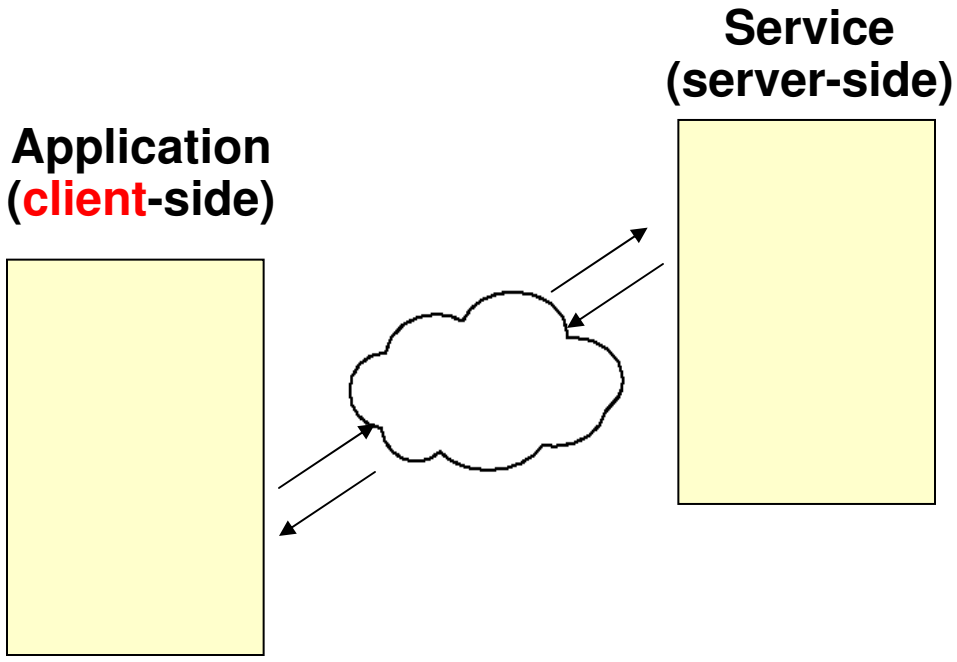
ASPECT SECURITY
application security specialists

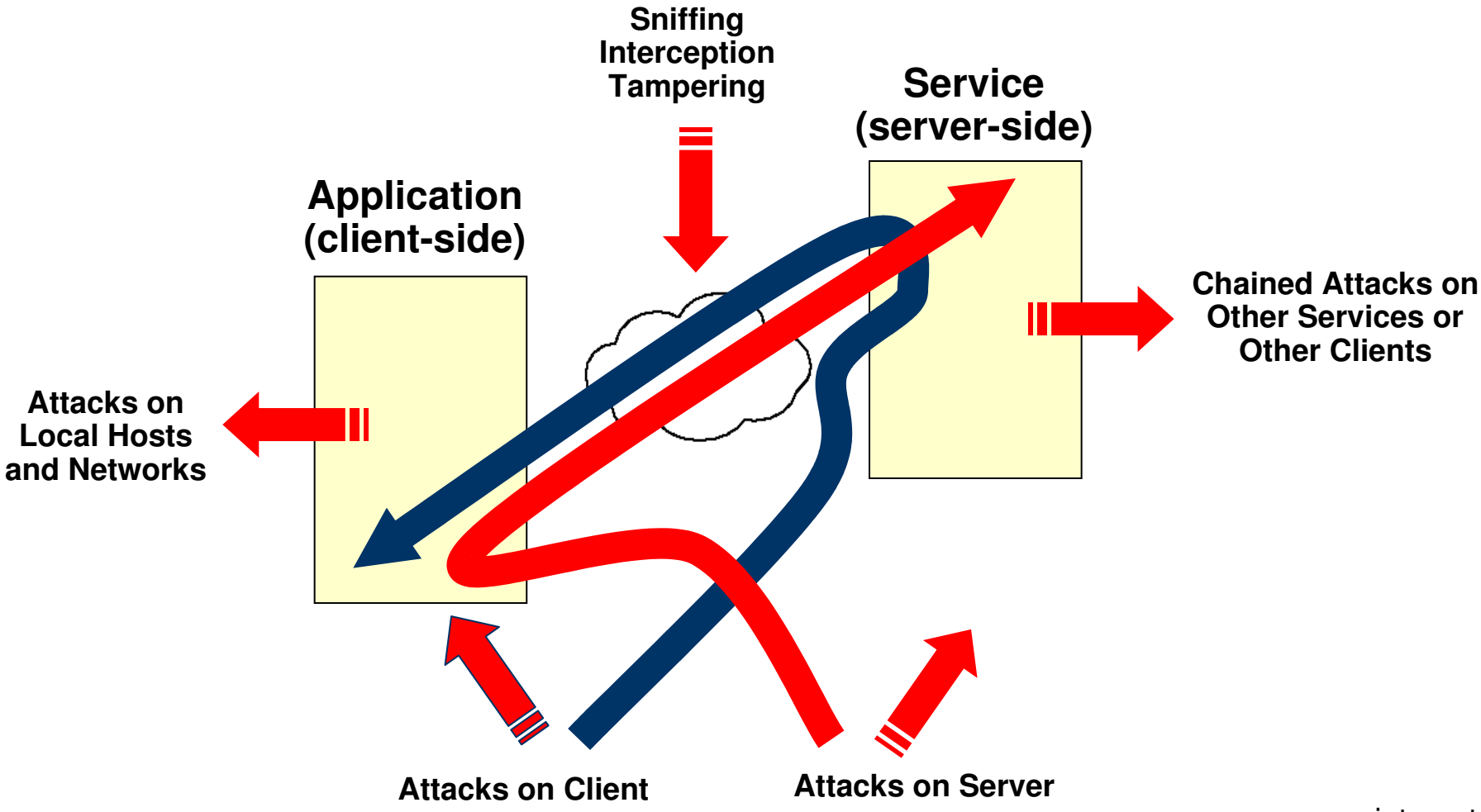
Securing J2EE Applications – Coding Patterns for Secure Connections to Services

