Building Secure Applications

Thomas Stiehm, CTO
tom.stiehm@coveros.com

© Copyright 2009 Coveros, Inc.. All rights reserved.

About Coveros

- Coveros helps organizations accelerate the delivery of secure, reliable software

© Copyright 2009 Coveros, Inc.. All rights reserved.

Why Application Security is Important

© Copyright 2009 Coveros, Inc.. All rights reserved.

The State of Application Security

coveros

- We are really good at finding bugs
- We are not so great at finding flaws
- We sometimes fix the problems we find
- We mostly never build security in

© Copyright 2009 Coveros, Inc.. All rights reserved.

Finding Bugs

- Common Implementation Errors
  - Buffer overflow
  - Race conditions
  - TOCTOU (time of check to time of use)
  - Unsafe environment variables
  - Unsafe system calls
  - Untrusted input problems

CodeSecure™

© Copyright 2009 Coveros, Inc.. All rights reserved.

Finding Flaws

- Flaws (Design Defects)
  - Misuse of cryptography
  - Compartmentalization problems in design
  - Privileged block protection failure
  - Type safety confusion error
  - Insecure auditing
  - Broken or illogical access control
  - Method over-riding problems

© Copyright 2009 Coveros, Inc.. All rights reserved.

Why you can't just test security into your apps

coveros

- Most Quality Assurance (QA) professionals are not application security specialist – most have no security testing training or experience
- Those organizations that do security testing tend to have an overreliance on automated security testing software
- Automated tools are not good at finding flaws
- New security defects are discovered every day and it takes time to get those programmed into tools
- Reliance on automated security testing tools without a grounding in security testing can lead to a false sense of security

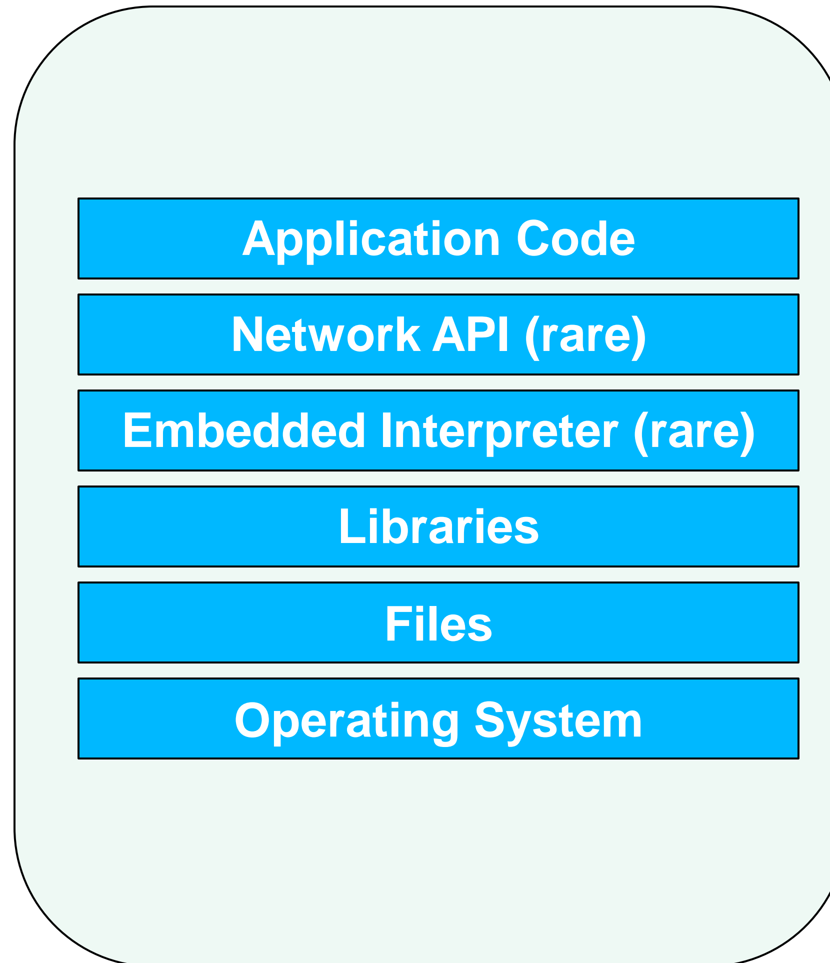
© Copyright 2009 Coveros, Inc.. All rights reserved.

