You Are Hacked ☹ End-to-End Java EE Security in Practice

Karthik Shyamsunder, Principal Technologist
Phani Pattapu, Engineer

Who Are We?

- Karthik Shyamsunder
  - Principal Technologist, Verisign
  - Adjunct Faculty, Johns Hopkins University
  - Twitter @kshyamsunder
  - Email karthik.shyamsunder@gmail.com

- Phani Pattapu
  - Software Engineer, Verisign
  - Twitter @phanipattapu
  - Email phani@pattapu.com

Overall Presentation Goal

1. Overview of Java EE security features that can help you in building secure enterprise applications

2. Best practices that you can employ while using Java EE security features

Outline

- The Internet Threat Model
- Java EE Security Model
- Web Tier Security
- EJB Tier Security
- EIS Tier Security
- Java EE Security Challenges
- Summary
- Questions

Outline

- **The Internet Threat Model**
- Java EE Security Model
- Web Tier Security
- EJB Tier Security
- EIS Tier Security
- Java EE Security Challenges
- Summary
- Questions

Headlines, Headlines, Headlines!

Crime
Internet scammers r
Customers wary of online banking
Hackers truste
Hackers hi
"What's been taken is bits of data that the
might put together into an identity."

From Computer Desktop Encyclopedia
Reproduced with permission.
© 2000 The Computer Language Co. Inc.

Hackers hit two more major sites, rock market

INQUIRER WIRE SERVICES

WASHINGTON — Attorney General Janet Reno announced a criminal investigation yesterday into the latest wave of hacker attacks on major Internet sites, as law enforcement officials conceded that they had little idea of who or what they

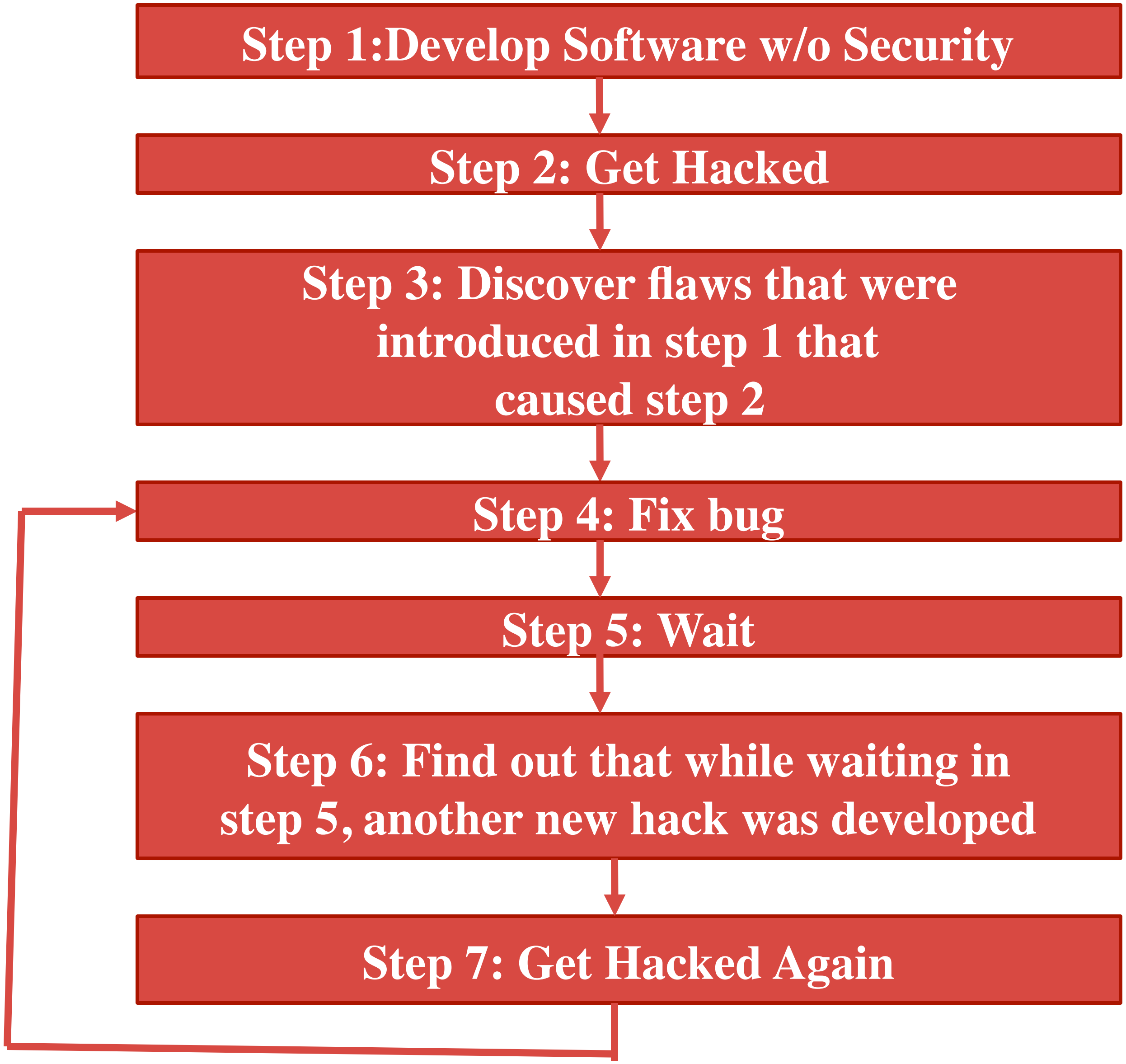
The Attacks ?

- **Who initiates the attack?**
  - Hackers, Crackers, Cyber Terrorists, Script Kiddies, Competitors, Industrial Spies, Foreign Countries
- **Why do they attack?**
  - Intellectually Motivated, Personally Motivated, Socially Motivated, Politically Motivated, Financially Motivated, Ego
- **Damage of an Attack**
  - Financial, Customers losing trust, Legal, Loss of Data, Bad Publicity

The Problem is Real

- **Cyber crimes and incidents are on the rise**
- **3 out of 4 business web sites are vulnerable to attack (Gartner)**
- **75% of the hacks occur at the application level (Gartner)**

Seven Steps of Doom



Step 1:Develop Software w/o Security

Step 2: Get Hacked

Step 3: Discover flaws that were introduced in step 1 that caused step 2

Step 4: Fix bug

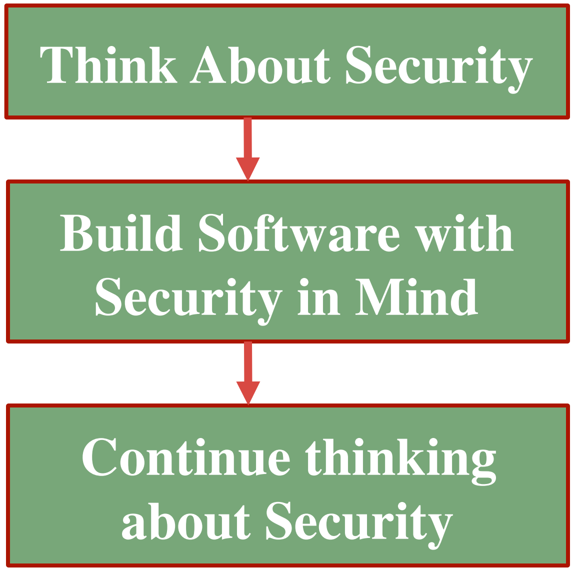
Step 5: Wait

Step 6: Find out that while waiting in step 5, another new hack was developed

Step 7: Get Hacked Again

Steps to Success

- Secure Design
- Secure Development
- Secure Testing
- Secure Deployment & Operations
- Audit Process