Jeff Williams
Aspect Security CEO
jeff.williams@aspectsecurity.com
August 9, 2006

How Developers See Services



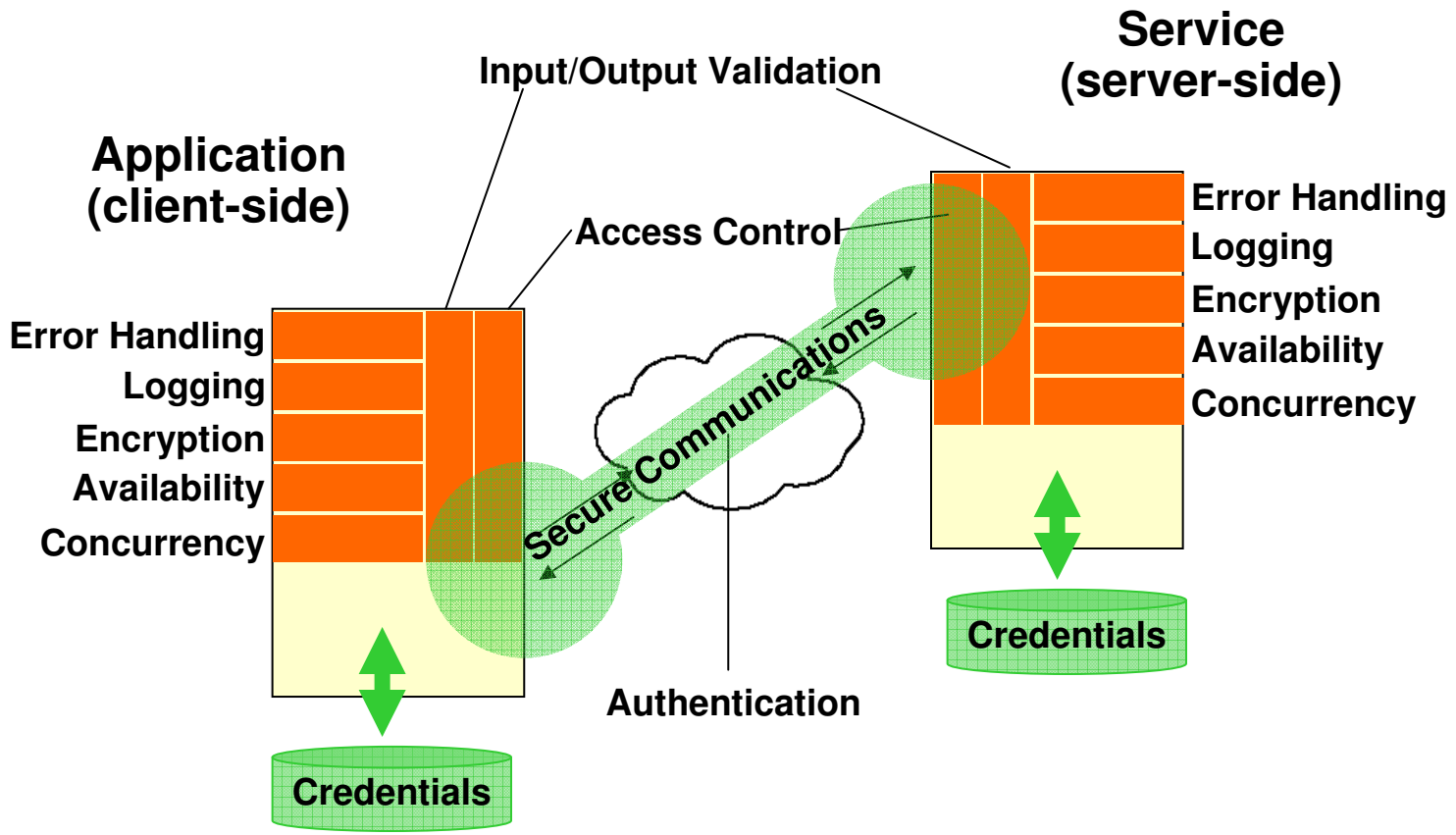
How Attackers See Services



Imagine the Future

- **Services mean trust relationships**
  - **Who are you?**
  - **What do you need?**
  - **What will you provide?**
  - **Will you protect my data?**
  - **Can I trust what you send me?**
  - **Will you attack me?**
  - **Can I trust your code?**
  - **Can I trust your other partners?**
  - **If something bad happens, who pays?**