Why building security into an application is hard

- Applications are complex and suites of applications that support a business are extremely complex
- The technology stacks used to implement modern application are enormous, deep and wide, working across several tiers including the client, middleware and data tiers
- Wide use of multiple application development frameworks, libraries and components
- Application security defects can be in any tier or spread across multiple tiers based on sophisticated interacts between the tiers
- Most application developers are not and will never be security aware much less security experts

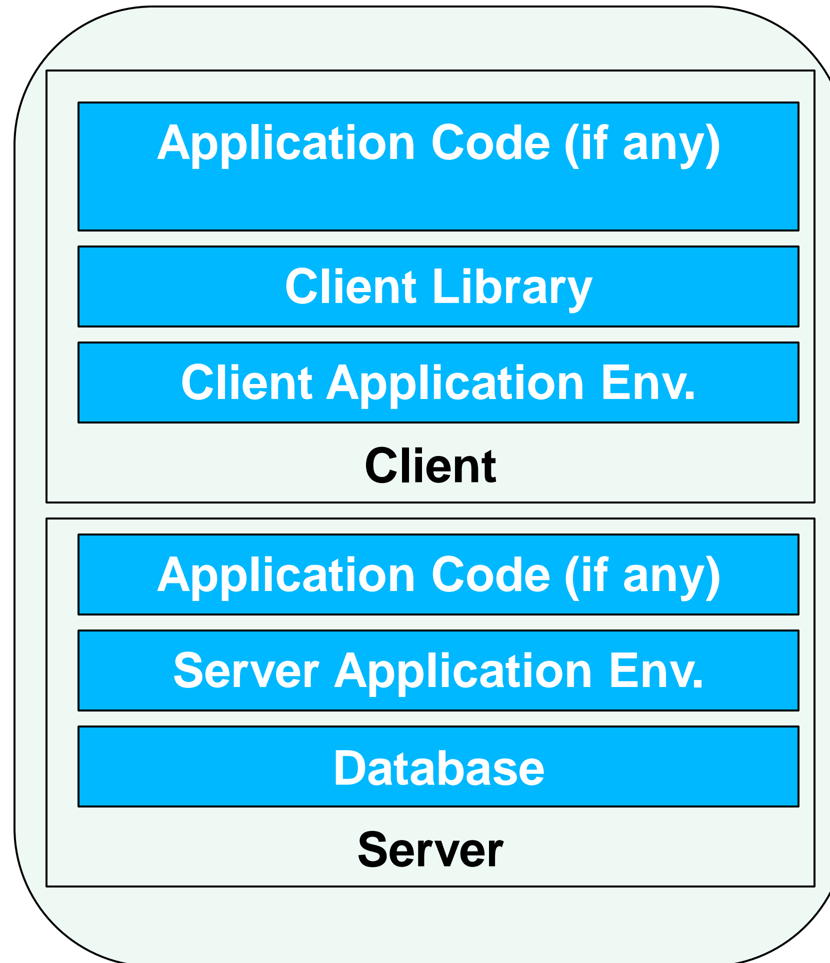
© Copyright 2009 Coveros, Inc.. All rights reserved.

Desktop Application Complexity - 80s Apps



© Copyright 2009 Coveros, Inc.. All rights reserved.