Think About Security

Build Software with Security in Mind

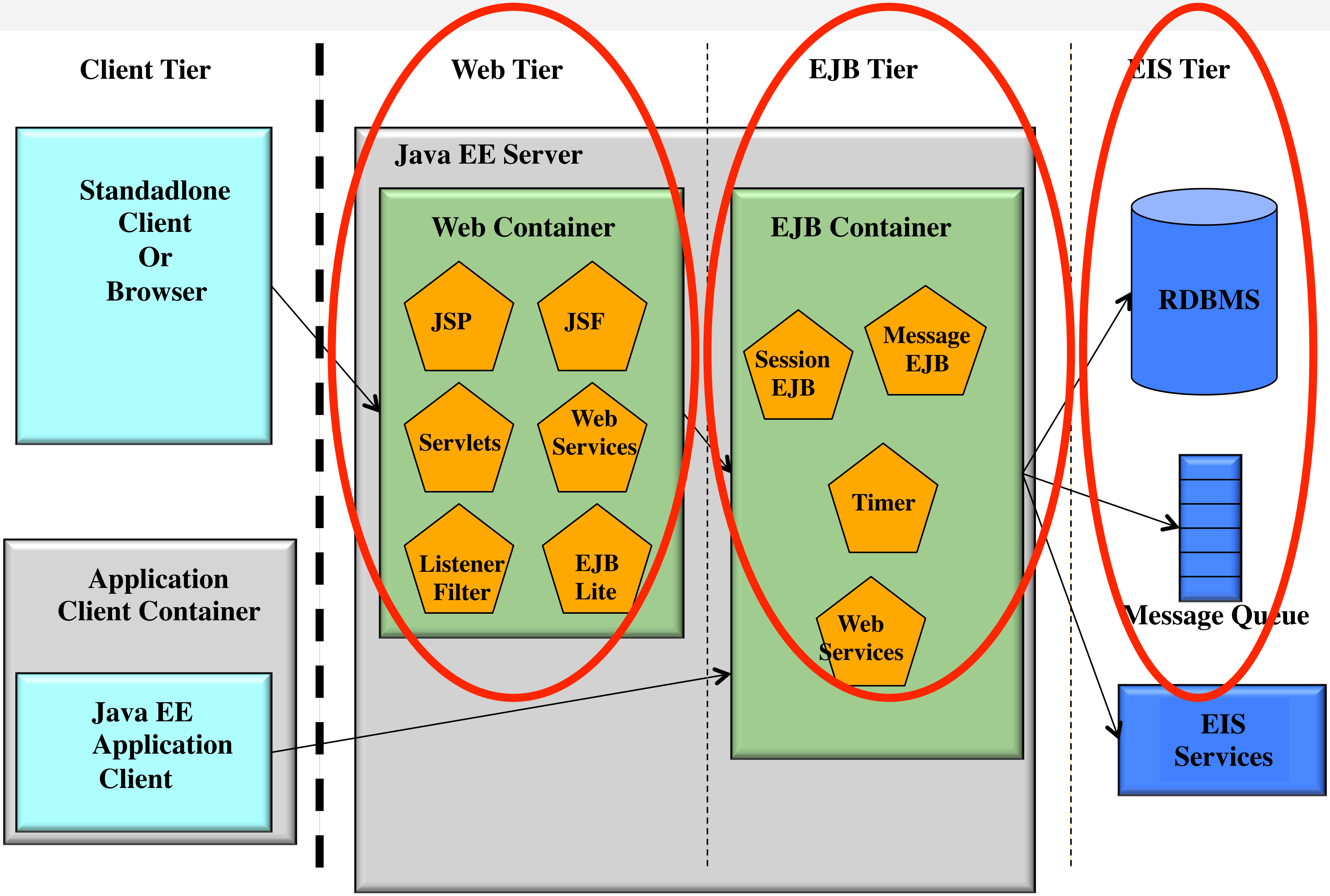
Continue thinking about Security

Build your Software on a Platform that Gives you Security Features

Outline

- The Internet Threat Model
- **Java EE Security Model**
- Web Tier Security
- EJB Tier Security
- EIS Tier Security
- Java EE Security Challenges
- Summary
- Questions

Java EE Architecture



Characteristics of App Security

- **Authentication**
  - Ensuring that users are who they say they are
- **Authorization, or Access control**
  - Ensuring users have permission to perform operations
- **Confidentiality, or Data privacy**
  - Ensuring that only authorized users can view sensitive data
- **Non-repudiation**
  - Ensuring that transactions can be proved to have happened
- **Auditing**
  - Maintaining record of transactions and security information
- **Quality of Service**
  - Ensuring that the users experience a good quality of service

Java EE App Server offers several mechanism to achieve these security characteristics

Two ways to express Security

1. **Declarative**
   1. Java Annotations
      - Advantage is easier to apply security as you are coding
      - Limited functionality compared to deployment descriptors
   2. Through Deployment descriptors
      - Advantage is security concern is kept outside the code
      - Can override security features defined through annotation
2. **Programmatic**
   - May times when declarative security alone may not be sufficient to express the application logic, in which programmatic security comes to the rescue
   - Security logic is embedded inside the applications and is used to make security decisions

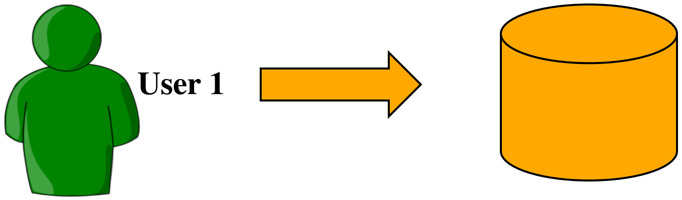
Prefer using Declarative Security, and fall back into Programmatic security when it is not sufficient for the application

Java EE Security Terminology

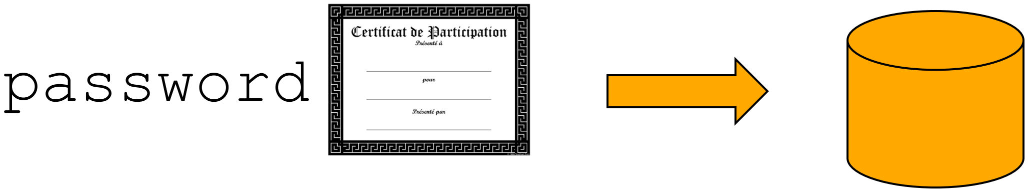
- **In order to achieve the Application Security in the Java Platform, Java EE specifically defines**
  - User
  - Credential
  - Group
  - Identity Storage
  - Security Realm
  - Principal
  - Role

User, Credential and Group

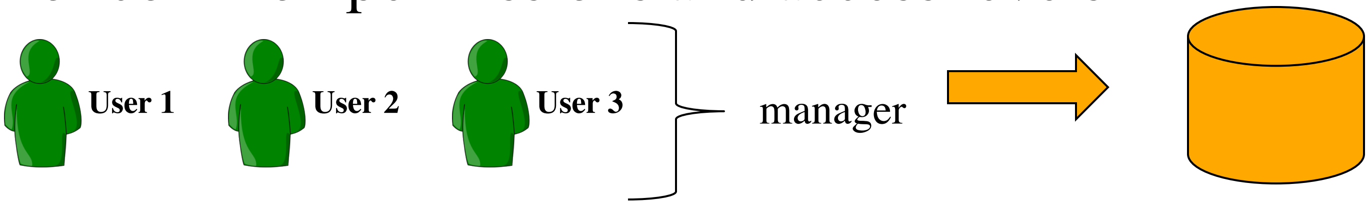
- **User**
  - An individual(or machine), whose identity is defined in the identity storage



- **Credential**
  - Information used to authenticate a user
  - Typically a password, sometimes a client certificate

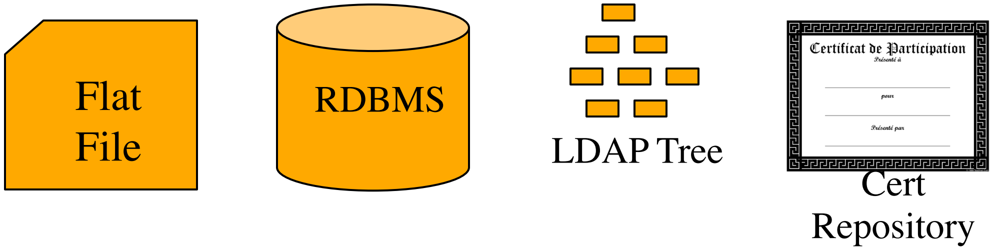


- **Group**
  - Set of users classified with a set of common characteristics -> a set of common permissions and access levels



Identity Storage and Security Realm

- **Identity Storage**
  - Place where you store user, group and credential



Flat File

RDBMS

LDAP Tree

Cert Repository

- **Security Realm**
  - Mechanism by which an application server stores user's credential and group information
  - Different application servers use different methods to map users, groups and roles to each other
  - Glassfish provides FILE, JDBC, LDAP, CERT
    - Also provides a few other ones like PAM for unix
    - If you want, you can also the app server specific API to provide your own custom implementation

Take a look at JDBC Realm

```
# USERS table
CREATE TABLE `users` (
  `user_id` int(10) NOT NULL AUTO_INCREMENT,
  `username` varchar(10) NOT NULL,
  `first_name` varchar(20) DEFAULT NULL,
  `middle_name` varchar(20) DEFAULT NULL,
  `last_name` varchar(20) DEFAULT NULL,
  `password` char(32) NOT NULL,
  PRIMARY KEY (`user_id`)
);
```

```
# GROUPS table
CREATE TABLE groups (
  `group_id int(10) NOT NULL,
  `group_name` varchar(20) NOT NULL,
  `group_desc` varchar(200) DEFAULT NULL,
  PRIMARY KEY (`group_id`)
);
```

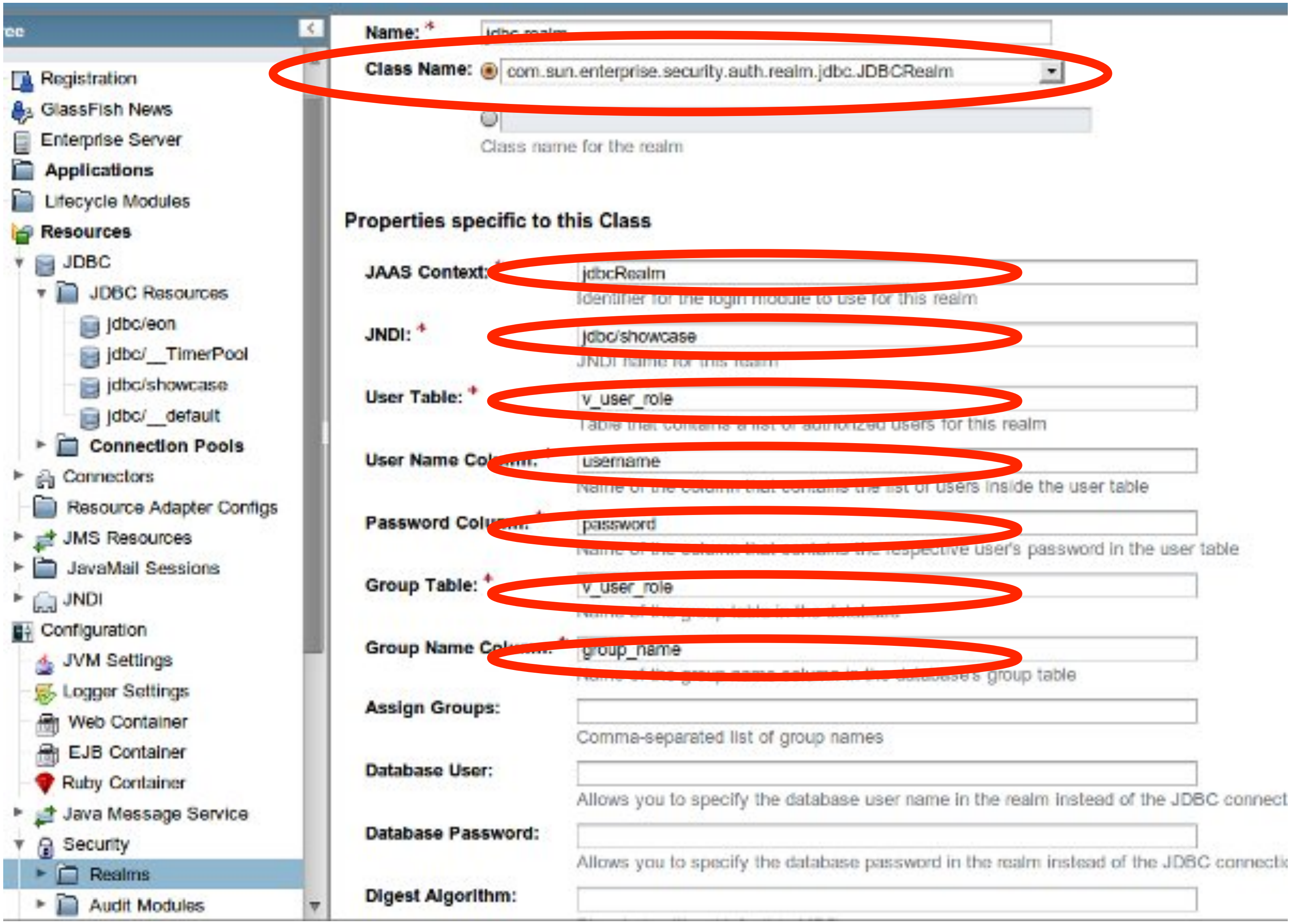
```
#USERS_GROUPS JOIN TABLE

CREATE TABLE `user_groups` (
  `user_id` int(10) NOT NULL,
  `group_id` int(10) NOT NULL,
   PRIMARY KEY (`user_id`,`group_id`),
);
```