Accessing Services Securely



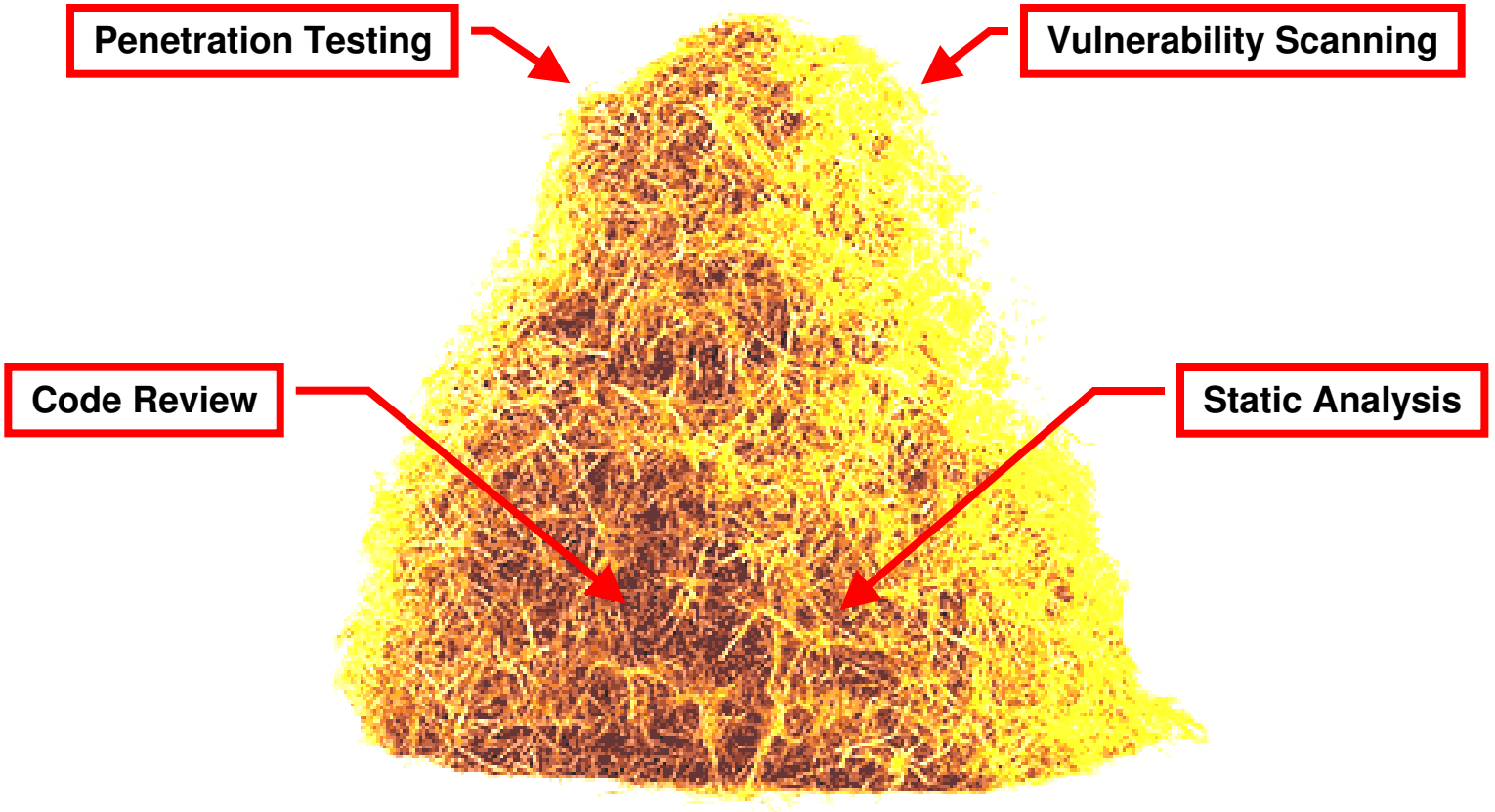
Note: the application is a "client" of the service, but might be a server application itself

What Does "Secure" Mean for a Service?