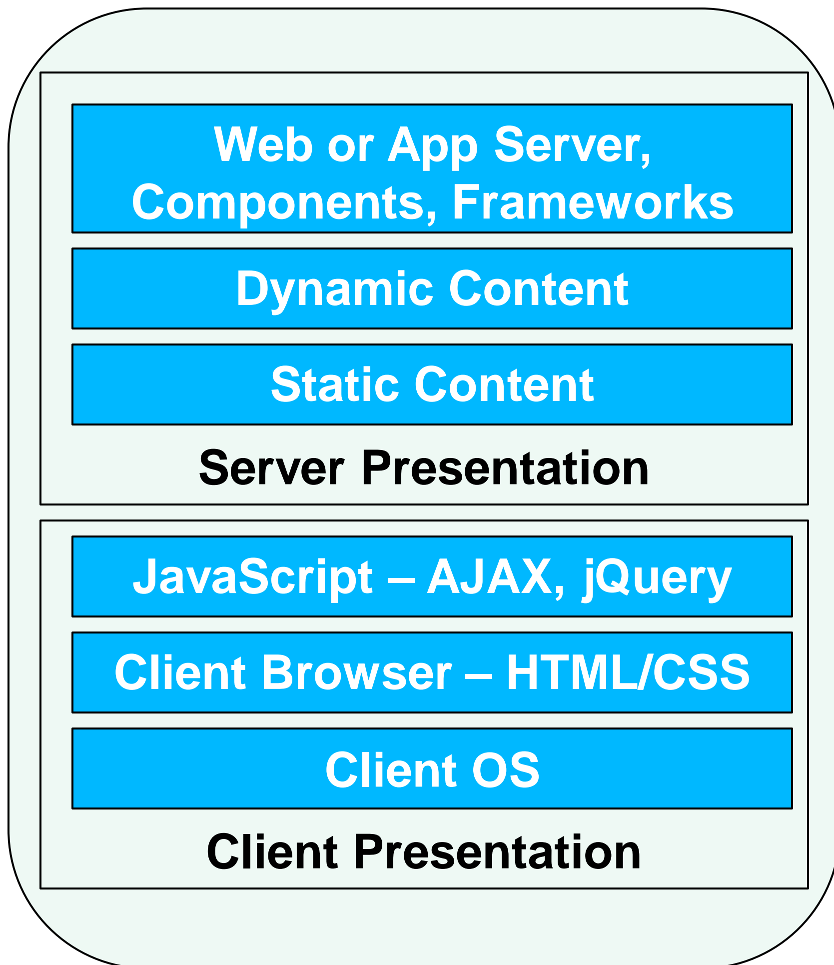
Client/Server Complexity - 90s Apps

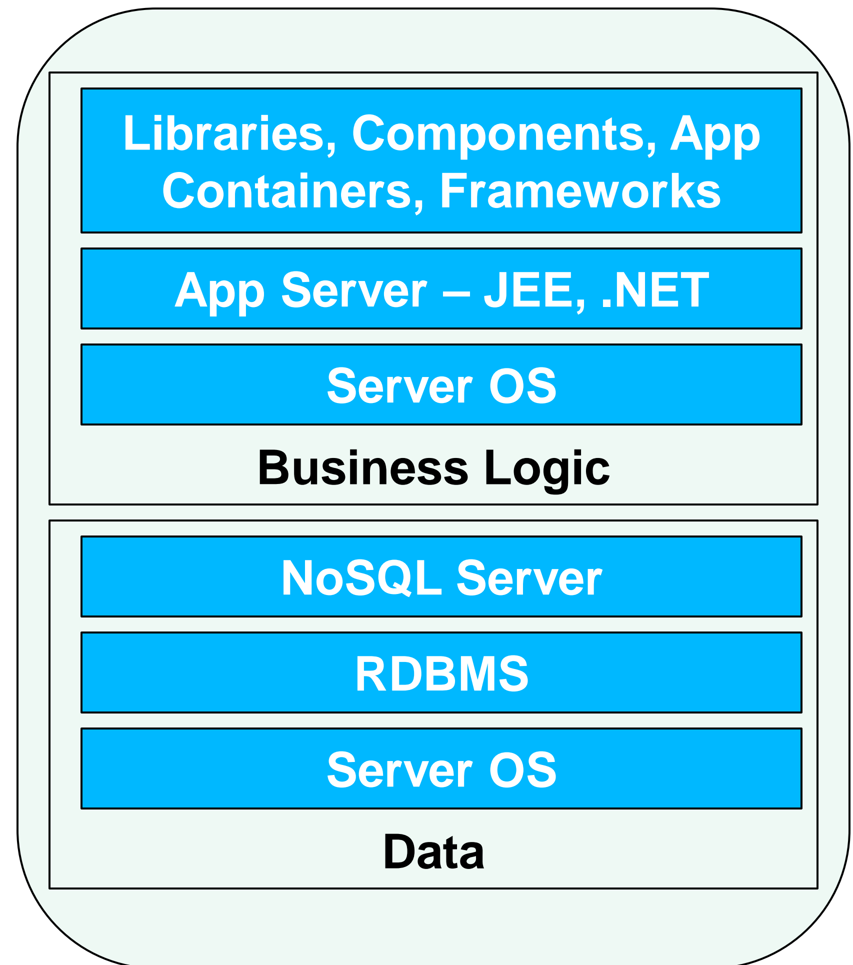


© Copyright 2009 Coveros, Inc.. All rights reserved.

Layers of Complexity – 2000s Web Apps

coveros





© Copyright 2009 Coveros, Inc.. All rights reserved.

Why is it hard to create Application Security Requirements?

- Most business users aren't security experts and will not actively consider security features on their own
- Security is often implicitly "assumed" into the requirements
- Even if a business user asks about application security they lack the expertise to specify what their needs are
- Application security is never a problem, until it is a problem and then it is an instant crisis
- Most software developers lack the ability to translate security vulnerabilities into business risk making prioritizing application security hard to do as compared to functional requirements
- Industry standards can help make the case for application security (think PCI, HIPAA and Basel II)

© Copyright 2009 Coveros, Inc.. All rights reserved.

How do you build security into Applications?

coveros

- Adopt and use an application security process from the beginning of the project
- Application Security requirements should be created along with the functional application requirements and implemented with the code features of the software
- Lead the security requirements process, sell the value of good security practices to the business
- Create security standards and practices and put security controls into your base software architecture
- Monitor compliance with your security standards
- Adopt the use of security tools such as static code analysis and web scanners
- Get in the practice of conducting manual security verification like code reviews and penetration testing

© Copyright 2009 Coveros, Inc.. All rights reserved.