Realm to RDBMS Table Mapping in Glassfish

```
CREATE VIEW `v_user_role` AS
  SELECT  u.username, u.password, g.group_name
  FROM `user_groups` ug
   INNER JOIN `users` u ON u.user_id = ug.user_id
   INNER JOIN `groups` g ON g.group_id =  ug.group_id;
```

```
INSERT  INTO `groups`(`group_id`,`group_name`,`group_desc`) VALUES
  (1,'USER','Regular users'),
  (2,'ADMIN','Administration users');

INSERT  INTO `users`(`user_id`,`username`,`first_name`,`middle_name`,
`last_name`,`password`) VALUES
  (1,'john','John',NULL,'Doe','6e0b7076126a29d5dfcbd54835387b7b'),
  (2,'admin',NULL,NULL,NULL,'21232f297a57a5a743894a0e4a801fc3');

INSERT  INTO `user_groups`(`user_id`,`group_id`) VALUES (1,1),(2,1),(2,2);
```

Mapping Tables to JDBC Realm



Name: * jdbc-realm

Class Name: ● com.sun.enterprise.security.auth.realm.jdbc.JDBCRealm

○

Class name for the realm

Properties specific to this Class

JAAS Context: * jdbcRealm
Identifier for the login module to use for this realm

JNDI: * jdbc/showcase
JNDI name for this realm

User Table: * v_user_role
Table that contains a list of authorized users for this realm

User Name Column: * username
Name of the column that contains the list of users inside the user table

Password Column: * password
Name of the column that contains the respective user's password in the user table

Group Table: * v_user_role
Name of the group table in the database

Group Name Column: * group_name
Name of the group name column in the database's group table

Assign Groups:
Comma-separated list of group names

Database User:
Allows you to specify the database user name in the realm instead of the JDBC connect

Database Password:
Allows you to specify the database password in the realm instead of the JDBC connectio

Digest Algorithm:

Principal and Role

- **Principal**
  - Java EE concept to represent a user, but is more generic to handle client side certs besides just the username
  - java.security.Principal represents a Principal

- **Role**
  - Java EE concept to define access levels
  - Java EE Roles are mapped to users and groups using vendor specific configuration files
  - Application developer specifies which roles can access which set of the application functionalities

Mapping Users & Groups to Roles

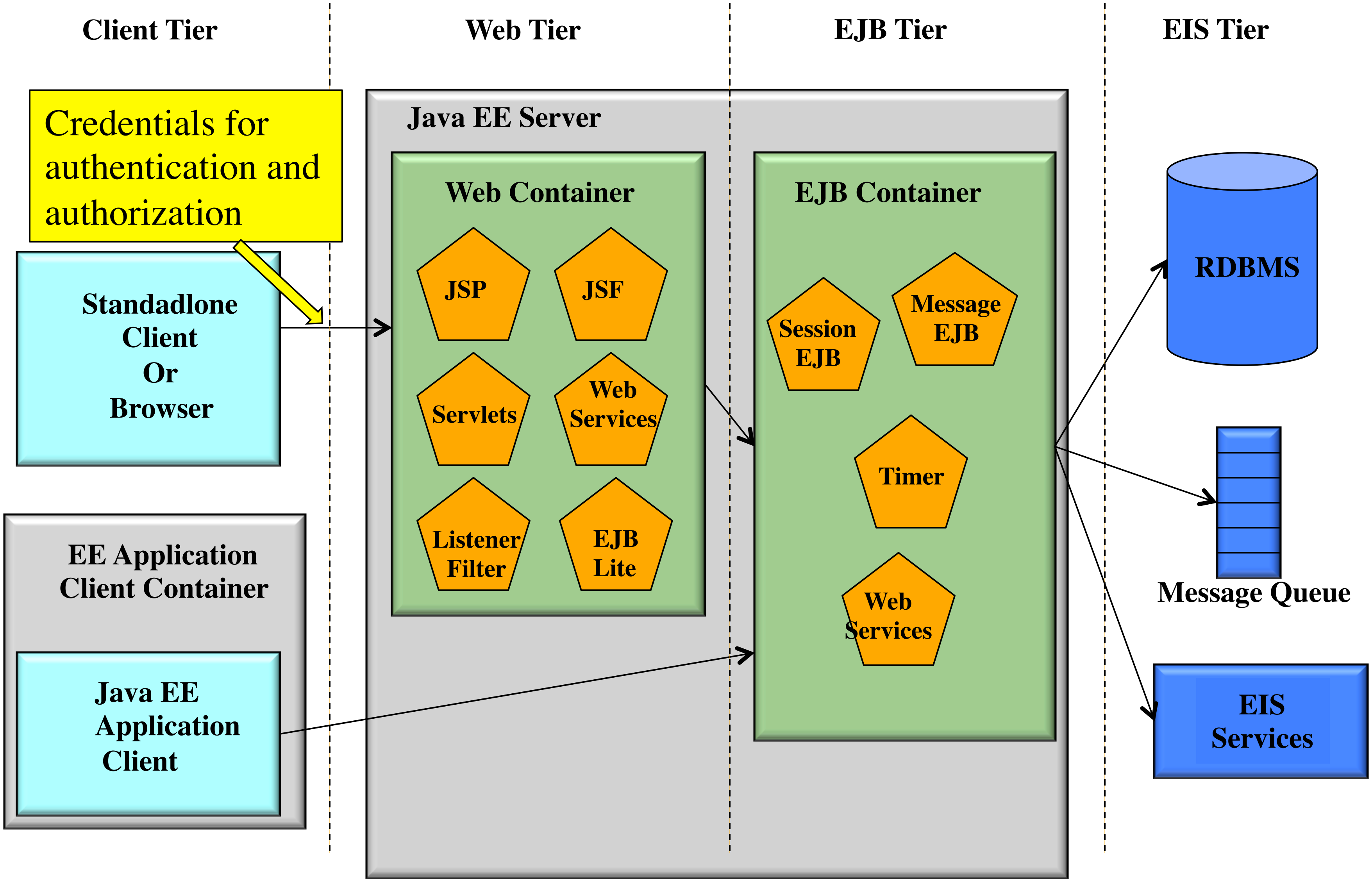
- **Mapping**
  - Essential to map the users and groups -> Java EE roles
  - Typically done in vendor specific configuration files files

```
<glassfish-web.xml>

        <security-role-mapping>
                <role-name>manager</role-name>
                <group-name>team-leads</group-name>
                <principal-name>ppattapu</principal-name>
                <principal-name>kshyamsu</principal-name>
        </security-role-mapping>

        <security-role-mapping>
                <role-name>administrator</role-name>
                <principal-name>ppattapu</principal-name>
        </security-role-mapping>

</glassfish-web.xml>
```

Outline

- The Internet Threat Model
- Java EE Security Model
- **Web Tier Security**
- EJB Tier Security
- EIS Tier Security
- Java EE Security Challenges
- Summary
- Questions

Accessing the Web Tier



Credentials for authentication and authorization

Authentication in Web Tier

- Authentication Methods
  - HTTP BASIC Authentication
  - HTTP DIGEST Authentication
  - Form Based Authentication
  - Client Certificate Authentication
- Important to Understand the pros/cons of each model
- Can be achieved
  - Declaratively
    - XML based only
    - No annotation based authentication exists
  - Programmatically (recent API)