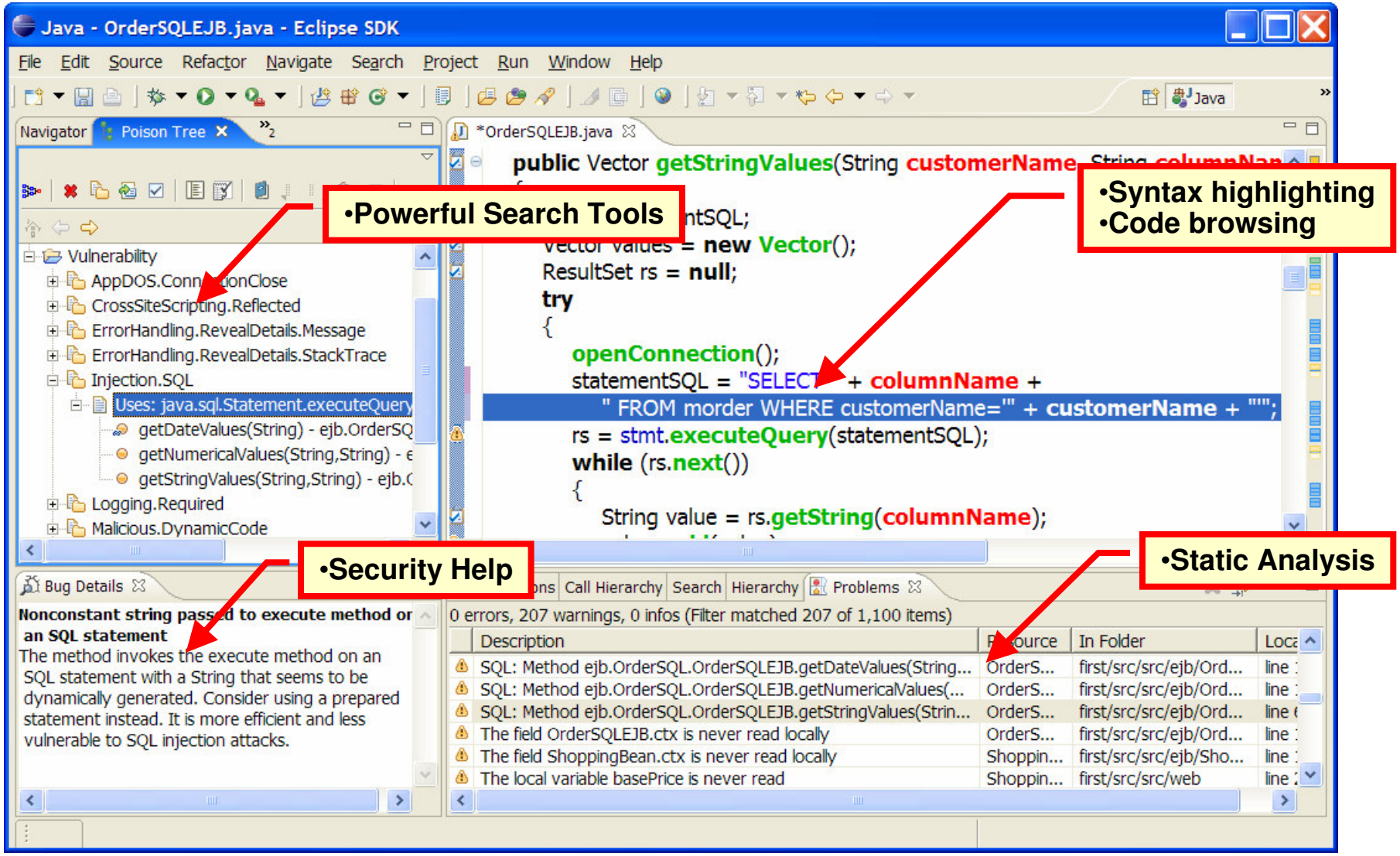
- Client-Side (App)
  - Secure Communications
  - Authentication and Sessions
  - Access Control
  - Validate & Encode Request
  - Validate & Encode Response
  - Error Handling
  - Logging & Intrusion Detection
  - Encryption
  - Availability
  - Concurrency
- Server-Side (Service)
  - Secure Communications
  - Authentication and Sessions
  - Access Control
  - Validate & Encode Request
  - Validate & Encode Response
  - Error Handling
  - Logging & Intrusion Detection
  - Encryption
  - Availability
  - Concurrency