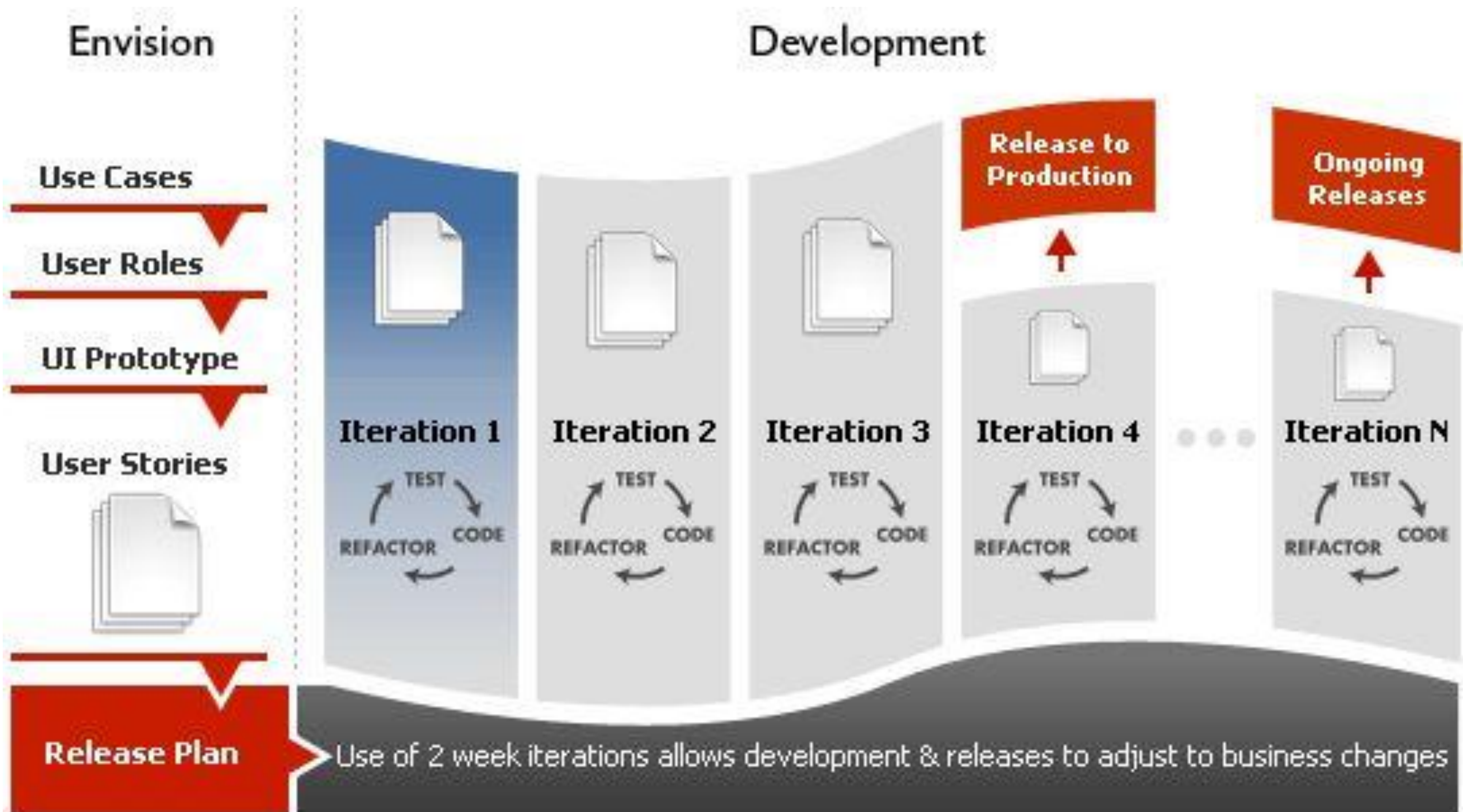
Agile and Security

coveros

- There is a widely held belief that secure applications cannot be developed using agile practices.
- This belief comes from:
  - Misunderstanding agile (aka myths about agile)
  - Misunderstanding application security
  - An excuse not to change
- We have successfully built many, many security-critical applications using agile

© Copyright 2009 Coveros, Inc.. All rights reserved.

SecureAgile^{TM} Development Process



Assures time-to-market while achieving security objectives

© Copyright 2009 Coveros, Inc.. All rights reserved.

SecureAgile^{TM} Security Practices

- Misuse/Abuse Cases
- Security Stories
- Defensive Design and Programming
- Continuous Software Security Assurance
  - Architectural risk analysis
  - Code analysis
  - Security and penetration testing

© Copyright 2009 Coveros, Inc.. All rights reserved.

