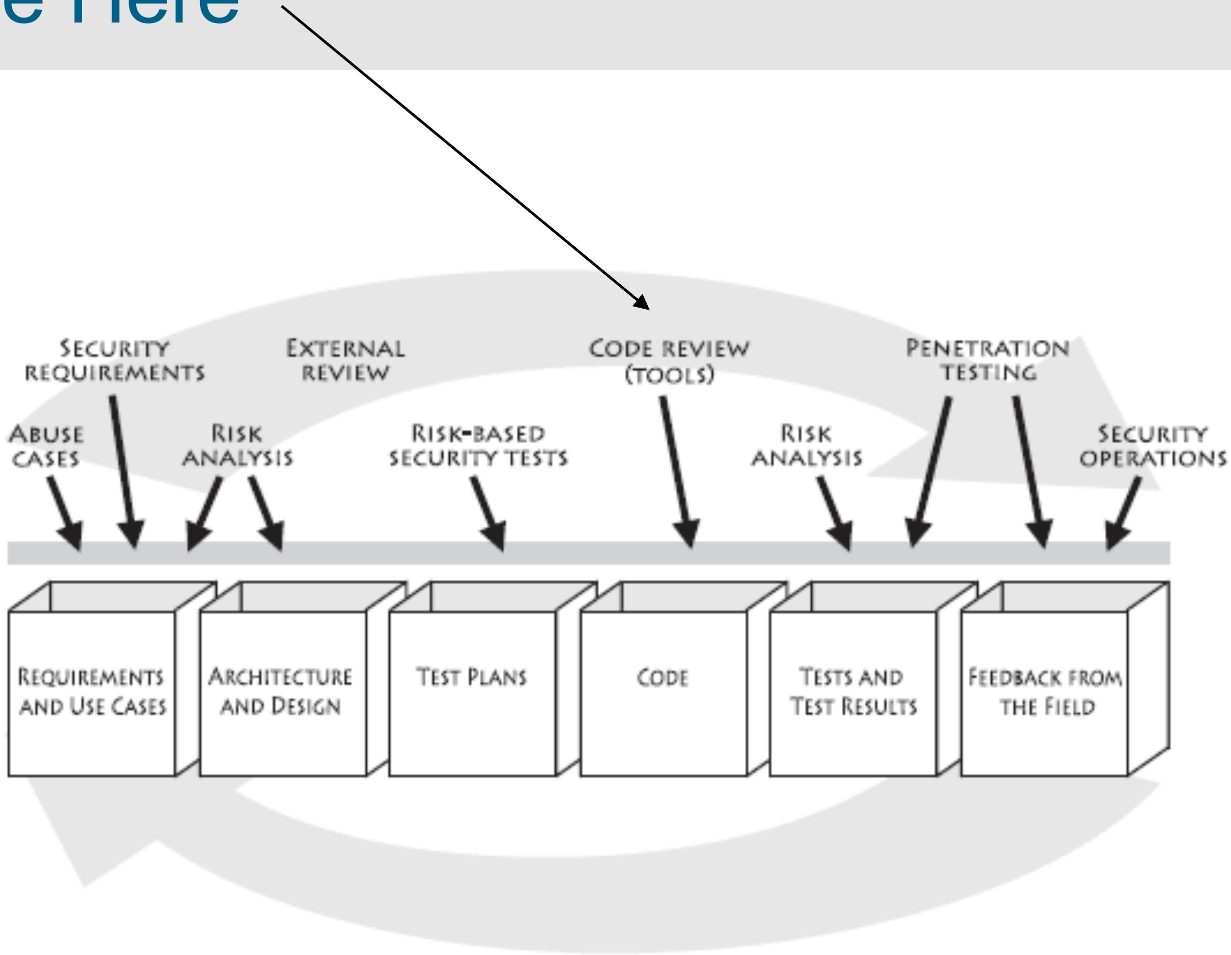
Reviewing Code for Security

John Steven
Senior Director
Advanced Technology Consulting
Cigital Inc.
jsteven@cigital.com

You Are Here



Security Requirements
External Review
Code Review (Tools)
Penetration Testing
Abuse Cases
Risk Analysis
Risk-Based Security Tests
Risk Analysis
Security Operations
Requirements and Use Cases
Architecture and Design
Test Plans
Code
Tests and Test Results
Feedback from the Field

Alternative Models / Methods

Contemporary Code Review Approaches

- Peer review
- Fagan-style code review
- Tool-based automated approach

Starting A Code Review w/ a Blank Sheet

Threat modeling MUST guide where we look…

…and for what.

cigital

Background
What is a Threat Model

What is a Threat Model

- Depiction of:
  - The system’s *attack surface*
  - *Threats* who can attack the system
  - *Assets* threats may compromise

- Some leverage risk management practices
  - Estimate *probability* of attack
  - Weight *impact* of successful attack

cigital

Threat Modeling – High-level process

1. Diagram structure
   1. Draw the *software* diagram
   2. Identify the attack surface
   3. Identify patterns' usage
   4. Identify frameworks
   5. Identify security controls
2. Show Principals, resolution
3. Show authorization required

cigital

Code Review Approaches (Highest Level)

Cigital's Three Approach

- Known Weakness Analysis
- Ambiguity Analysis
- Underlying Framework Analysis

cigital

Known Weakness Analysis: Checklist #1

Ask: is each element:

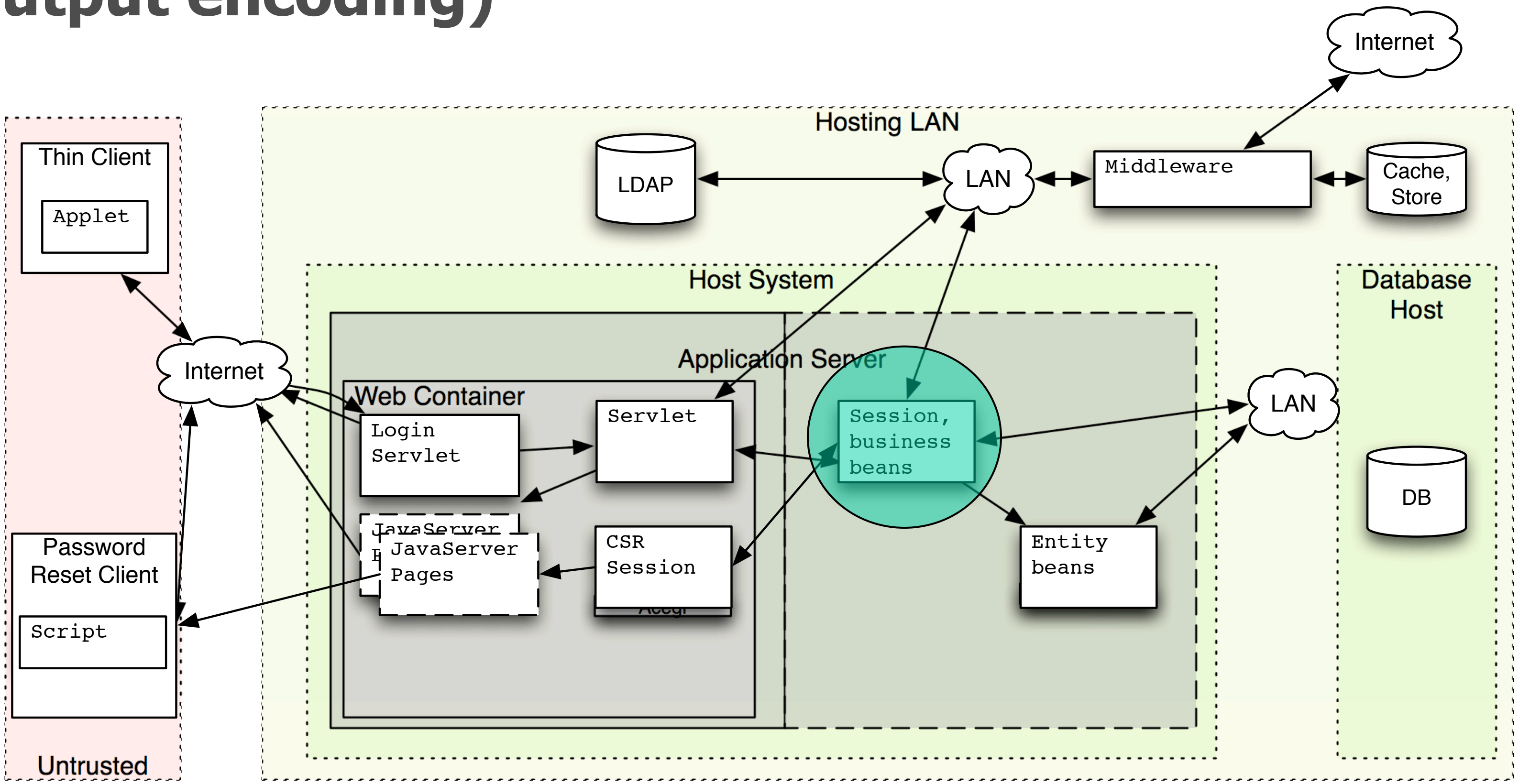
- Control absent?
- Used ineffectively
  - What’s the effect of digesting a password?
  - Does code signing prevent malicious code?
  - What does SSL (w/o) certs provide?
- Implemented correctly?
- Present, but unused

Jeff Williams has suggested this framework for security controls for some time

cigital

Key Structural Components narrow search

Component diagrams show critical choke points for security controls (input validation, authentication, output encoding)