Services are bidirectional attack vectors

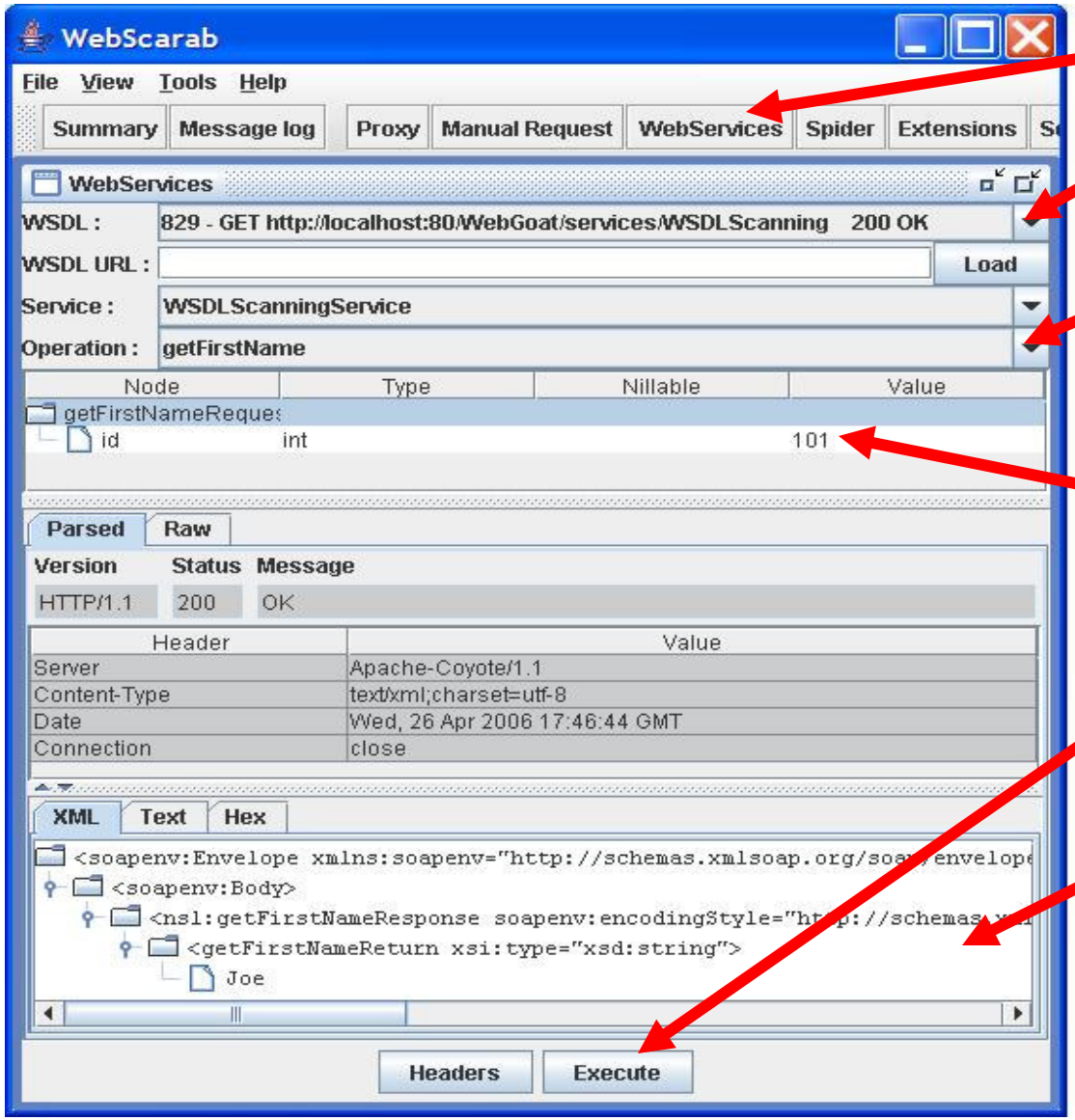
Techniques for Verifying Service Use



Using Eclipse for Code Review



Using WebScarab for Penetration Testing



Choose WebServices feature

Choose the WSDL

Choose the operation to execute

Add the parameter value

Execute the request

View the response

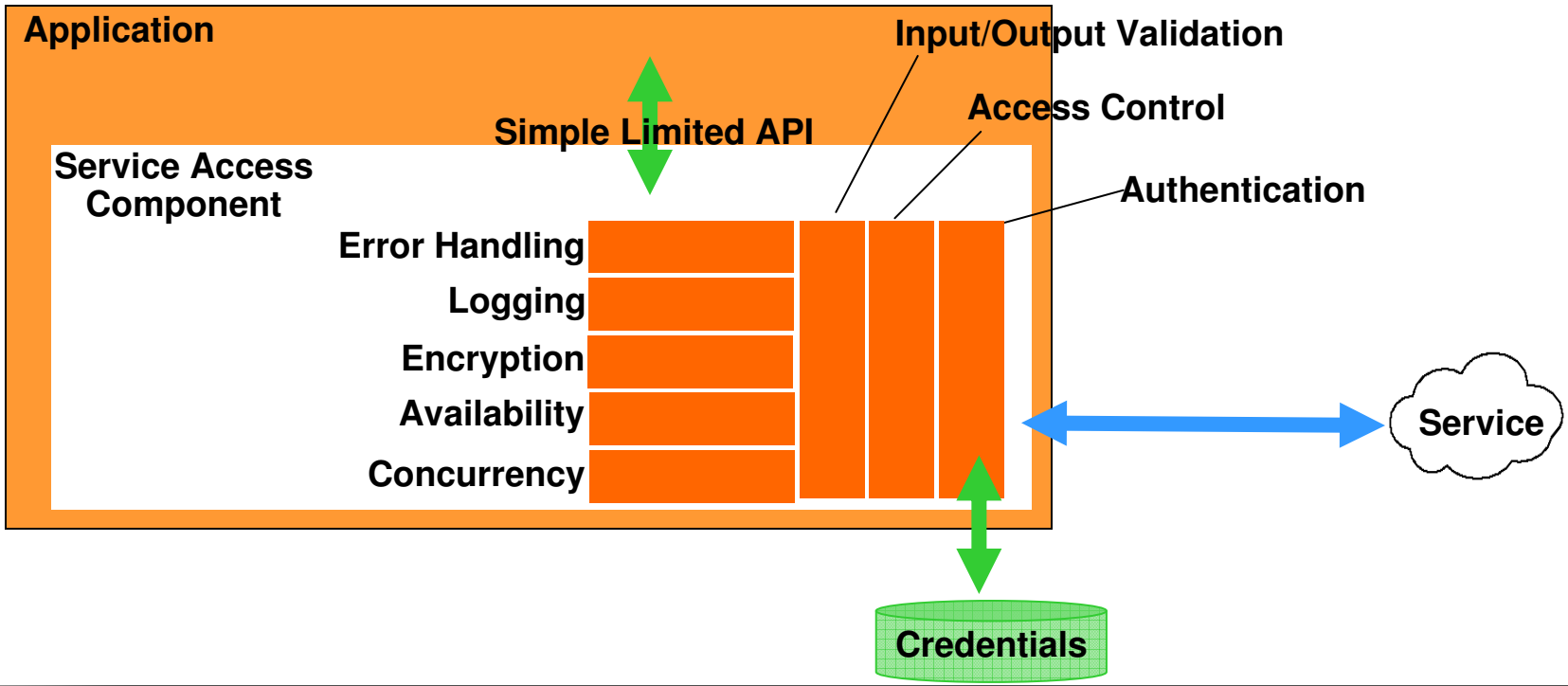
Use the WebScarab summary feature to view the HTTP traffic that WebScarab created.