Misuse / Abuse Case Development

- Purpose: Define the possible mechanisms an adversary might exploit to compromise your system
- Approach:
  - Misuse cases are extensions to use cases that highlight ways in which the system might be misused accidentally
  - Abuse cases are extensions to use cases that highlight ways in which the system might be abused on purpose
- Results:
  - Insight into potential abuses that can be avoided and also tested for later

© Copyright 2009 Coveros, Inc.. All rights reserved.

Security Stories

- Purpose: Document the non-functional security requirements associated with the system
- Approach
  - "User shall not …" nomenclature
- Purpose
  - Assure all explicit security requirements are documented to aid secure development and testing activities

© Copyright 2009 Coveros, Inc.. All rights reserved.

Defensive design and coding

coveros

- Incorporation of security controls into software design and code
  - Security frameworks like OWASP ESAPI
- Use of vetted components
  - Libraries of secure components
- Build security controls into your application architecture
- Examination of design / code looking for realization of architectural risks and misuses / abuses

© Copyright 2009 Coveros, Inc.. All rights reserved.

Software Assurance

coveros

- Architectural risk analysis
  - Assess architecture against threat model, attack patterns, known weaknesses
- Secure code review
  - Both automated and manual
- Security testing
  - Risk-based testing
  - Testing of security functionality
- Penetration testing
  - Performed during the release process

© Copyright 2009 Coveros, Inc.. All rights reserved.

Continuous Integration

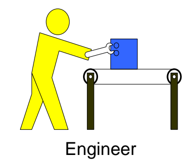
coveros

- Automation of build, test, deploy process
  - Check-in builds / tests
  - Nightly code integrations and regression tests
  - Automated promotion between test stages
  - Automated notification of build failures

- A critical capability to have when building software using agile … and supports security analysis

- Integration of code analysis and automated security testing can result in identification of security issues early in the process

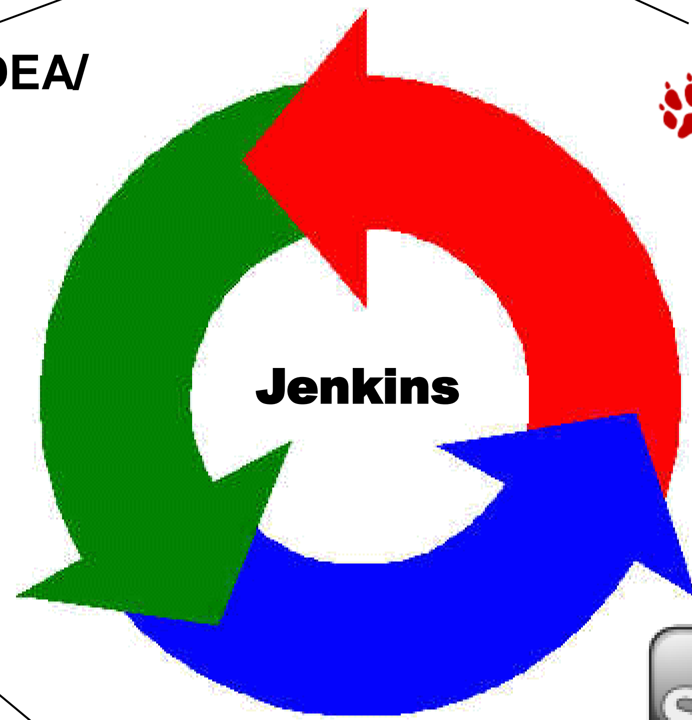
© Copyright 2009 Coveros, Inc.. All rights reserved.



Track progress

Create code

IntelliJ IDEA/ Eclipse



Test security

Version code

Build application

Test application

© Copyright 2009 Coveros, Inc.. All rights reserved.

Real World Example

coveros

- Division of a Fortune 500 Bank
- Creating a suite of applications to be used for a new service offering within the bank
- All applications will use the same core architecture and infrastructure providing a large number of components for each individual application to use
- The applications will be accessible over the Internet to select clients and bank staff

© Copyright 2009 Coveros, Inc.. All rights reserved.

How we got involved

coveros

- During the development of the suite of applications the bank commissioned an architectural risk analysis
- The architectural risk analysis focused on the design and architecture of the core components
- During the architectural risk analysis many deficiencies where discovered in the design of the application
- As a result of the findings a team was created to help the architecture team build security controls into the core components of the system
- The bank had already purchased a static analysis tools to find core level defects. The team decided to extend the use of the tool to make sure that development used the security controls

© Copyright 2009 Coveros, Inc.. All rights reserved.