1 - Diagram Software Structure

1.1 - Anchor in Software Architecture

Consider where attacks occur

Top-down

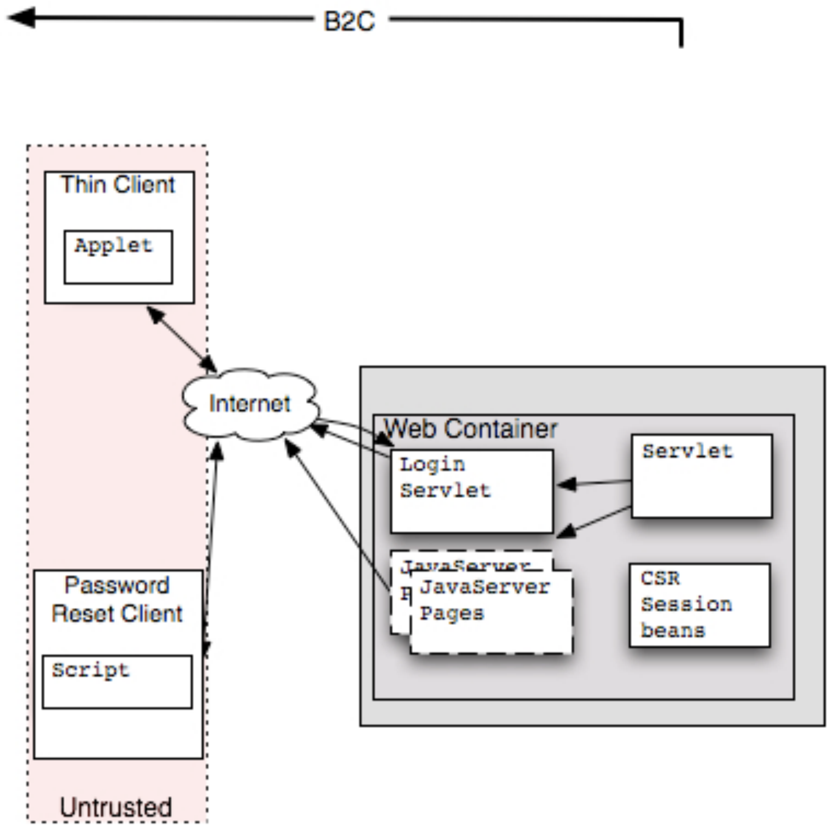
- Enumerate business objects
  - Sensitive data
  - Privileged functionality

Bottom-up

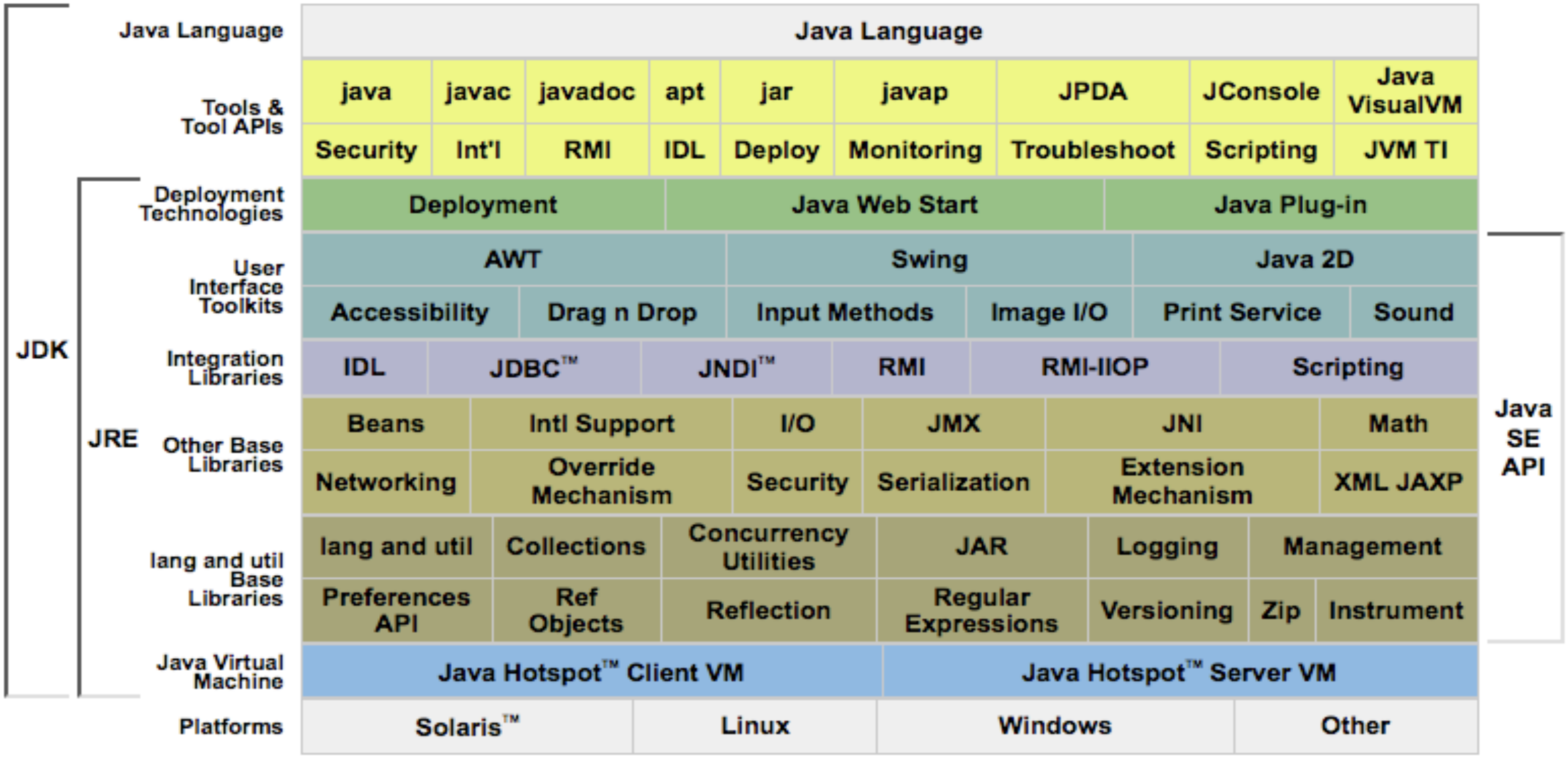
- Enumerate application entities
  - Sensitive data
  - Privileged functionality

Look for

- Middleware
- Open source
- Frameworks



Avoid ‘the stack’



JDK

JRE

Java SE API

Java Language: Java Language

Tools & Tool APIs: java | javac | javadoc | apt | jar | javap | JPDA | JConsole | Java VisualVM

Security | Int'l | RMI | IDL | Deploy | Monitoring | Troubleshoot | Scripting | JVM TI

Deployment Technologies: Deployment | Java Web Start | Java Plug-in

User Interface Toolkits: AWT | Swing | Java 2D

Accessibility | Drag n Drop | Input Methods | Image I/O | Print Service | Sound

Integration Libraries: IDL | JDBC™ | JNDI™ | RMI | RMI-IIOP | Scripting

Other Base Libraries: Beans | Intl Support | I/O | JMX | JNI | Math

Networking | Override Mechanism | Security | Serialization | Extension Mechanism | XML JAXP

lang and util Base Libraries: lang and util | Collections | Concurrency Utilities | JAR | Logging | Management

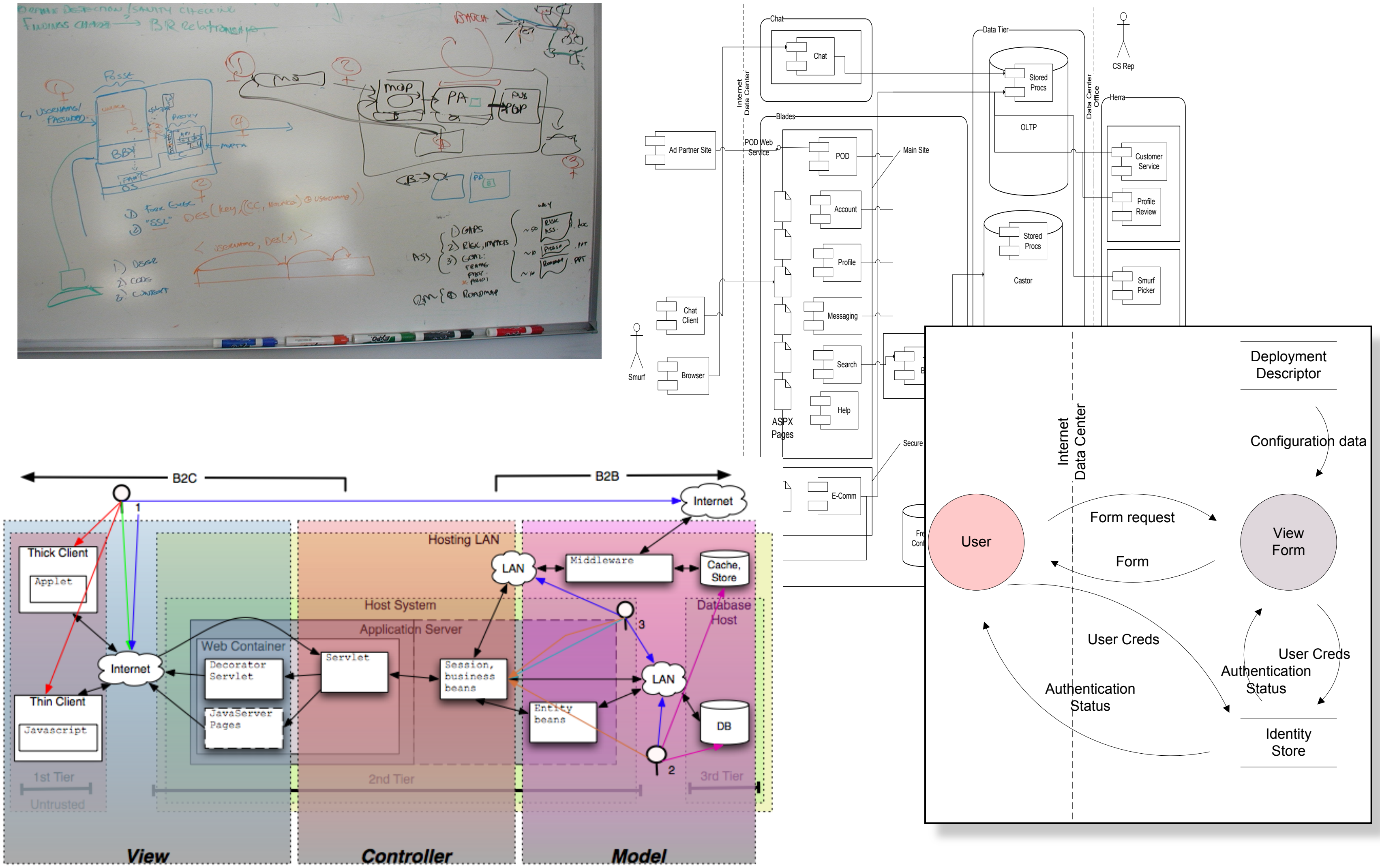
Preferences API | Ref Objects | Reflection | Regular Expressions | Versioning | Zip | Instrument