Finding Services

- Search for them!
  - Start with the architecture diagram
  - Can be automated with tools
- Client Examples
  - Sockets – search for use of java.net.*
  - HTTP – search for use of URI, URL
  - Operating System – search for Runtime.exec()
  - Web Services – search for AXIS
- Server Examples
  - Database – search for use of JDBC
  - Servlet – search for use of ServletRequest
  - Custom services – search for use of libraries

Architecture for Accessing Services

- Create a "Service Access" Component
  - Isolates details of using the service
  - Provides a single implementation of security features
  - May be a façade on top of a more powerful library



Secure Service - Client Pattern

```
// pseudo-code template for invoking a service with security
...
   if ( !isAuthorized ) throw AuthorizationException
   if ( !isValidInput ) throw ValidationException
   try {
      credentials = encryptedProperties.getCredentials()
      service = open( credentials ) // SSL? Least privilege?
      encode( parameters )
      results = service.invoke( parameters )
      validate( results )
      log success
   } catch Exception e {
      log error
      throw proper exception
   } finally {
      close connection
   }
   encode( results )
   do something with results
...
```

- Secure Communications
- Authentication and Sessions
- Access Control
- Validate & Encode Request
- Validate & Encode Response
- Error Handling
- Logging & Intrusion Detection
- Encryption
- Availability
- Concurrency

Client Example: LDAP Using JNDI

```
// Set up environment for creating initial context
Hashtable env = new Hashtable(11);
env.put(Context.INITIAL_CONTEXT_FACTORY,
     "com.sun.jndi.ldap.LdapCtxFactory");
env.put(Context.PROVIDER_URL,
     "ldaps://localhost:636/o=jndi");

// Authenticate
env.put(Context.SECURITY_AUTHENTICATION, "simple");
env.put(Context.SECURITY_PRINCIPAL, "cn=user, ou=group, o=jndi");
env.put(Context.SECURITY_CREDENTIALS, "password");

DirContext ctx = null;
try {
    ctx = new InitialDirContext(env);
    String group = request.getParameter( "group" );
    System.out.println(ctx.lookup( "ou=" + group ));
} catch (NamingException e) {
    e.printStackTrace();
} finally {
    ctx.close();
}
```

Anything wrong here?

Client Example: TCP/IP Socket

```
try {
    Socket t = new Socket(args[0], 7);
    DataInputStream dis =
       new DataInputStream(t.getInputStream());
    PrintStream ps = new PrintStream(t.getOutputStream());
    ps.println("Hello");
    String str = dis.readLine();
    if (str.equals("Hello"))
      System.out.println("Alive!");
    else
      System.out.println("Dead or echo port not responding");
    t.close();
}
catch (IOException e) {
    e.printStackTrace();
}
```

Anything wrong here?

Client Example: Web Service

```
public class TestClient {
 public static void main(String [] args) {
  try {
    String endpoint = "https://localhost:8443/axis/Service.jws";
    System.setProperty("javax.net.ssl.trustStore",
       "/etc/security/.keystore");

    Service service = new Service();
    Call call = (Call)service.createCall();
    call.setTargetEndpointAddress( new java.net.URL(endpoint) );
    call.setOperationName( new QName("serviceName") );
    call.setUsername("user");
    call.setPassword("password");
    call.setTimeout( 20000 );  // timeout after 20 seconds

    String ret = (String) call.invoke( new Object[] { args[0] } );
    System.out.println("Response: " + response );
  } catch (Exception e) {
    System.err.println(e.toString());
  }
 }
```

Anything wrong here?

Client Example: E-mail

```
public void sendEmail( HttpServletRequest request )
{
     String to = request.getParameter( "to" );
     String from = request.getParameter( "from" );
     String text = request.getParameter( "msg" );
    Properties props = new Properties();
    props.setProperty("mail.transport.protocol", "smtp");
    props.setProperty("mail.host", "mymail.server.org");
    props.setProperty("mail.user", "emailuser");
    props.setProperty("mail.password", "password");
    Session mailSession = Session.getDefaultInstance(props, null);
    Transport transport = mailSession.getTransport();
    MimeMessage message = new MimeMessage(mailSession);
    message.setContent( text, "text/plain");
    message.addRecipient(Message.RecipientType.TO,
           new InternetAddress( to ));
     msg.setFrom(new InternetAddress( from ));
     msg.setSubject( "Check out this cool site" );
    transport.connect();
    transport.sendMessage(message,
           message.getRecipients(Message.RecipientType.TO));
    transport.close();
}
```

Anything wrong here?

Client Example: Google

```
StringBuffer results = new StringBuffer();
try {
   GoogleSearch gs = new GoogleSearch();
   gs.setKey("cd3H5SNQFHLjlSGI0vKhxFYUKKrx/M4g");
   gs.setQueryString(QUERY_FROM_PARAM);
   gs.setMaxResults(10);
   GoogleSearchResult sr = gs.doSearch();
   GoogleSearchResultElement[] results = sr.getResultElements();

   for (int index = 0; index < results.length; index++) {
        String title = results[index].getTitle();
        String url = results[index].getURL();
        String summary = results[index].getSnippet();
        results.append(title + ":" +
               summary + ":" + url + "\n" );
    }
} catch (Exception e) {
   e.printStackTrace();
}
return results;
```

Anything wrong here?

Secure Service - Server Pattern

```
// pseudo-code template for implementing a service with security
...
   hash = hash( password )
   if ( !isAuthenticated( username, hash ) ) throw
  AuthenticationException
   if ( !isAuthorized ) throw AuthorizationException
   if ( !isValidInput ) throw ValidationException
   try {
      encode( parameters )
      results = do something with parameters
      validate( results )
      log success
   } catch Exception e {
      log error
      throw proper exception
   } finally {
      close connection
   }
   encode( results )
   do something with results
...
```

- Secure Communications
- Authentication and Sessions
- Access Control
- Validate & Encode Request
- Validate & Encode Response
- Error Handling
- Logging & Intrusion Detection
- Encryption
- Availability
- Concurrency

Server Example - Web Service

```
package server;
import javax.jws.WebService;

@WebService
public class HelloImpl {
   public String sayHello(String name) {
     return "Hello, " + name + "!";
   }
}
```

From the tutorial…
"Take another look at the steps that we went through, and notice how little code we wrote to expose our original code as a Web service. These tools are only going to get better; at some point we will just think, "I want this as a Web service," and it will happen."