Technology Stack

coveros

- Java, Java Enterprise Edition (JEE)
- Google Web Toolkit (GWT)
- Spring Framework
- Red Hat Linux
- Sybase
- Oracle
- GigaSpaces

© Copyright 2009 Coveros, Inc.. All rights reserved.

Development Environment

coveros

- Intellij IDEA
- Windows XP
- Maven 2.0
- Hudson CI server
- GWT Ext

© Copyright 2009 Coveros, Inc.. All rights reserved.

Security Components

- OWASP ESAPI
- Hibernate Validator – JSR 303 Reference Implementation
- Spring Security Framework
- Custom Built Security Components

© Copyright 2009 Coveros, Inc.. All rights reserved.

Defects Discovered

coveros

- Lack of input validation
- No output encoding
- Weak authentication
- Exposed services, no URL or request level access checking
- Lack of logging for an audit trail
- No clickjacking prevention

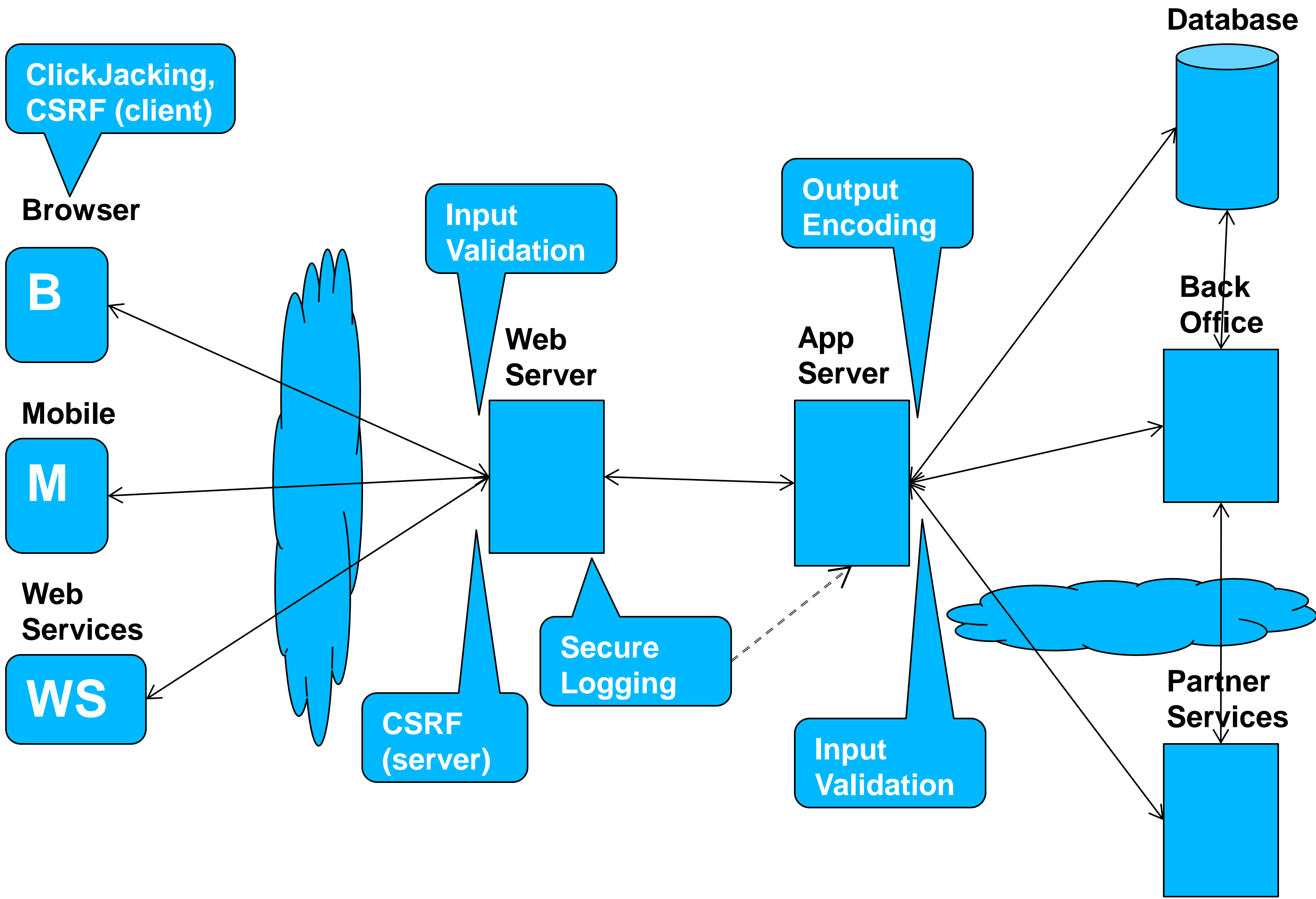
© Copyright 2009 Coveros, Inc.. All rights reserved.

