URL Access
- A9: Insufficient Transport Layer Protection
- A10: Unvalidated Redirects and Forwards

### 2013
- A1: Injection
- A2: Broken Authentication and Session Management
- A3: Cross-Site Scripting (XSS)
- A4: Insecure Direct Object References
- A5: Security Misconfiguration
- A6: Sensitive Data Exposure
- A7: Missing Function Level Access Control
- A8: Cross-Site Request Forgery (CSRF)
- A9: Using Components with Known Vulnerabilities
- 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…
  o A type of analysis focusing on how we design a system
  o 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
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

## NVD Database

### 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)**

### 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 2.4.5, 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, 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
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
  o XSS
  o CSRF
  o Logging and auditing
  o 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
  o Prediction
  o Capture
  o Fixation

• State sent to the client between requests is altered or replayed
Client Code – RIA, Mobile, …

• Processing moved to the client
  o RIA
  o Mobile
  o 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

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

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