Anything wrong here?

Web Service Attack Names

- Coercive Parsing
  - Inject malicious content into XML
  - Solution: Validate before parsing
- XPath/XQuery Injection
  - Tamper with query changing meaning
  - Solution: Validate anything used in query
- Recursive Payload
  - Recursive references create DOS attack
  - Solution: Validate for recursion
- External Entity Attack
  - Use untrustworthy sources of data
  - Solution: Use well known URIs
- Schema Poisoning
  - Alter processing information
  - Solution: Use only trusted schemas
- XML Parameter Tampering
  - Submit malicious scripts or data
  - Solution: Validate request carefully
- Oversized Payload
  - Oversized files create DOS attack
  - Solution: Validate and enforce size limits
- SOAP Fault
  - Return full stack trace to attacker
  - Solution: Generate appropriate errors
- WSDL Scanning
  - Scan and invoke everything in the WSDL
  - Solution: Authenticate and authorize
- XML Denial of Service
  - Overwhelm a web service with requests
  - Solution: Authenticate and set quotas

Example XML Attacks

- **Example: Recursive Entity Reference**

```
<?xml version="1.0"?>
<!DOCTYPE a [
        <!ENTITY a "<element>&b;</element>">
        <!ENTITY b "&a;"> ]>
    <element>&a;</element>
```

- **Example: Code Injection**
  - **In PHP, an attacker can provide an XML file that uses single quotes to escape into the eval() call, and execute PHP code on the target server**
- **Example: External Entity Attack**
  - **Internet Explorer does not properly check to make sure that the XML data source is not redirected**
  - **See http://www.microsoft.com/technet/security/bulletin/MS05-025.mspx**

Web Services - Validation Paradox

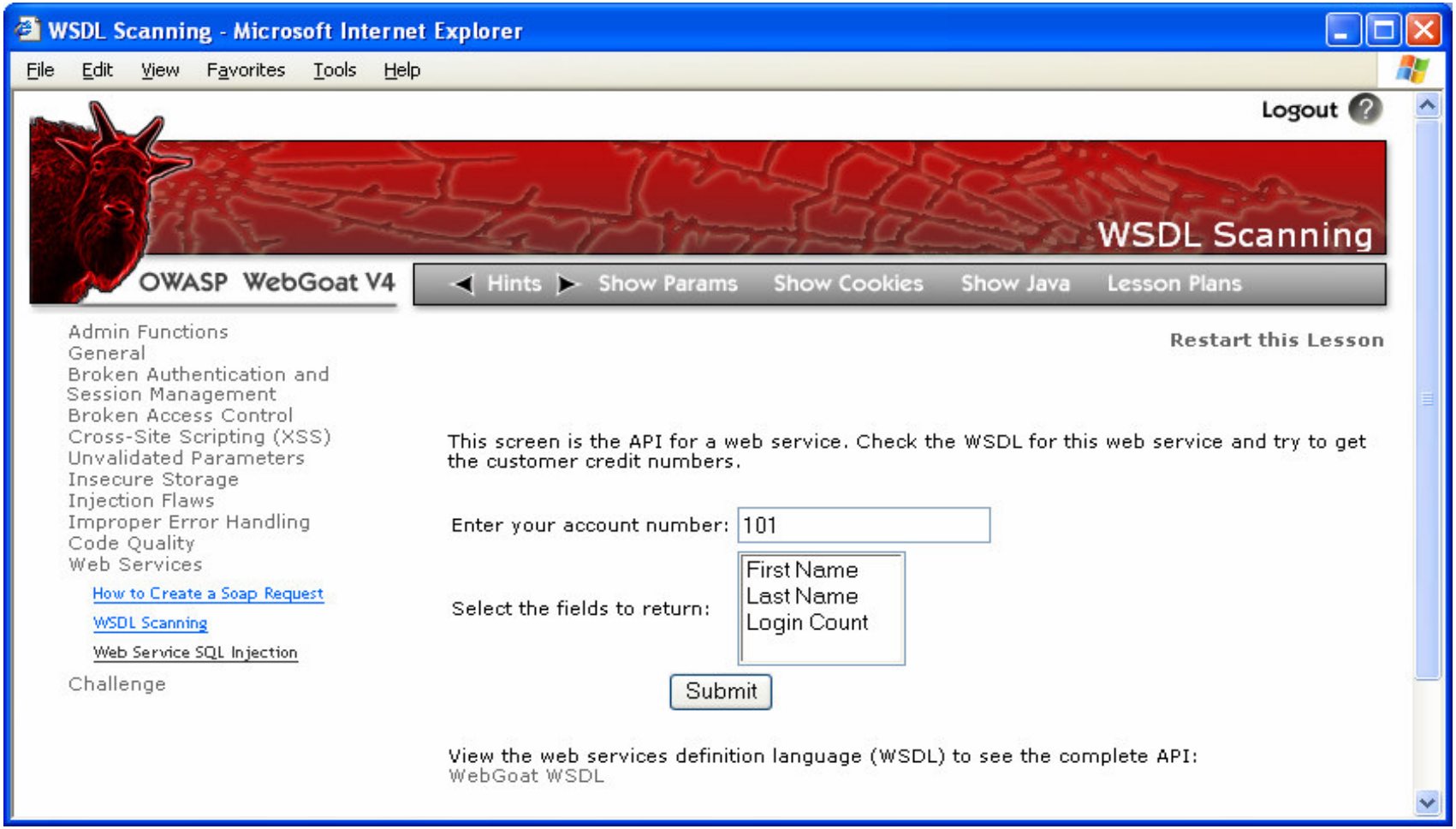
- You must parse before validating
  - Examine at each element and attribute
  - Validate using a set of validation rules or schema
- You must validate before parsing
  - Many XML attacks attempt to break the parser
  - Validate before parsing
- Solution
  - Ideal: Integrate security validation into parsers
  - Current: Do your own validation (size, recursion, attacks) before feeding documents into the parser