Copyright 2002 by Randy Glasbergen. www.glasbergen.com

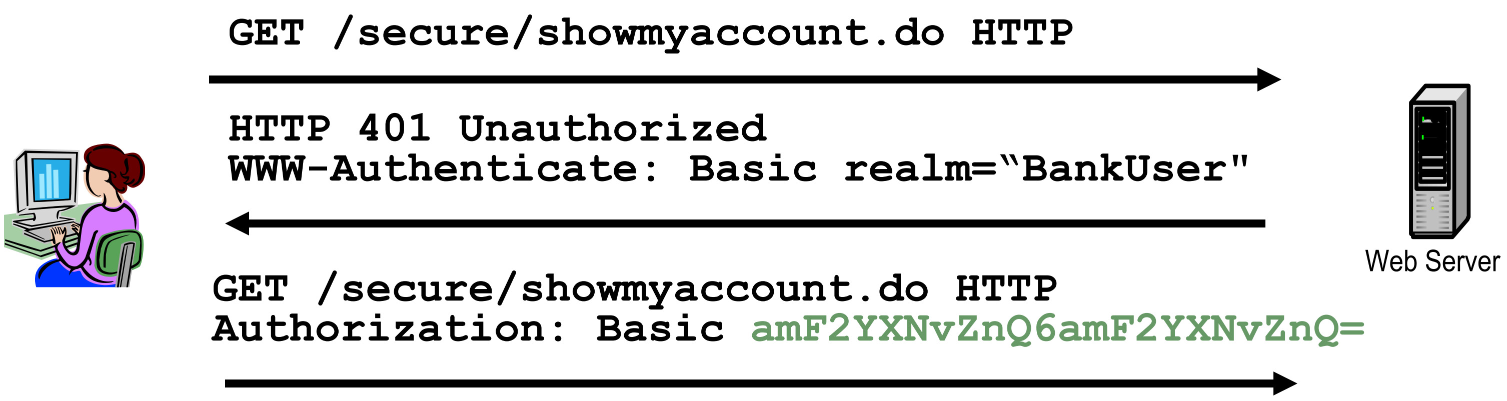
"Someone got my Social Security number off the intern
and stole my identity. Thank God — *I hated being me!"*

Declarative Authentication web.xml

```
<web.xml>

...

<login-config>
  <auth-method>BASIC|DIGEST|CLIENT-CERT|FORM</auth-method>
  <realm-name>name-as-defined-in-appserver</real-name>
  <form-login-config>
    <form-login-page>url-of-login-page</form-login-page>
    <form-error-page>url-of-error-page</form-error-page>
  </form-login-config>
</login-config>


...

</web.xml>
```

Declarative Authentication – Basic

```
<login-config>
   <auth-method>BASIC</auth-method>
</login-config>
```

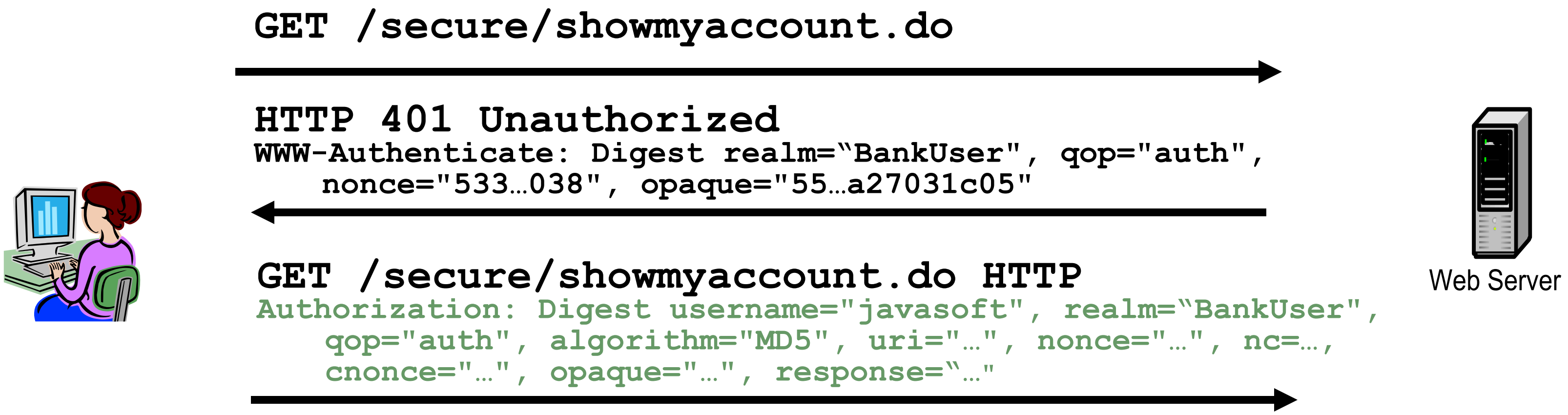


```
GET /secure/showmyaccount.do HTTP

HTTP 401 Unauthorized
WWW-Authenticate: Basic realm=“BankUser"

GET /secure/showmyaccount.do HTTP
Authorization: Basic amF2YXNvZnQ6amF2YXNvZnQ=
```

- Password sent is BASE64Encoded; Not Encrypted

Declarative Authentication – Digest

```
<login-config>
   <auth-method>DIGEST</auth-method>
</login-config>
```



```
GET /secure/showmyaccount.do

HTTP 401 Unauthorized
WWW-Authenticate: Digest realm=“BankUser", qop="auth",
    nonce="533…038", opaque="55…a27031c05"

GET /secure/showmyaccount.do HTTP
Authorization: Digest username="javasoft", realm=“BankUser",
    qop="auth", algorithm="MD5", uri="…", nonce="…", nc=…,
    cnonce="…", opaque="…", response=“…"
```

- Password is not sent to Server; Only DIGEST

Declarative Authentication – Form

- **Declarative Form based auth relies upon**
  - a login page that uses special built in HTML fields to posted to a special server side service
    - j_username and j_userpassword are POST form fields
    - j_security_check is POST submission service that starts the login process

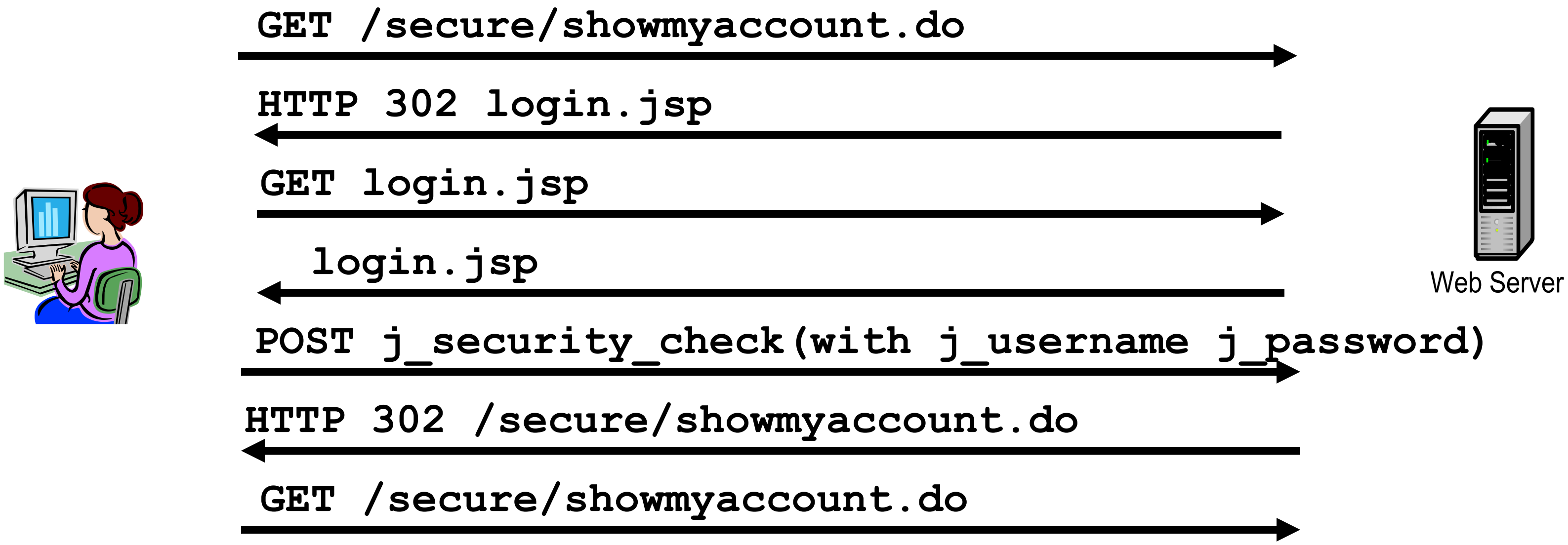
```html
<html>
  ...
  <form method="post" action="j_security_check">
    <input type="text" name="j_username">
    <input type="password" name= "j_password">
  </form>
</html>
```

  - An error page that is displayed when authentication failes

```html
<html>
  ...
  <h1> Login Faile :-( </h1>
</html>
```

Declarative Authentication – Form

```
<login-config>
   <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>login.jsp</form-login-page>
    <form-error-page>error.jsp</form-error-page>
  </form-login-config>
</login-config>
```



GET /secure/showmyaccount.do

HTTP 302 login.jsp

GET login.jsp

login.jsp

POST j_security_check(with j_username j_password)