Java Virtual Machine: Java Hotspot™ Client VM | Java Hotspot™ Server VM

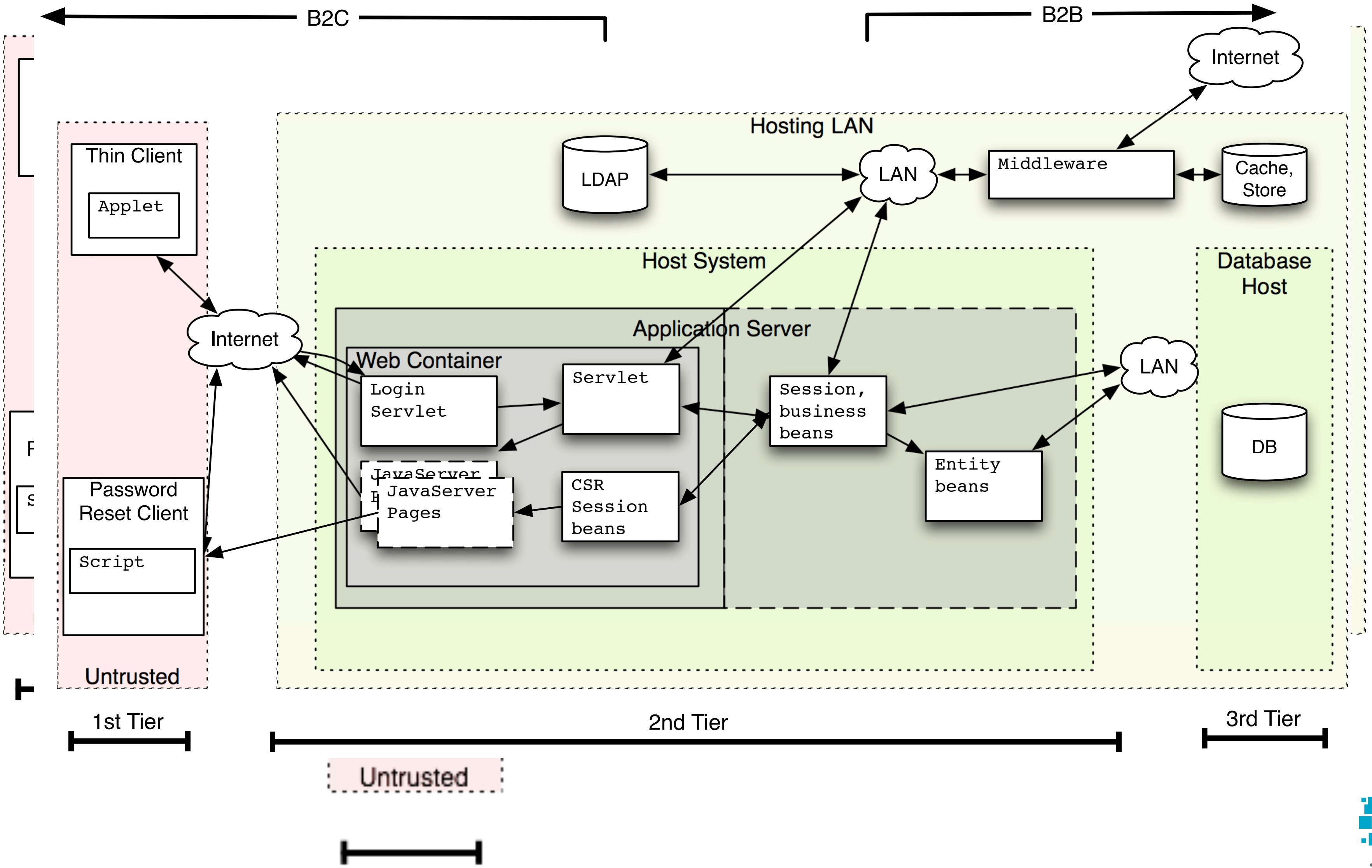
Platforms: Solaris™ | Linux | Windows | Other

What does this diagram tell you about component interaction?

Architecture Diagrams



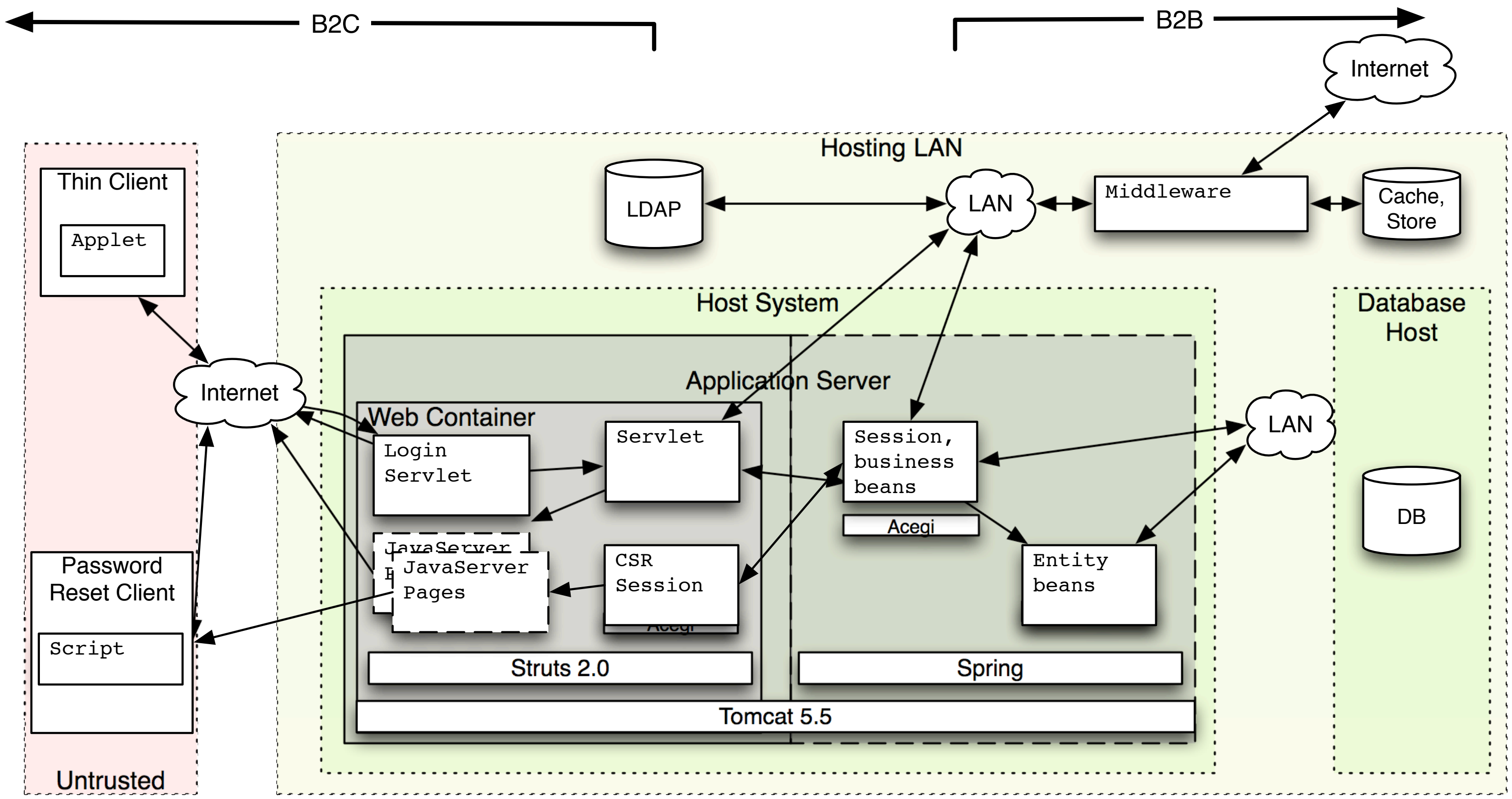
1.2 – Identify Application Attack Surface



cigital

1.5 – Identify Frameworks

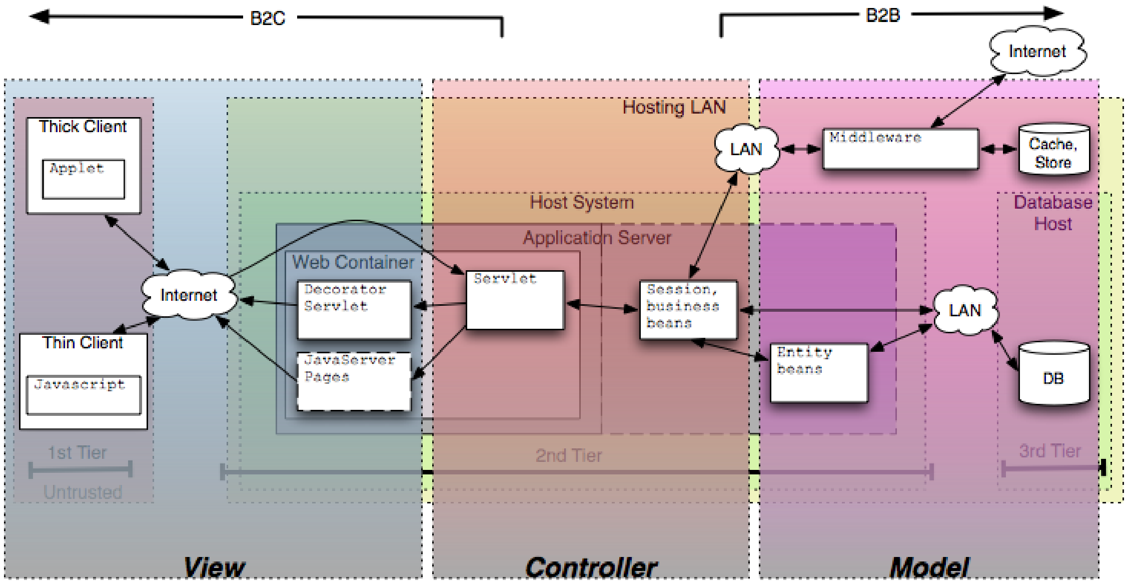
Showing frameworks indicates where important service contracts exist ‘up’ and ‘down’



Identifying the Attack Surface as a Developer

- Struts1
- Struts2
- Spring?