Web Services - SOAP Faults

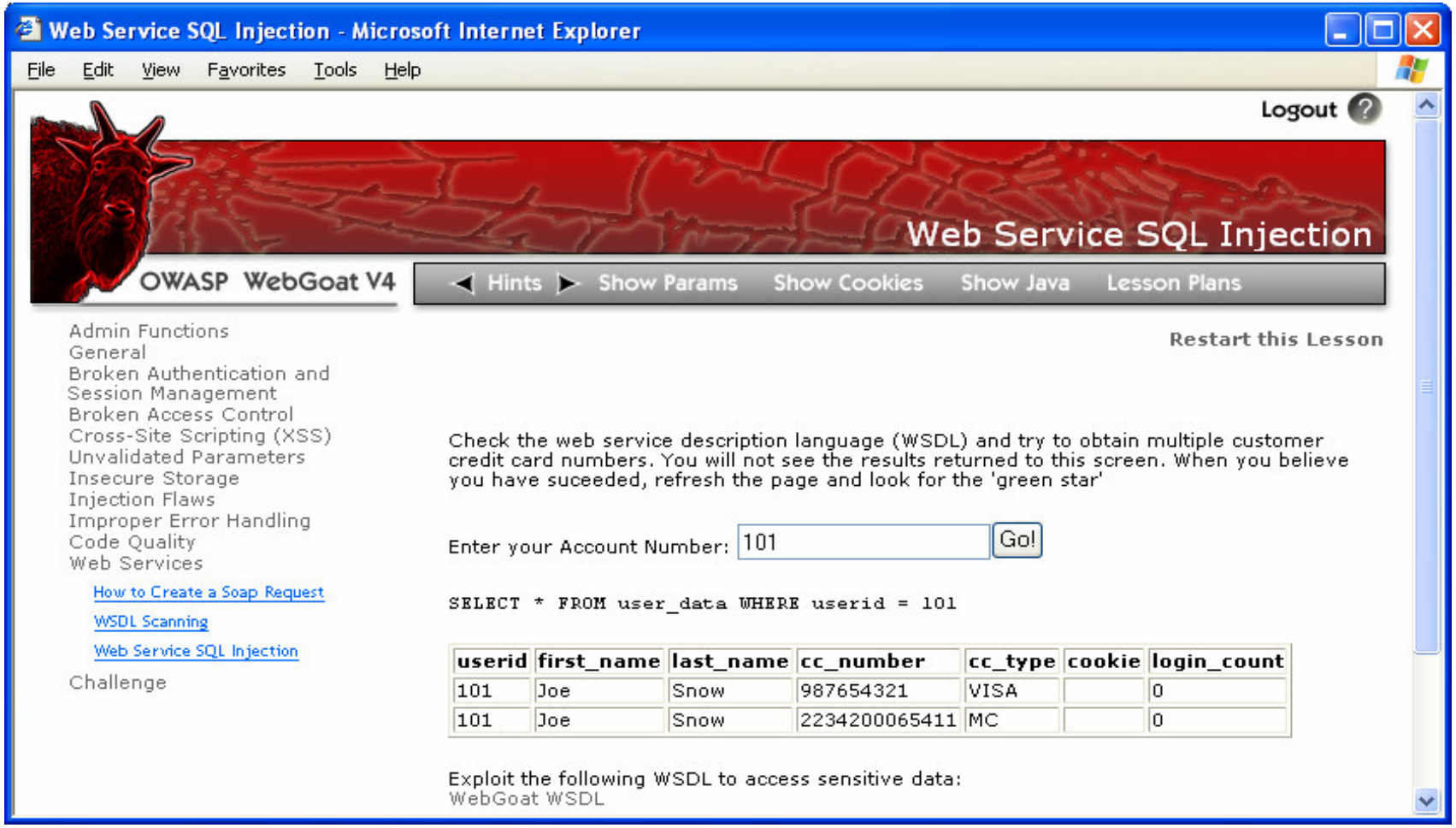
- Same issues as web application
  - Handle all errors
  - Don't expose internals
  - Don't provide other information useful to an attacker
- SOAP Fault
  - Simple XML based description of an error
  - WebSphere generates a Java exception and serializes into a SOAP fault

```
<soap:Envelope
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
       <soap:Fault>
           <faultcode>soap:Server</faultcode>
           <faultstring>…full stack trace…</faultstring>
       </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

WebGoat – WSDL Scanning



WebGoat – Web Service SQL Injection



Security in a Service Oriented World

- **Services will create massive interconnected trust "web"**
  - **Most services are security disasters**
  - **Far worse than web applications**
- **Securing services is possible**
  - **Takes some thought and planning**
- **Action plan – the time to address this is NOW**
  - **Before you have hundreds of insecure services to deal with**
  - **Find out whether this is really a problem in your organization**
  - **Start a secure services initiative**
    - Standards and guidelines
    - Tools and training
    - Process improvements

Q&A