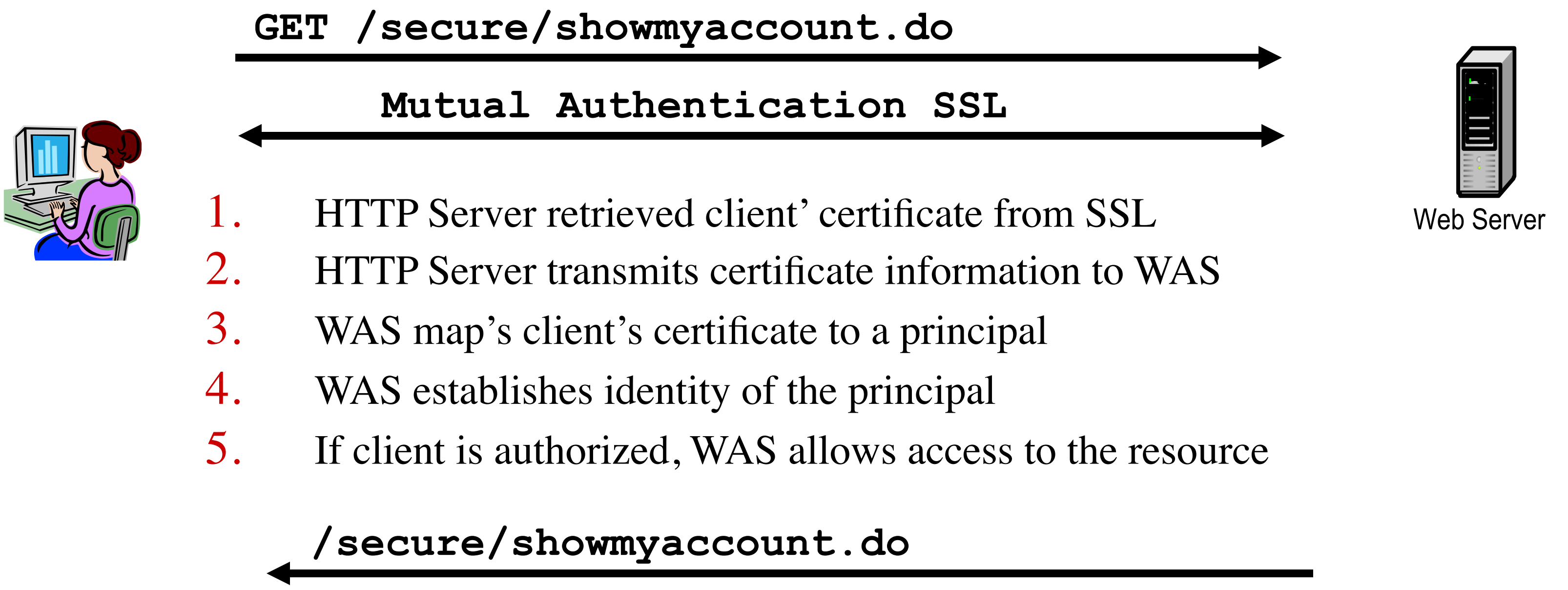
HTTP 302 /secure/showmyaccount.do

GET /secure/showmyaccount.do

- Username/Password sent via POST

Declarative Authentication – Cert

```
<login-config>
   <auth-method>CLIENT-CERT</auth-method>
</login-config>
```



GET /secure/showmyaccount.do

Mutual Authentication SSL

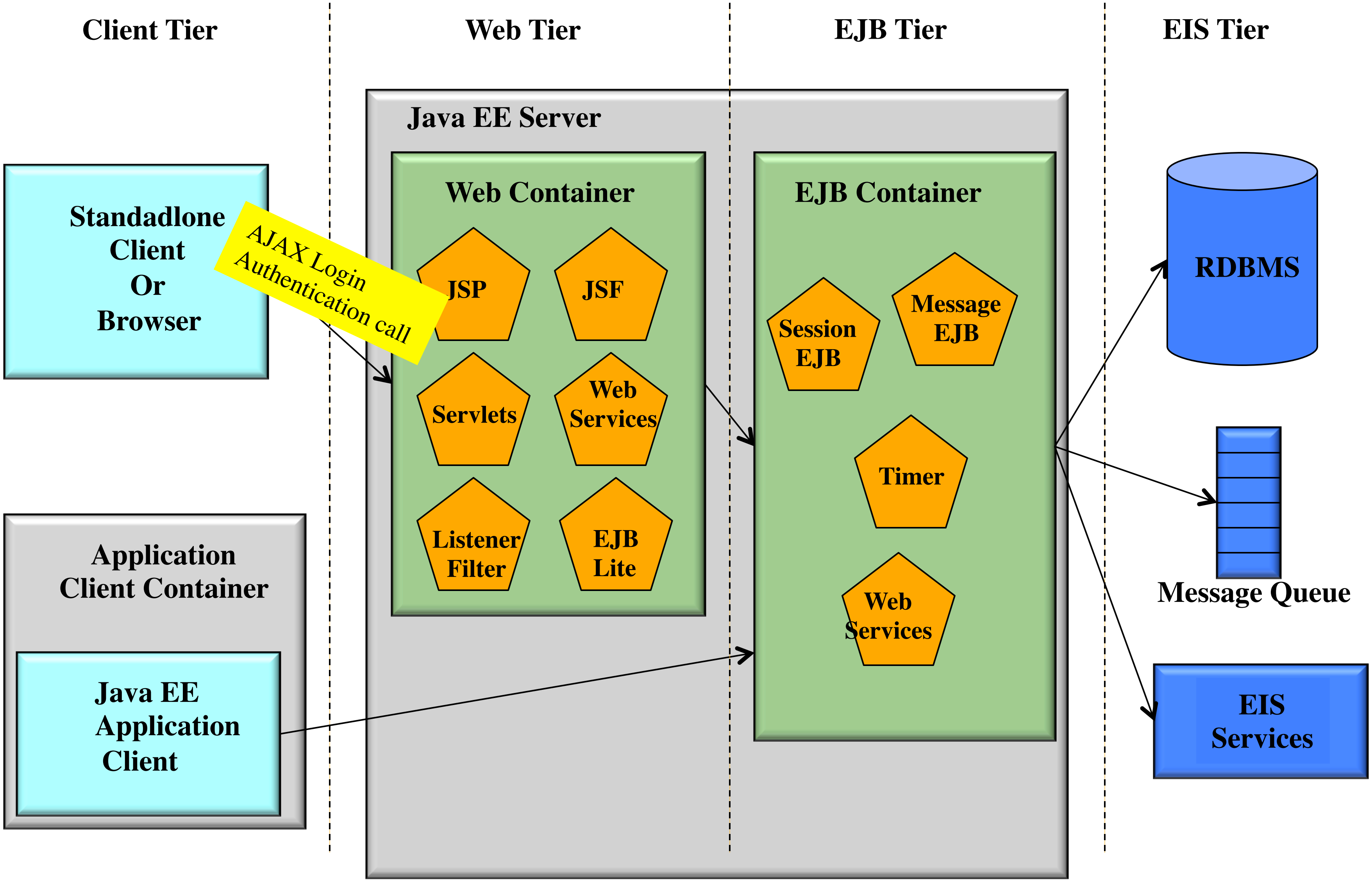
Web Server

1. HTTP Server retrieved client' certificate from SSL
2. HTTP Server transmits certificate information to WAS
3. WAS map's client's certificate to a principal
4. WAS establishes identity of the principal
5. If client is authorized, WAS allows access to the resource

/secure/showmyaccount.do

- Stronger Authentication Model
- Uses digital signature instead of password

Programmatic Authentication in Web Tier



Programmatic Authentication

- **`HttpServletRequest` interface has several methods to enable programmatic authentication**
  - `void login(String user, String password)`
    - Allows a program to collect username and password information as an alternative to form-based deployment
    - Aids in building in making REST API based clients authenticate
      - Helps in building custom UI forms such as AJAX API based login
  - `void authenticate()`
    - Allows an application to instigate authentication of the request caller by the container from within an unconstrained request context

Session Management in Web Tier

- **Define Idle Session timeout**

```
<session-config>
   <session-timeout>30</session-timeout> </
session-config>
```

```
session.setMaxInactiveInterval(long t)
```

- **Explicit logout to invalidate session when the user logs out**

```
if (logoutAction)
   request.logout() ;
```

Authorization in Web Tier

- **Proper Authorization**
  - Authorization refers to access control
  - Regulate access to functionality based on the role of the user
- Can be achieved
  - Declaratively
  - Programmatically
  - Combination of Declarative and Programmatic (Preferred)

"IT LOOKS LIKE EVERYONE WILL BE GETTING WHAT THEY WANT THIS YEAR...SOMEBODY POSTED MY CREDIT CARD NUMBER ON THE INTERNET!"

Declarative Authorization web.xml

```
<security-role> <!-- Can have Many -->
  <role-name>role-name from-vendor-specific-file</role-name>
</security-role>

<security-constraint> <!-- Can have Many -->
  <web-resource-collection>
    <web-resource-name>Descriptive name</web-resource-name>
    <url-pattern>url-pattern</url-pattern>
    <http-method>GET|POST|PUT…</http-method> <!-- multiple -->
    <http-method-omission>GET|POST|PUT…</http-method-omission>
  </web-resource-collection>
  <auth-constraint>
    <role-name>role-name from above</role-name> <!-- multiple -->
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee><!-- later --></transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

Declarative Authorization Example

```
<security-role>
  <role-name>HR</role-name>
</security-role>

<security-constraint>
  <web-resource-collection>
    <web-resource-name>payroll</web-resource-name>
    <url-pattern>/employee/payroll/*</url-pattern>
    <http-method>GET</http-method>
    <http-method>POST</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>HR</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee><!-- later --></transport-guarantee>
  </user-data-constraint>
</security-constraint>
```

Declarative Authorization using Annotations

- **@DeclareRoles (*list-of-roles*)**
  - Analogous to `<security-role>` in web.xml
  - Specified on a class (EJB/Servlet) to declare all the roles used in it
  - Typically used in conjunction with Programmatic authorization
- **@ServletSecurity**
  - Analogous to `<security-constraint>` in web.xml
  - Type annotation used to specify security constraints
- **@HttpConstraint**
  - Constraint applies to all HTTP methods
- **@HttpMethodConstraint**
  - Analogous to `<http-method>` in web.xml
  - Constraint applies to only a specific HTTP method
- **@RunAs (*role-name*)**
  - Used to specify the role to be used for propagated security

Declarative Authorization using Annotations Example