1.3 - Annotate with design patterns



cigital

Design Patterns, isn't that a bit Hifalutin?

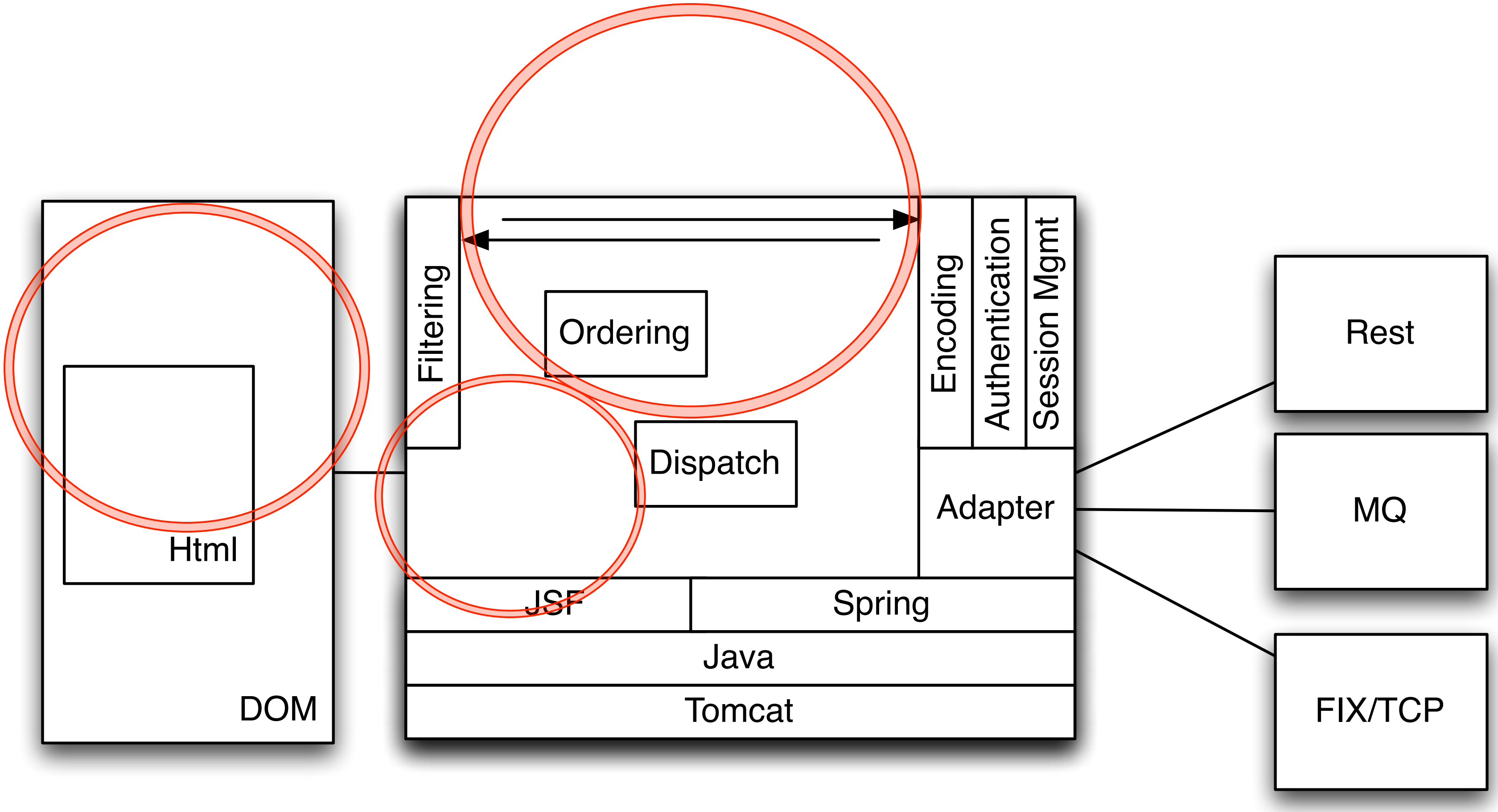
- I'm supposed to find exploits
- Besides, I don't have good design docs
- These guys do *not* look like security researchers

Once Patterns' Responsibilities Defined

- Find them
- Figure out how they apply
- Evaluate the responsibility (next)
- Decide what common attack patterns apply (later)

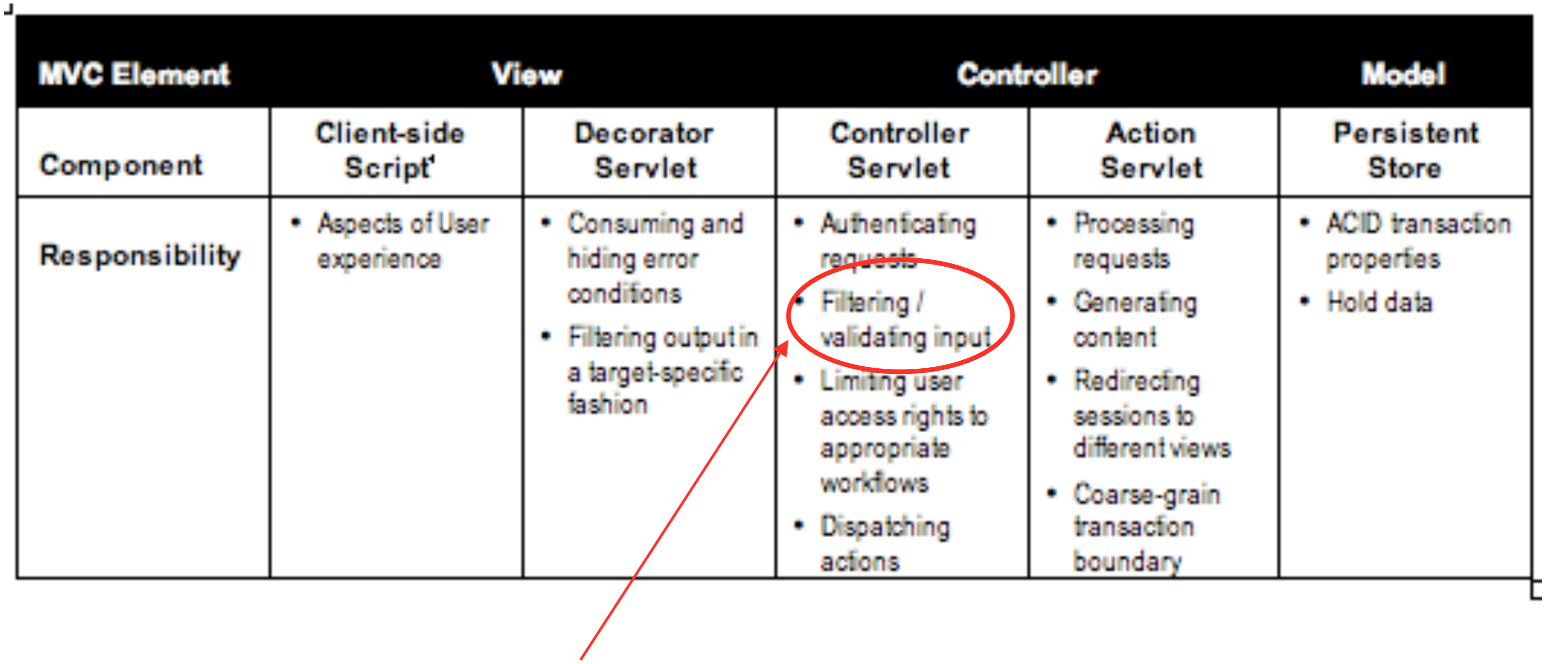
cigital

Exercise: Find repsonsibilities



cigital

1.4 – Consider Patterns’ responsibilities



MVC Element	View		Controller		Model
Component	Client-side Script’	Decorator Servlet	Controller Servlet	Action Servlet	Persistent Store
Responsibility	• Aspects of User experience	• Consuming and hiding error conditions • Filtering output in a target-specific fashion	• Authenticating requests • Filtering / validating input • Limiting user access rights to appropriate workflows • Dispatching actions	• Processing requests • Generating content • Redirecting sessions to different views • Coarse-grain transaction boundary	• ACID transaction properties • Hold data

- Document specific standards for implementing each responsibility

Explicit Responsibilities Mean Better Advice

Client Side

- User Interface
- Responsive, instant
- Apply validation
  - Perhaps imperfect
  - Perhaps quickly
- Give the user *good* advice
  - Be as specific as possible
  - Help the user

Server side

- Business logic
- Decode
- Canonicalize
- Apply
  - Known-good
  - White-list
  - Black list
- Respond to attack
  - Defend self
  - Retain intelligence
  - Monitor
  - Prevent future attack

cigital

Configuration (Declaratively)

```
<bean id=“UserNameValidator"
class=“org.springframework.petclinic.web.UserNameValidator" />

<bean id=“AddUserForm”
class=“org.springframework.petclinic.web.AddUserForm”>
 <property name="validator" ref=“UserNameValidator" />
```