Security Controls Implemented

coveros

- Clickjacking prevention
- CSRF prevention
- Input validation
- Output encoding
- Secure logging for audit and intrusion detection

© Copyright 2009 Coveros, Inc.. All rights reserved.



© Copyright 2009 Coveros, Inc.. All rights reserved.

Clickjacking prevention

- Clickjacking is where a malicious site contains a hidden frame around a trusted site
- Since users can’t see the hidden frame an attacker can make them think they are using a trusted site directly

- The design of the suite of applications allowed a clickjacking prevention control to be placed at the top level window or document
- This prevented clickjacking and didn’t allow individual development teams the ability to turn off the control

© Copyright 2009 Coveros, Inc.. All rights reserved.

CSRF prevention

coveros

- Cross Site Request Forgery is when an application accepts an unauthorized request from a user that was fooled into submitting it.

- The CSRF security control created a hard to guess token for each user session
- The CSRF security control was integrated into the request controller so that each request could be evaluated before any other processing happened
- If the request didn't have the token it was rejected

© Copyright 2009 Coveros, Inc.. All rights reserved.

Input Validation

coveros

- Input Validation is the process of checking all input into an application to make sure it is valid and usable by the application before it is used
- The Input Validation control that we created for the bank focused on user input
- It was implemented in the core architecture by sub-classing and overriding the GWT component that dispatched requests to an appropriate handler
- By using the Hibernate Validator to annotate all objects coming from the client side we could quickly determine if the input data was valid and usable by the application

© Copyright 2009 Coveros, Inc.. All rights reserved.

Output Encoding

coveros

- Output encoding is the process of formatting data coming out of an application appropriately for the context that will consume the output, ex. Escaping SQL to prevent SQL injection attacks
- The output encoding control we created for the bank focused on making sure that dynamic SQL statements used in to the application didn't contain SQL controls that could be used for a SQL injection attack
- Where ever possible and practical all dynamic SQL statements were converted to prepared statements

© Copyright 2009 Coveros, Inc.. All rights reserved.

Secure Logging for Audit and Intrusion Detection

coveros

- Secure logging is logging to a device that can't be tampered with if the server the log originates from is compromised
- The secure logging control we created for the bank was used for audit and intrusion detection
- The secure logs where written to a log server
- The Intrusion Detection System (IDS) monitored the log server looking for specific log codes that indicated anomalous traffic
- If the anomalous traffic surpassed specific thresholds an alert would be sent to the security team to investigate the issue

© Copyright 2009 Coveros, Inc.. All rights reserved.

Bottom-line

coveros

- Adopt and use an application security process
- Perform architectural risk analysis to find flaws, do it early and refresh it often (at least once per release)
- Write security requirements and test for them
- Perform threat modeling (abuse cases) and write security stories
- Use static analysis tools to check your code and site, they will point you to potential problem areas
- Use manual inspection to find defects in problem areas
- Use security testing tools, like web app scanners, to make sure you don't miss anything obvious

© Copyright 2009 Coveros, Inc.. All rights reserved.

Bottom-line

- Use Penetration testing techniques to find non-obvious and new vulnerabilities, try to break your application
- Build security controls into your architecture, make them hard to bypass and use tools to check to make sure they are being used
- Build security controls that fail safe
- Automate your static analysis and security testing tools, run them often and monitor the results
- Fix Critical and High priority issues immediately

© Copyright 2009 Coveros, Inc.. All rights reserved.

Thank You

© Copyright 2009 Coveros, Inc.. All rights reserved.