```
@WebServlet("/payroll")
@ServletSecurity(value=@HttpConstraint(transportGuarantee=TransportGuarentee.
CONFIDENTIAL, rolesAllowed={"MANAGER"}),
                 httpMethodConstraints={
                   @HttpMethodConstraint(value="GET", rolesAllowed={"HR"}),
                   @HttpMethodConstraint("TRACE"),
                   @HttpMethodConstraint(value="DELETE",
         emptyRoleSemantic=ServletSecurity.EmptyRoleSemantic.DENY),
                 })

public class MyServlet extends HttpServlet {

         /* Code goes here */

}
```

Programmatic Authorization in Web Tier

- **`HttpServletRequest` interface has several methods to aid in programmatic authorization**
  - `String getRemoteUser()`
    - Returns the login of the user making this request, if the user has been authenticated, null otherwise
  - `Principal getUserPrincipal()`
    - Returns java.secucrity.Principal object containing the name of the authenticated user
    - There are specialized versions of Principal like X509 Principal which have more information associated with the user
  - `boolean isUserInRole(String role)`
    - Returns a boolean indicating whether the authenticated user is included in the specified logical “role”

Programmatic Authorization in Web Tier Example

```
@WebServlet("/payroll")
@DeclareRoles("HR")
public class PayrollServlet extends HttpServlet {

    protected void processRequest(HttpServletRequest request,
                          HttpServletResponse response) throws
                          ServletException, IOException {
        ...

        if (request.isUserInRole("HR")) {
                 hireNewEmployee("Blah”);
        }
        ...
    }
}
```

Confidentiality and Non-Repudiation in Web Tier

- **Use HTTPS to secure communication**
  - Uses Secured Sockets as Transport Layer
  - Encrypts all data communication between end points
  - Use Client Certificates to achieve Non-Repudiation
- **<transport-guarantee> element in web.xml achieves confidentiality and non-repudiation**
  - Takes on values CONFIDENTIAL, INTEGRAL, NONE
  - CONFIDENTIAL and INTEGRAL mean the same - SSL

```
<user-data-constraint>
  <transport-guarantee>CONFIDENTIAL</transport-guarantee>
</user-data-constraint>
```

```
request.getScheme(); // Programmatically to check
                     // communication scheme (HTTP/HTTPS)
```

Auditing in Web Tier

- **Filters are great ways by which you can capture every request/response into audit log**

```
<filter-mapping>
  <filter-name>AuditFilter</filter-name>
  <filter-url>/*</filter-url>
</filter-mapping>
```

```
public class ValidationFilter implements javax.servlet.Filter {

    public void doFilter(ServletRequest req, ServletResponse res,
                                                FilterChain fc) {
        logRequest(req);
        fc.doFilter(req, res);
        logResponse(res);
    }

    private void logRequest(HttpServletRequest req) { ... }
    private void logResponse(HttpServletRequest req) { ... }

}
```

Quality of Service in Web Tier

- **Ensure a high Quality of Service to *ALL* users**
- **Minimizing Denial Of Service attacks**
- **Preventing hackers from injecting malicious code such as Cross Site Scripting**
  - Never trust the client!
  - Validate all input data to the application for correctness, data type, format, length, range etc
  - Don’t use blacklisting, use white listing
  - Escaping Input might be required
  - Encode all output response

Quality of Service in Web Tier

- **Minimizing Hackers knowing Internal Details**
  - Use server side comments

```
    <%-- JSP style comment --%>
```

  - Handle Global declarative Exceptions in web.xml

```
<error-page>
    <exception-type>FQEN</exception-type>
    <location>path/to/resource</location>
 <error-page>
```

  - Use HTTP error codes

```
<error-page>
    <error-code>HTTPErrorCodeNumber</error-code>
    <location>path/to/resource</location>
 <error-page>
```

JavaOne 2004

JavaOne™

Sun's 2004 Worldwide Java Developer Conference™

You Are Hacked☹

Ten Secrets to Securing Your J2EE Web Applications

Karthik Shyamsunder,
Principal Engineer

VeriSign® The Value of Trust™

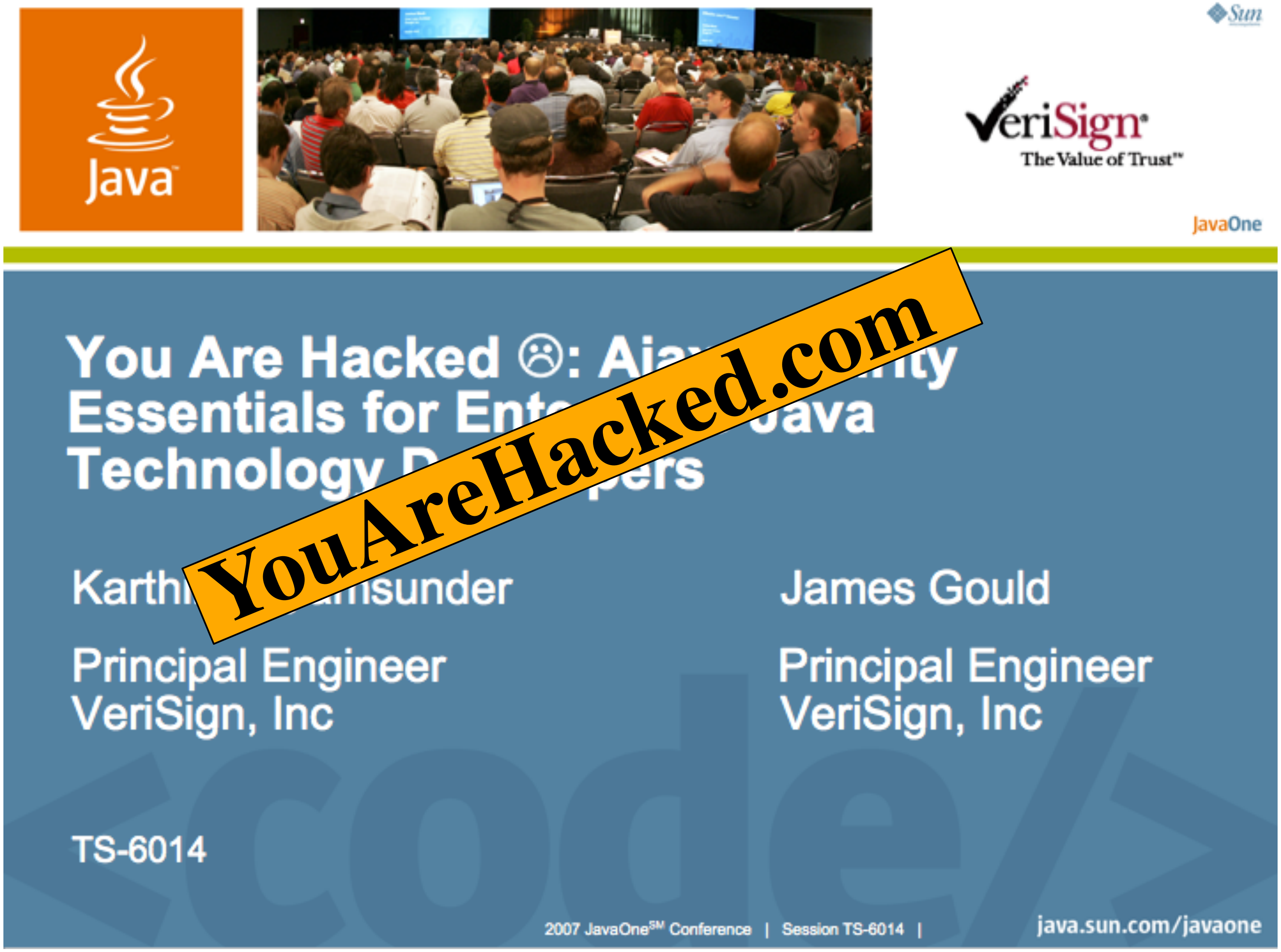
java.sun.com/javaone/sf

Java™

Sun microsystems

1 | 2004 JavaOne^{SM} Conference | Session TS-2137

JavaOne 2007



YouAreHacked.com

You Are Hacked ☹: Aja ity
Essentials for Ent Java
Technology D pers

Karthi nsunder
Principal Engineer
VeriSign, Inc

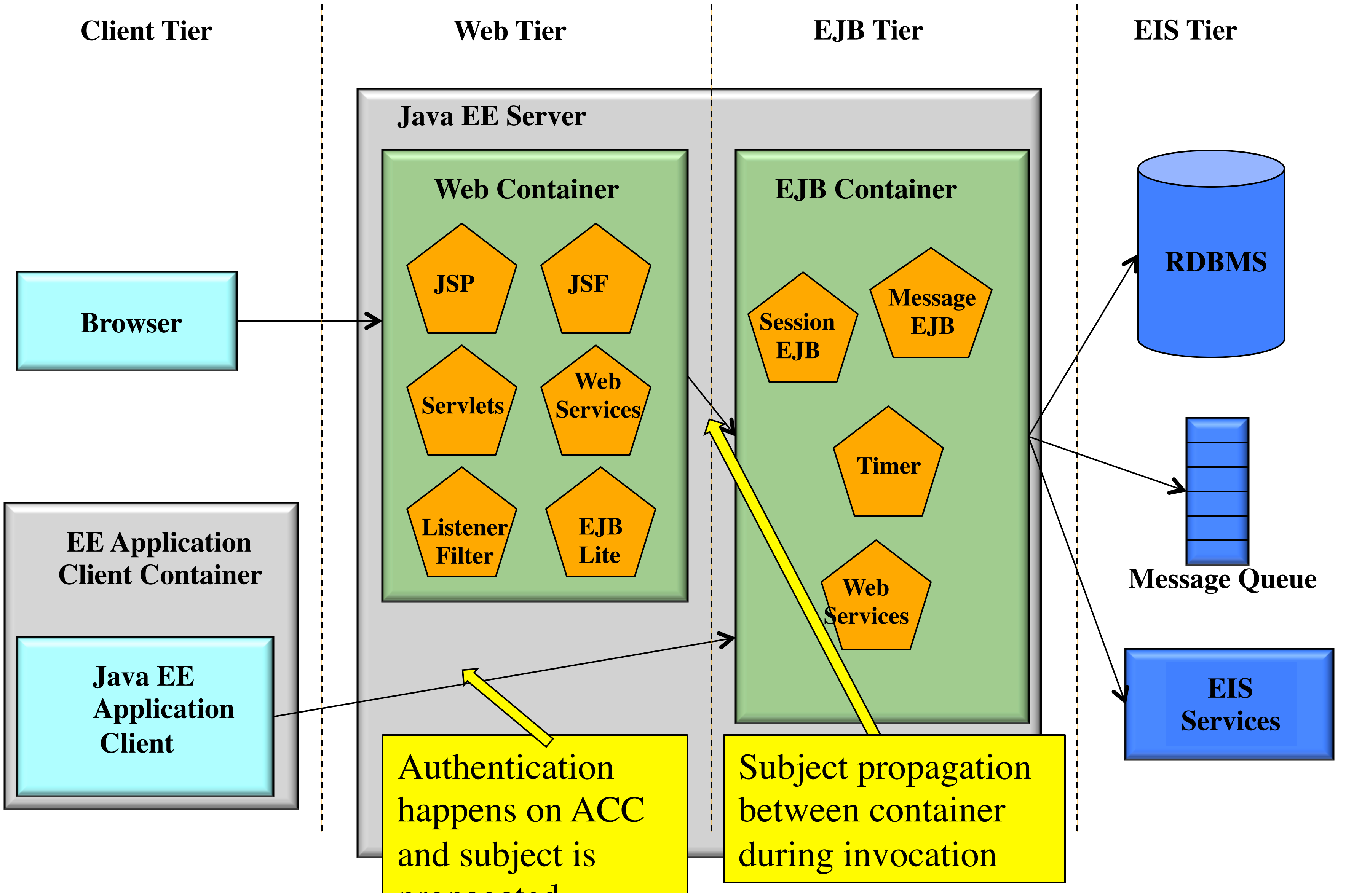
James Gould
Principal Engineer
VeriSign, Inc

TS-6014

Outline

- The Internet Threat Model
- Java EE Architecture Model
- Java EE Security Model
- Web Tier Security
- **EJB Tier Security**
- EIS Tier Security
- Java EE Security Challenges
- Summary
- Questions

Authentication for EJB Tier



Authentication happens on ACC and subject is propagated

Subject propagation between container during invocation

Programmatic Authentication in EJB Tier (Glassfish)

- **com.sun.appserv.security.ProgrammaticLogin**
  - Boolean login(String user, String password)
  - Boolean login(String user, String password, String realm, boolean errors)
    - Will set the SecurityContext in the name of the given user upon successful login
  - Boolean logout()
  - Boolean logout(boolean errors)
    - Will attempt to logout the user

Declarative Authorization ejb-jar.xml