Programmic (imperatively)

```
@RequestMapping(method = RequestMethod.POST)
public String processSubmit(@ModelAttribute Owner owner,
                            BindingResult result,
                            SessionStatus status) {
        new OwnerValidator().validate(owner, result);
        if (result.hasErrors()) {
                return "ownerForm";
        }
        else {
                this.clinic.storeOwner(owner);
                status.setComplete();
                return "redirect:owner.do?ownerId=" + owner.getId();
        }
```

Aspect-Oriented (Constraint Validation)

```
@NotBlank
@Pattern(regexp="^[a-zA-Z_-]*")
@Size(min=8, max=15)
@Constraint(validatedBy = UserNameValidator.class) //mixed!!!
private String userName;

public String getUserName(){
       return this.userName;
}

public void setUserName(String userName){
       this.userName = userName;
}
```

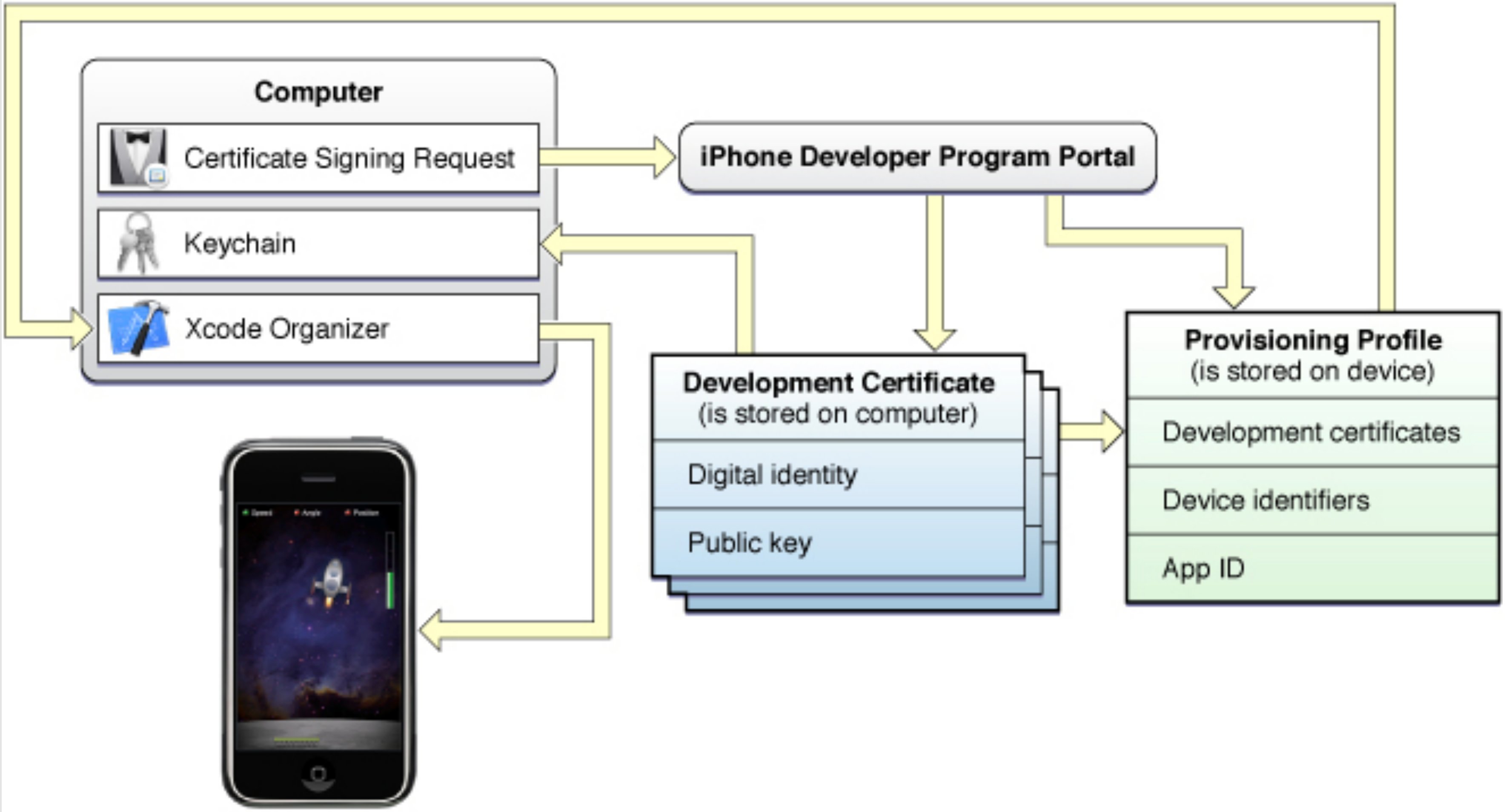
Remediation Advice

- Use declarative model

```
<validator name=”pwCharSet"
classname=”org.myorg.PWCharSetValidator"
method="validatePWCharSet" msg="errors.pwChars"/>

<field property="password" depends="required, pwCharSet">
        <arg0 key="typeForm.password.displayname"/>
        <var> <var-name>Password</var-name>
        <var-value>password</var-value> </var>
</field>
```

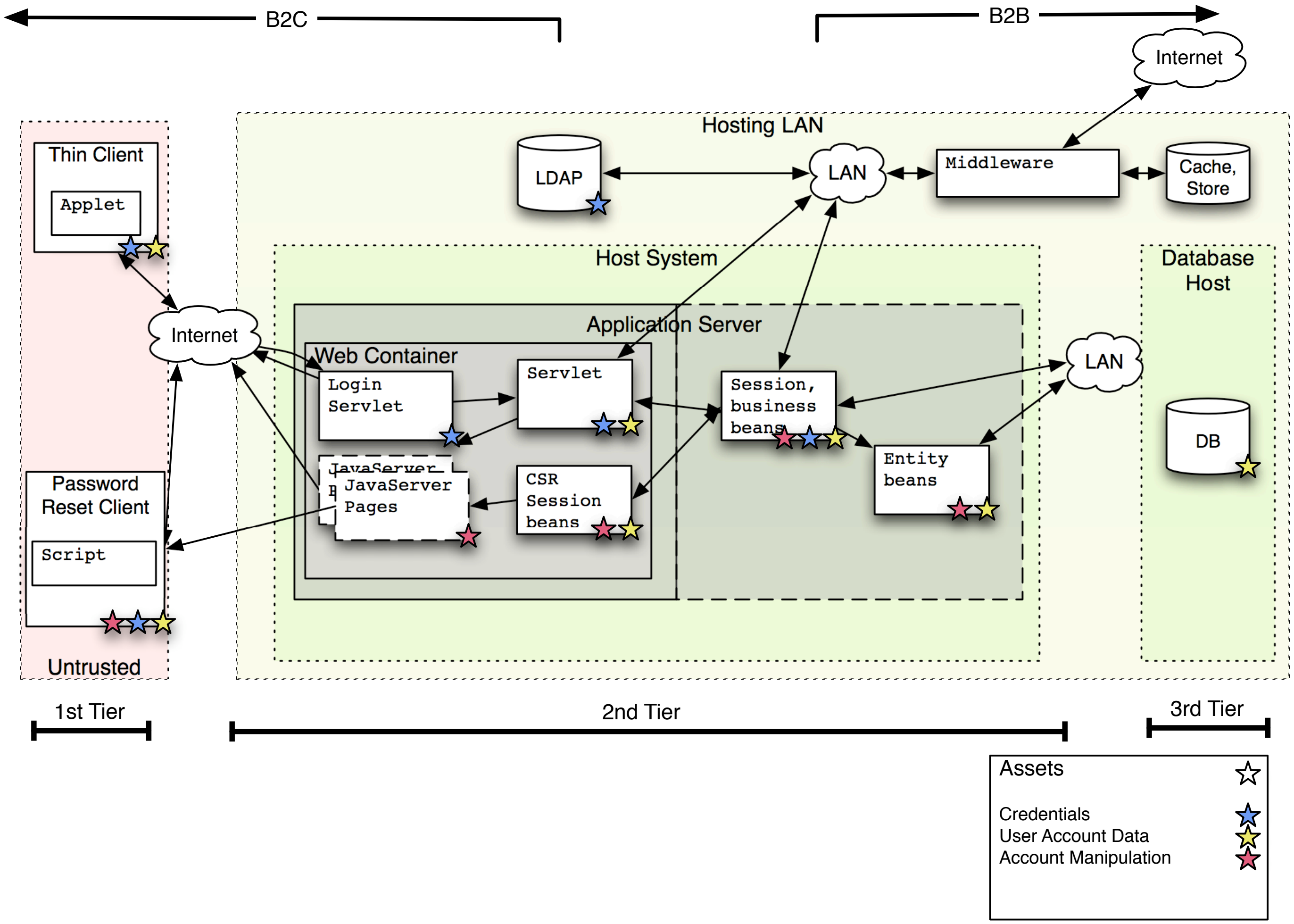
- Encapsulate validators as ‘plugins’
- Chain validator use with `depends=`