Supplemental Material

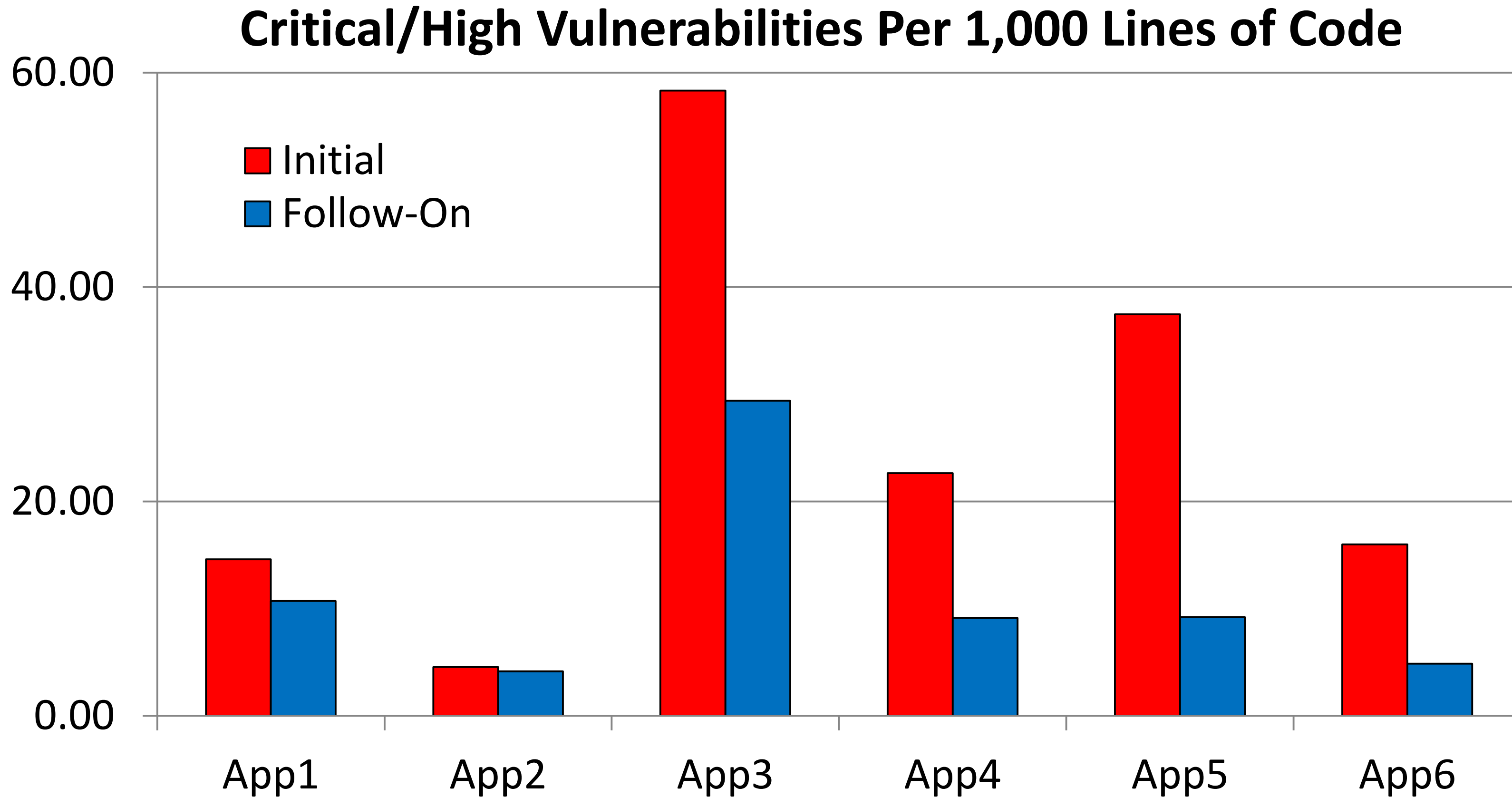
© Copyright 2009 Coveros, Inc.. All rights reserved.

Vulnerabilities to what out for

- OWASP Top Ten:
  - https://www.owasp.org/index.php/Top_10_2010
- **2011 CWE/SANS Top 25 Most Dangerous Software Errors**
  - http://cwe.mitre.org/top25/
- There is a lot of overlap as there are major categories that generate a lot of vulnerabilities
- For Example:
  - Injection Attacks and
  - Misconfigurations

© Copyright 2009 Coveros, Inc.. All rights reserved.

Fixing the Problem – One DoD Initiative



But there are 1,000's of apps … do the math

© Copyright 2009 Coveros, Inc.. All rights reserved.

GWT Input Validation Example

```
import com.google.gwt.user.server.rpc.RPCRequest;
import com.google.gwt.user.server.rpc.RemoteServiceServlet;

public class MyAppRemoteServiceServlet extends
   RemoteServiceServlet {
protected void onAfterRequestDeserialized(RPCRequest
   rpcRequest)
   {
      super.onAfterRequestDeserialized(rpcRequest);
      Object[] params = rpcRequest.getParameters();
    // validate objects in params list
   }
}
```

© Copyright 2009 Coveros, Inc.. All rights reserved.

GWT and Hibernate Validator Example

```
public class MyAppRemoteServiceServlet extends
  RemoteServiceServlet {
protected void onAfterRequestDeserialized(RPCRequest
  rpcRequest)
   {
      super.onAfterRequestDeserialized(rpcRequest);
      HashSet<ConstraintViolation<Object>> constraintViolations =
  new HashSet<ConstraintViolation<Object>>();
      Object[] params = rpcRequest.getParameters();
      for (Object param : params)
      {
            constraintViolations.addAll(validator.validate(param));
      }
      if (constraintViolations.size()>0) thrown new Exception();
   }
```

© Copyright 2009 Coveros, Inc.. All rights reserved.