```
<security-role> <!-- Can have Many -->
  <role-name>role-name</role-name>
</security-role>

<method-permission>
  <description>user-friendly description</description>
  <role-name>role-name</role-name> <!-- Can have Many -->
  <unchecked /> <!-- instead of role names -->
  <method> <!-- Can have Many -->
    <ejb-name>name of the EJB</ejb-name>
    <method-name>method-name or ‘*’</method-name>
      <method-params>
         <method-param>param type</method-param>
      </method-params>
  </method>
</method-permission>

<exclude-list>
 <description>user-friendly description</description>
 <method> <!-- same as above → </method>
</exclude-list>
```

Declarative Authorization ejb-jar.xml Example

```
<method-permission>
  <role-name>MANAGER</role-name>
  <method>
    <ejb-name>payrollService</ejb-name>
    <method-name>hireNewEmployee</method-name>
  </method>
</method-permission>

<method-permission>
  <role-name>HR</role-name>
  <method>
    <ejb-name>payrollService</ejb-name>
    <method-name>*</method-name>
  </method>
</method-permission>

<exclude-list>
  <method>
    <ejb-name>payrollService</ejb-name>
    <method-name>fireAllEmployees</method-name>
  </method>
</exclude-list>
```

More Security Related Deployment Descriptors on EJB Tier

- **<run-as>**
  - Declared within the <security-identity> in EJB declarations in ejb-jar.xml
  - It specifies the identity to use to execute the EJB methods
- **<use-caller-identity>**
  - Declared within the <security-identity> in EJB declarations in ejb-jar.xml
  - Specifies that the caller’s security identity needs to be used to execute the EJB methods

Declarative Authorization using Annotations

- **@RolesAllowed (*list-of-roles*)**
  - Can be applied to a Class or a Method
  - When applied to a method, it specifies roles that have access to it
  - When applied to a class, it specifies roles that have access to all the methods, unless the method is also annotated with this
- **@PermitAll**
  - Can be applied to a Class or a Method
  - Permits users with any role to access the method or all methods in the class
- **@DenyAll**
  - Can be applied to a Class or a Method
  - Denies all users access to a specific method or all methods in the class
- **@DeclareRoles (*list-of-roles*)**
- **@RunAs (*role-name*)**

Declarative Authorization using Annotaions Example

```
@RolesAllowed(“RestrictedUsers”)
public class EmployeeService{

    public long viewEmployeeInfo() { //... }

    @RolesAllowed(“Manager”)
    public void hireNew() { //... }

    @PermitAll
    public long provideFeedback() { //... }

    @DenyAll
    public long fireAllEmployees() { //... }

}
```

Programmatic Authorization in EJB Tier

- **javax.ejb.EJBContext**
  - java.security.Principal getCallerPrincipal()
    - Obtain the java.security.Principal that identifies the caller.
  - boolean isCallerInRole(String roleName)
    - Test if the caller has a given security role.

Programmatic Authorization in EJB Tier Example

```
@Stateless
@DeclareRoles("HR")
public class PayrollBean implements PayrollService {
     @Resource SessionContext ctx;

     public void updateEmployeeInfo(EmplInfo info) {
        // obtain the caller principal.
        callerPrincipal = ctx.getCallerPrincipal();

        // obtain the caller principal's name.
        callerKey = callerPrincipal.getName();
        oldInfo = ... Use caller name to retrieve caller info;

         // The salary field can be changed only by callers
         // who have the security role "payroll"
         if (info.salary != oldInfo.salary &&
             !ctx.isCallerInRole("payroll")) {
                 throw new SecurityException(...);
         }
         ...
     }
}
```

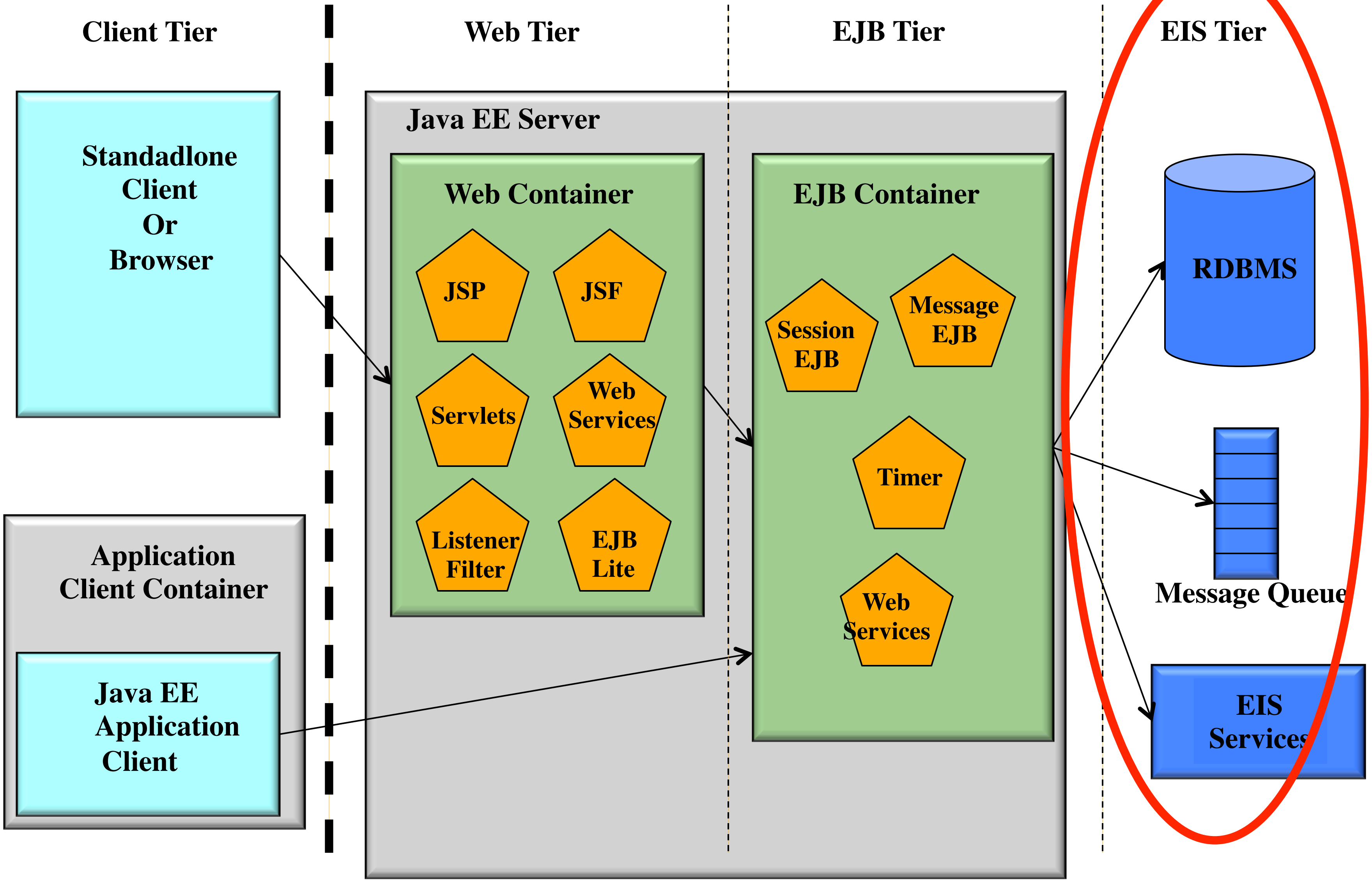
Auditing in EJB Tier

- **EJB Interceptors provide a nice way of auditing the EJB invocations**
  - @Interceptors
  - @AroundInvoke

Outline

- **The Internet Threat Model**
- Java EE Architecture Model
- Java EE Security Model
- Web Tier Security
- EJB Tier Security
- **EIS Tier Security**
- Java EE Security Challenges
- Summary
- Questions

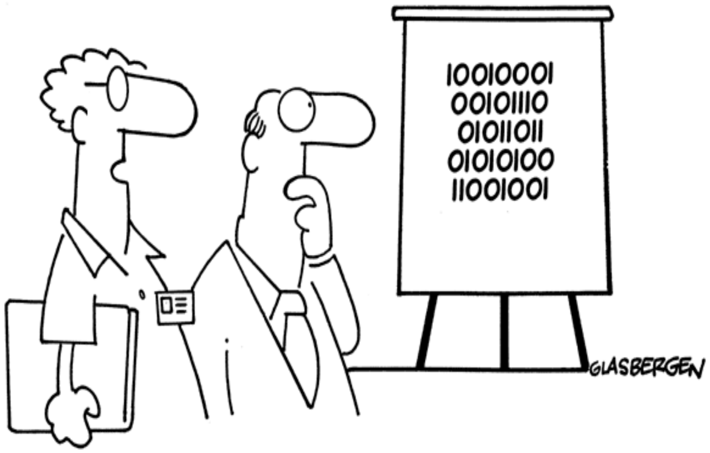
Protecting the EIS Tier



Stealing Stored and Transient Data

- **Many security compromises are because applications store and transmit data in in plain text**

Copyright 2003 by Randy Glasbergen.
www.glasbergen.com



"We've devised a new security encryption code. Each digit is printed upside down."

- **Encrypt all critical data**
  - Passwords, Cookies, Hidden fields
- **Do not invent your own encryption algorithm**
- **Chooses a good cryptography library**
  - Choose a library that has been exposed to public scrutiny
  - Make sure that there are no open vulnerabilities

Protecting Databases

- **Use EE server connection pool for DB connections, which encrypt DB credentials**
- **Prefer using JPA for CRUD operations**
  – If you have to write SQL, prefer using prepared statements