1.6 – Identify Controls Explicitly



2.3 – Identify Assets flow through the system

Assets exist not only in rest, but also flow through the system

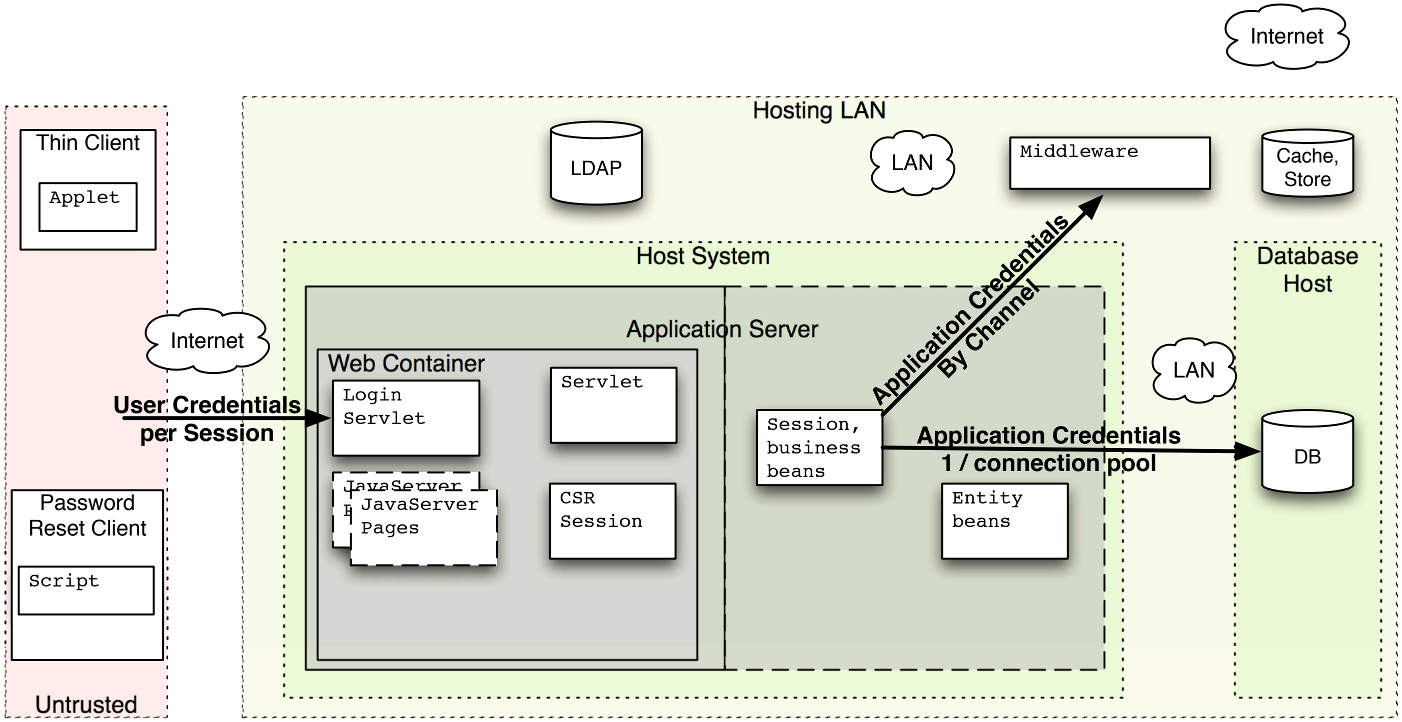


Encapsulation: Struts, Spring

```
<s:form action='Cart'>
               <s:textfield name='quantity' label='Quantity' />
               <m:iterate_items collection=“%=
org.myorg.Skeleton.StoreInventory.getStoreInventory(true) %>"/>

Purse: <c:property name="purse.value" /><br>
<s:submit/></s:form>

<!— By compound property -->
<bean id=”person” class=”org.myorg.app.Person”>
        <property name=”SocialSecurityNumber” value=”555555555”>
</bean>
```

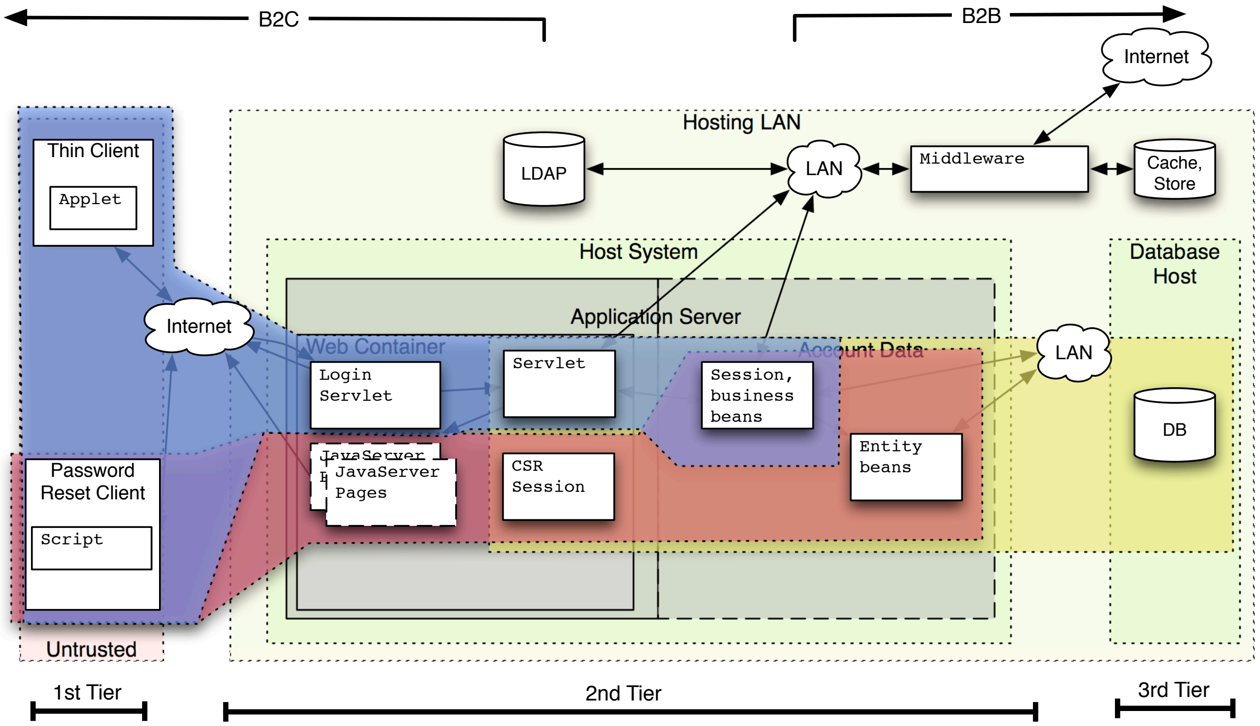
3.2 – Identity Principal Resolution

Arrows indicate resolution of principal/assertion propagation



3.4 – Show Authorization in Structure

Coloration shows authorization by role



Authorization, Where it occurs

```
<authz:authorize ifAllGranted="ROLE_SUPERVISOR">
  <td>
    <A HREF="delete.jsp?id=<c:out value="${contact.id}"/>">Delete</A>
  </td>
</authz:authorize>
```

```
<bean id="contactServiceMethodProtection"
class="org.acegisecurity.intercept.method.aopalliance.MethodSecurityInterceptor">
        <property name="validateConfigAttributes">
            <value>true</value>
        </property>
        <property name="authenticationManager">
            <ref bean="providerManager"/>
        </property>
        <property name="accessDecisionManager">
            <ref local="methodAccessDecisionManager"/>
        <property name="objectDefinitionSource">
                              <value>

          com.myorg.service.ContactService.deleteContact=ROLE_SUPERVISOR
                              ...
                              </value>
                    </property>
          </bean>
```

Authentication, Where it occurs

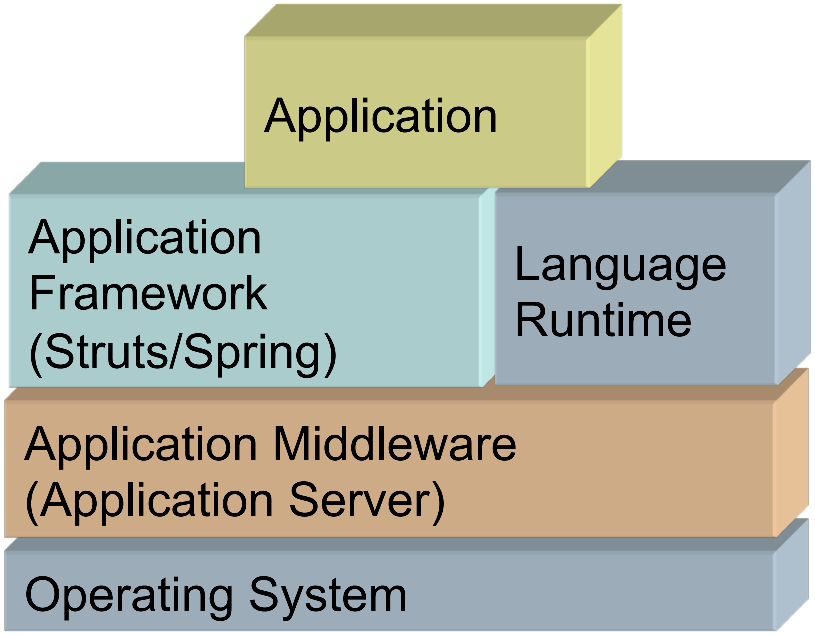
```
JSONRPCBridge json_bridge = (JSONRPCBridge) session.getAttribute
("JSONRPCBridge");
json_bridge.registerObject("Authentication",
SecurityContextHolder.getContext().getAuthentication());

function retrieveCredential()
{
    try {
        jsonrpc = new JSONRpcClient("/org/JSON-RPC");
         // Call a Java method on the server
         var result = jsonrpc.Authentication.getCredentials();
         alert(result);
    } catch(e) {
        alert(e);
    }
}
```

Underlying Framework Analysis

Software, Software Everywhere

Dependencies on Underlying Framework

Software is built upon layers of other software


Application
Application Framework (Struts/Spring)
Language Runtime
Application Middleware (Application Server)
Operating System