```
String usr = reg.getParameter(“userName”);
String passwd = reg.getParameter(“password”);
String sql = “SELECT * FROM USERS WHERE username = '” +
   usr + “' AND password = '” + pwd + “'”;
```

Consider username: `' OR 1=1--` password: `foobar1`

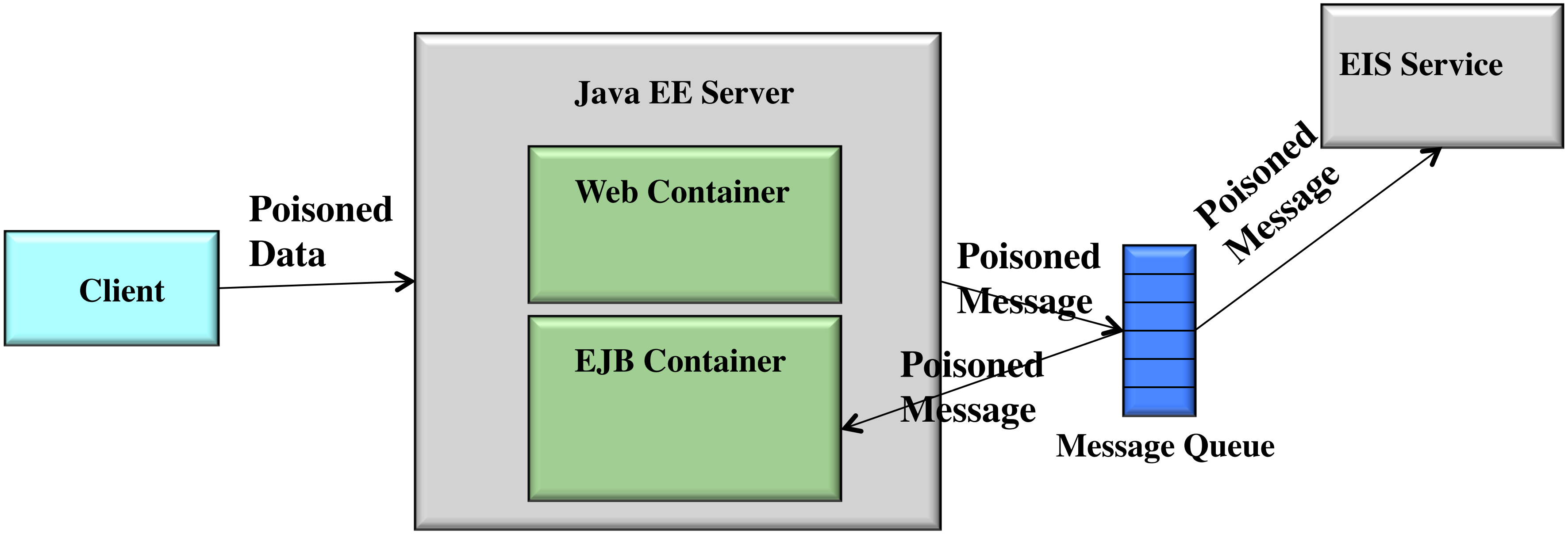
```
SELECT * FROM USERS WHERE
     username = '' OR 1=1-- AND password= 'foobar1'
```

```
    PreparedStatement ps = conn.prepareStatement(“SELECT
       * FROM USERS where username=? AND password=?”  );
    ps.setString(1, request.getParameter(“username”));
    ps.setString(2, request.getParameter(“password”));
```

Protecting Message Queues

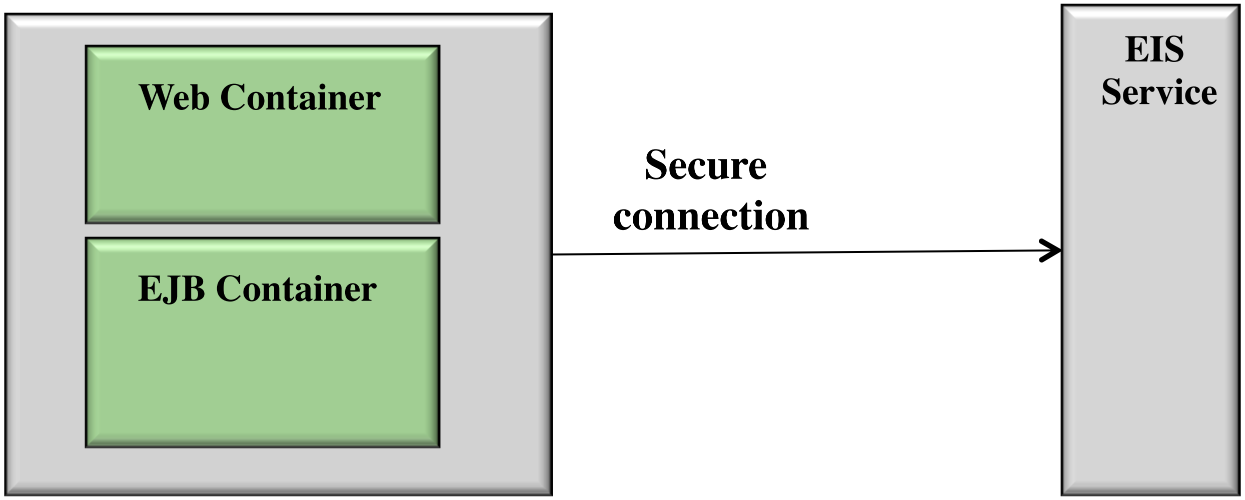
- Validate messages before you put into the message queue to prevent poisoned message

- Set limits on size of messages that can be put on the queue



Protecting Integrated Systems

- **Communicate with external EIS services over secure channel**
  - Use JSSE secure socket API for secure TCP communication
- **Use EJBs for enterprise integration with init and max-beans-in-pool configuration**
  - Sizing EJBs is done using app server specific EJB configuration



Outline

- **The Internet Threat Model**
- Java EE Architecture Model
- Java EE Security Model
- Web Tier Security
- EJB Tier Security
- EIS Tier Security
- **Java EE Security Challenges**
- Summary
- Questions

Java EE Security Challenges

- No out-of-box support for SAML
- No Remember Me
- No Error Messaging
- No URL for logout
- No Support for Regular Expressions for URLs
- Linking Groups/Principals to Roles is vendor specific
- Unable to define Roles on the fly

Summary

- **Security is a problem!**
  - With more and more enterprise applications moving to the cloud, the attack surface has only gone up
- **Java EE provides several security mechanisms**
  - Understand what they are and implement them in your application
- **Think Security First**
  - While building software, keep security in mind
  - Secure Platform -> Secure Design -> Secure Development -> Secure Testing -> Secure Deployment

Other JavaOne Security Sessions

- **New Security Features and Fundamentals: JDK 8 and Beyond**
  - Tuesday, Oct 2 @ 6:30PM
  - Brad Wetmore & Jeff Nisewanger
- **Security in the real world**
  - Wednesday, Oct 3 @ 1:00PM
  - Ryan Sciampacone
- **Front-to-Back Security for Mobile, HTML5, and Java EE Applications**
  - Thursday, Oct 4 @ 2:00PM
  - Marius Bogoevici & Jay Balunas

thank you

Q&A

You Are Hacked ☹ End-to-End Java EE Security

Karthik Shyamsunder, Principal Technologist
Phani Pattapu, Engineer