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

Framework Security Controls

- The application environment provides controls. What are the limitations?
  - Cryptography
    - Example: JCA
  - Authentication and Authorization
    - Example: JAAS
  - Sandboxing
    - JavaScript Same Origin Policy

cigital

Session Management

- In Web.xml
  - `<httpCookies httpOnlyCookies="true" …>`

- In code:

  -
    ```
    String sessionid = request.getSession().getId();
    response.setHeader("SET-COOKIE", "JSESSIONID=" + sessionid
    + "; HttpOnly");
    ```

cigital

JCA

- Check:
  - Cipher being used is appropriate for job
  - IV
    - 00000000?
    - Hard-coded?
  - Padding
  - Mode

ARA Is About Identifying Flaws

FLAWS - Design

- Misuse of cryptography
- Duplicated data or code
- Lack of consistent input validation
- Missing authorization checks
- Insecure or lack of auditing
- Lack of authentication or session management on APIs
- Missing compartmentalization
- Assigning too much privilege or failing to give up privilege

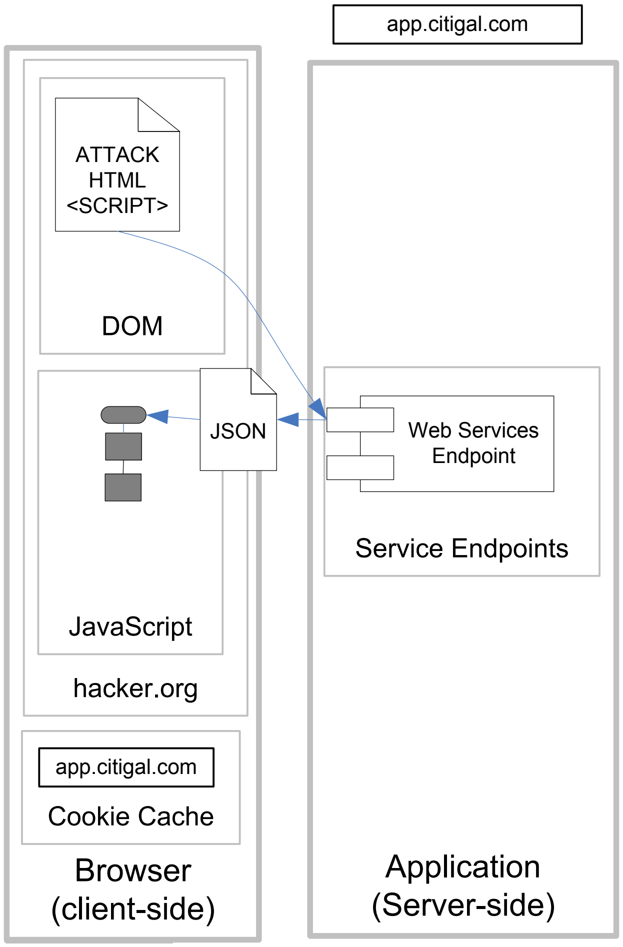
Remediation Advice

```
User  <%= ESAPI.encoder().encodeForHTML(user.getName()) %>!

<img src="/profile/photo?user=<%= ESAPI.encoder().encodeForURL(user.getId()) %>"
    alt="<%= "Photo of "+ESAPI.encoder().encodeForHTMLAttribute(user.getId()) %>"
    onclick="<%= "openProfile('"+ESAPI.encoder().encodeForHTMLAttribute(
                ESAPI.encoder().encodeForJavaScript(user.getId())) + "')" %>" />
```

- Use a context aware encoder, just as JXT:
  - http://code.google.com/p/owasp-jxt/
  - Uses `{user.getName()}` style syntax

cigital

JavaScript Hijacking



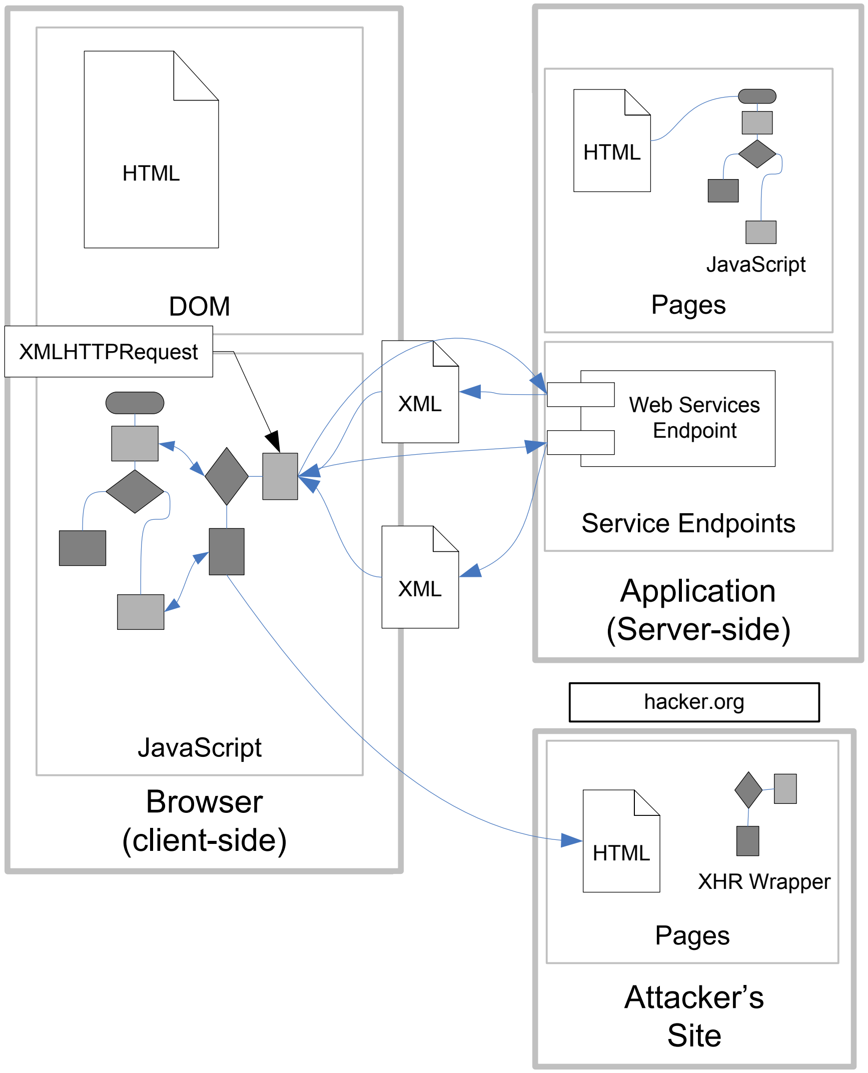
- JavaScript Hijacking requires that the application return JSON objects
- The attacker loads the attack script into the JavaScript environment
- The attacking page uses a <SCRIPT> tag to make the cross page reference

Remediation Advice

- Framework like Caja (http://code.google.com/p/google-caja/)
  - And *careful* application
- Scopes or removes:
  - eval,
  - Function, Function.constructor
  - with
- Freezes objects

cigital

Ajax Interposition



1. Modify the XMLHTTPRequest prototype

```
var xmlreqc=XMLHttpRequest;
XMLHttpRequest = function() {
this.XHR = new xmlreqc();
return this;
}
```

2. Wrap the send method

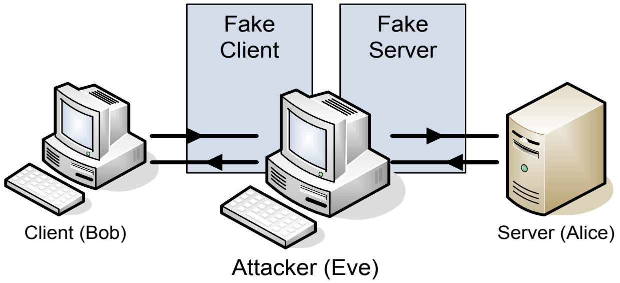
```
XMLHttpRequest.prototype.send =
     function (content){
//..add code to steal or alter
     content
Sniff_and_Modify(content);
// Pass call on
return this.XHR.send(pay);
}
```

APIs Across Stateless Protocols

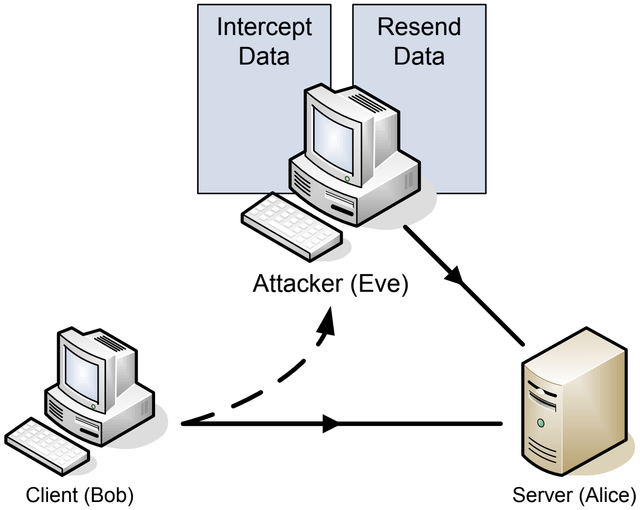
- Identifiers representing state can be abused
  - Prediction
  - Capture
  - Fixation
- State sent to the client between requests is altered or replayed
- Relevant Attack Patterns
  - Session hijacking/fixation
  - CSRF
  - Message Replay
  - Parameter manipulation

Distributed Architecture

- Distributed systems are susceptible to network-based attacks
  - Eavesdropping
  - Tampering
  - Spoofing
  - Hijacking
  - Observing
- Relevant Attack Patterns
  - Interposition attacks
  - Network sniffing
  - Replay attacks



Interposition Attack



Replay Attack

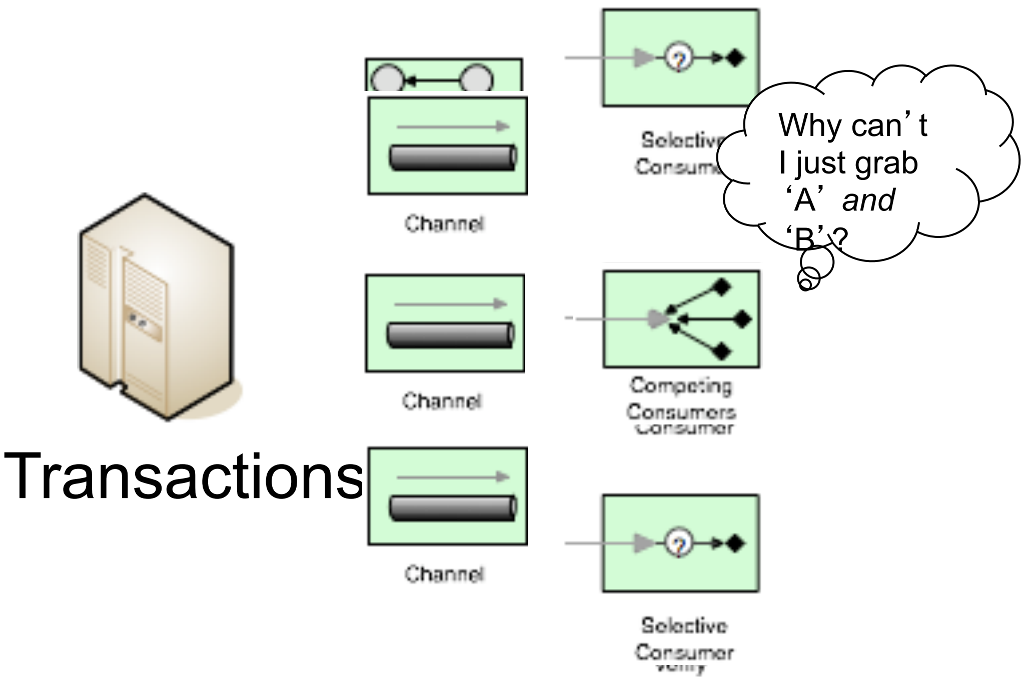
Dynamic Code Generation and Interpretation

- 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?
- Relevant Attack Patterns
  - Cross Site Scripting (XSS)
  - SQL Injection
  - Buffer overflow
  - XML Injection
  - Shell command Injection
  - Cross-Site Request Forgery (CSRF)

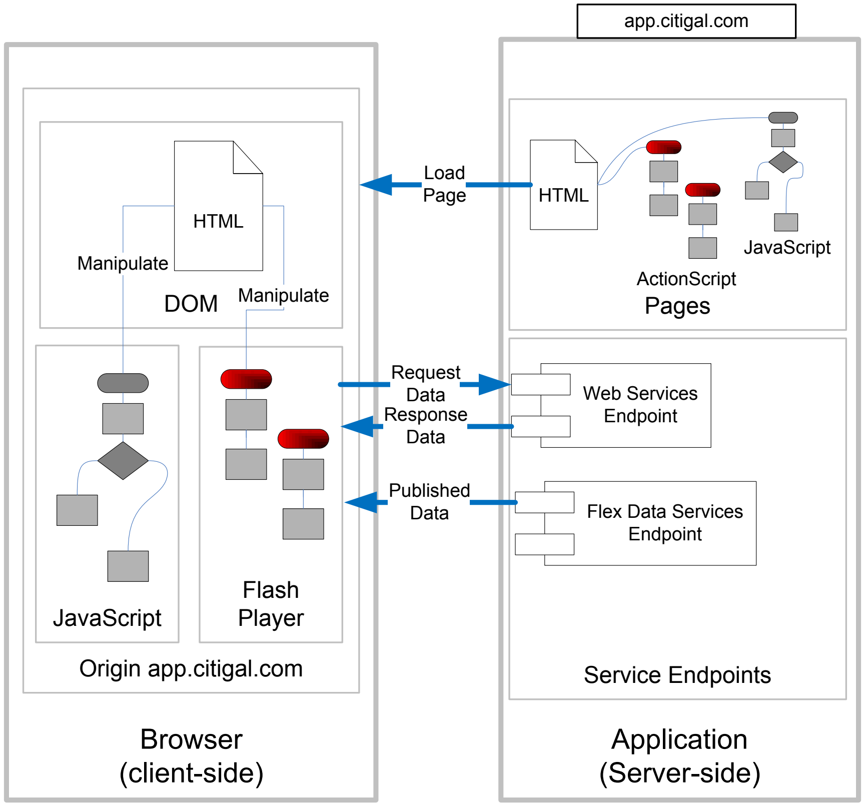
Service-oriented Architecture (SOA)

- 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
- Relevant Attack Patterns:
  - XML Injection / SQL Injection
  - Session Management Attacks
  - Direct File Manipulation

ARA's find 'Flaw's



Rich Internet Applications



- Processing moves to the client-side
- Relevant Attack Patterns
  - Direct API calls
  - CSRF
  - XSS
- Unique Attacks
  - JavaScript Hijacking
  - Ajax Interposition

Pass tech.-specific KM by REFERENCE

- Do not duplicate technical resources in your T.M., that' s a later step.

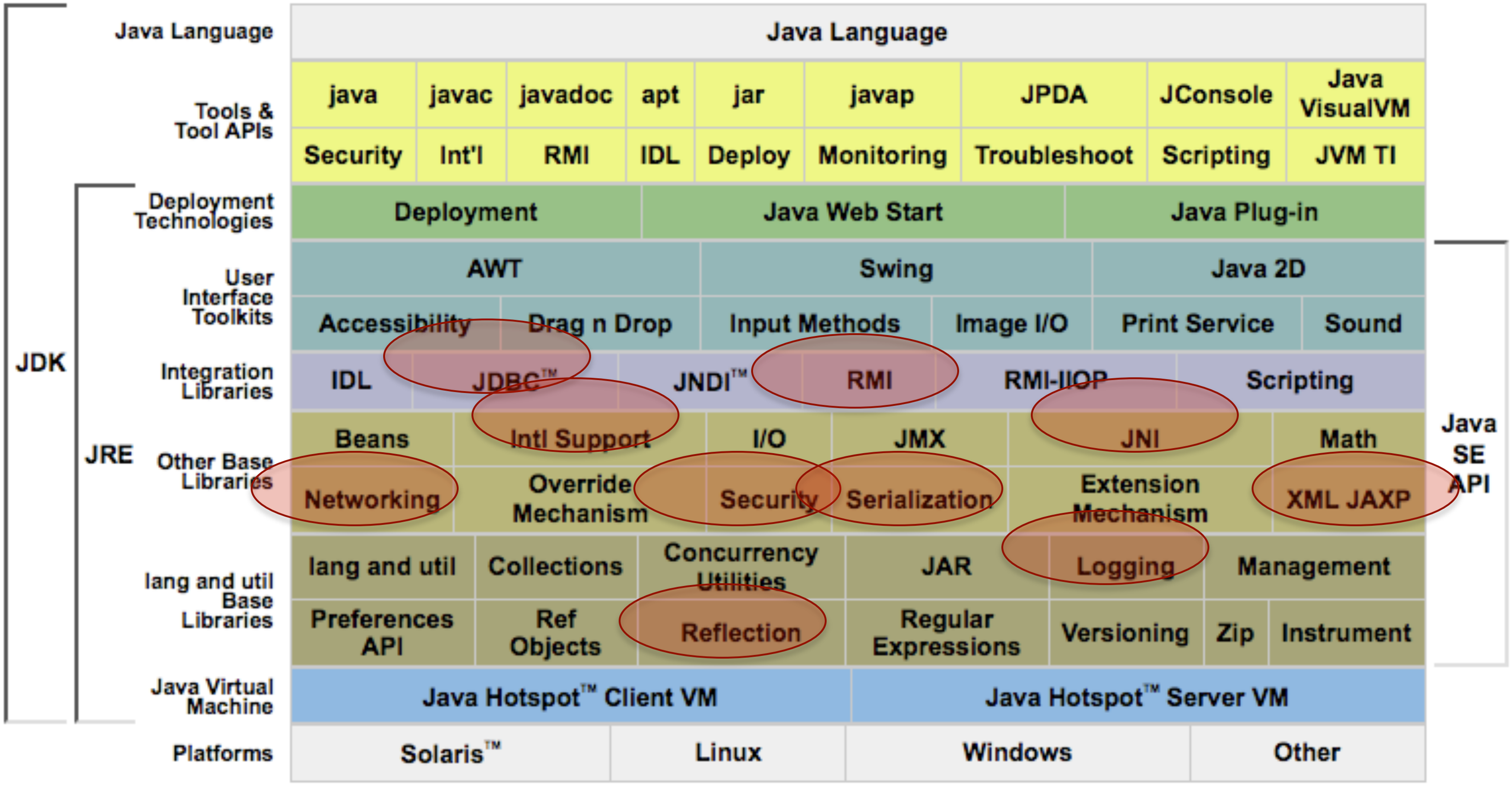
- Reference:
  - Code review guide:
    - http://www.owasp.org/index.php/Code_Review_Guide_Frontispiece
  - Testing guide:
    - http://www.owasp.org/index.php/Category:OWASP_Testing_Project

cigital

Critical Functionality Pointers

- Based on idiom/paradigm
- Control Patterns
  - Command Patterns
  - Inversion of Control containers
  - Session Management and other flow-drivers
- Underlying frameworks
  - Callbacks
  - Plugins
  - Frameworks
- Security features

Exercise: Key Structural Elements of Java EE Apps



JDK

JRE

Java Language: Java Language

Tools & Tool APIs: java, javac, javadoc, apt, jar, javap, JPDA, JConsole, Java VisualVM

Security, Int'l, RMI, IDL, Deploy, Monitoring, Troubleshoot, Scripting, JVM TI

Deployment Technologies: Deployment, Java Web Start, Java Plug-in

User Interface Toolkits: AWT, Swing, Java 2D

Accessibility, Drag n Drop, Input Methods, Image I/O, Print Service, Sound

Integration Libraries: IDL, JDBC™, JNDI™, RMI, RMI-IIOP, Scripting

Other Base Libraries: Beans, Intl Support, I/O, JMX, JNI, Math

Networking, Override Mechanism, Security, Serialization, Extension Mechanism, XML JAXP

lang and util Base Libraries: lang and util, Collections, Concurrency Utilities, JAR, Logging, Management

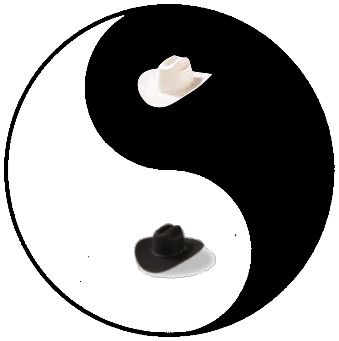
Preferences API, Ref Objects, Reflection, Regular Expressions, Versioning, Zip, Instrument

Java Virtual Machine: Java Hotspot™ Client VM, Java Hotspot™ Server VM

Platforms: Solaris™, Linux, Windows, Other

Java SE API

Thank you for your time.



cigital