Server-Side Web Development

JSP

Lecture #7 2007

Web Security

- Based on cryptography
- SSL / TLS current encryption standards
- HTTPS = HTTP through a SSL tunnel
  (no changes in JSP required)

Cryptography

- Mathematical tools for enabling trust
- Based on fundamental assumptions
  - algorithms are safe (there are no shortcuts)
  - parameter space searches for keys takes a long time
  - techniques used as intended
- Message: data
- Algorithm: the encryption method
- Key: encryption key, parameter to encryption algorithm
- Cipher text: the encrypted message

One-Way Encryption

- Messages are encrypted using secret keys
- Messages can not be decrypted
- Cipher texts are (to a high probability) uniquely mapped to message content
- Cipher texts are used instead of messages in situations where messages must be kept secret (e.g., passwords)
- Closely related to hashcodes and Message Authentication Codes (MACs)

Symmetric Encryption

- Commonly referred to as *private key encryption*
- Messages are encrypted and decrypted using the same key
- Anyone with access to the key can decrypt the message
- Fast
- Suffers from *the key distribution problem*

Asymmetric Encryption

- Commonly referred to as *public key encryption*
- Messages are encrypted using key pairs (public & private)
- One key used for encryption, the other for decryption
- Public key distributed as much as possible
- Private key kept secret
- Versatile and more secure than symmetric algorithms
- Slow

Asymmetric Encryption

- Encrypt message using public key - encryption
- Encrypt message using private key - signatures
- Messages can be both encrypted and signed
- As long as the keys can be trusted
  - messages can be kept secret (only receiver can decrypt)
  - senders and receivers can be authenticated
  - message content can be trusted

Certificates

- Certificate = signed tuple of public key & identity
- Certificates can be self-signed or signed by others
- Self-signed certificates can be used for encryption (but suffer from *the key distribution problem*)
- Certificates signed by trusted parties can be used for encryption, authentication and message integrity checks

Public Key Infrastructures (PKI)

- Virtual infrastructures consisting of clients, servers and Certificate Authorities (CA)
- CAs are trusted third parties which provide signed certificates (i.e., signs public keys)
- CA certificates are distributed in browsers and similar tools (trusted and considered known by all)
- Since CA public keys are known, (signed) certificates can be validated offline (without connecting to the CA)
- Secure connections are established between parties using certificates and encryption algorithms
- Network traffic *tunneled* through encrypted channels

Secure Socket Layer (SSL)

- A protocol for establishing secure connections using certificates and cryptography algorithms
- Transport Level Security (TLS) = SSL v3.0 (almost)
- Clients use server certificate to authenticate server
- Servers use client certificate to authenticate client (optional)
- Once identities have been established, encryption keys are exchanged and symmetric encryption algorithms are used
- SSL clients uses *keystores* to manage certificates and keys

Keystores

- An encrypted database used to store keys and certificates
- Usually stored in a single file called `.keystore`
- Applications must provide database decryption key (username & password) to access keystore content
- Keystores only containing public keys and certificates are commonly referred to as *truststores*
- Keystores can be shared between SSL applications (usually only done for truststores)

HTTPS

- Not an actual protocol
- HTTPS = HTTP through a SSL/TLS tunnel
- The server needs to be provided with a certificate
- If the server is to authenticate clients, the clients need (CA signed) certificates as well
- HTTPS Web servers usually references keystores via configuration (providing filename, username, password)
- Default port 443 (HTTP default port is 80)
- JSP can check if a page was requested via HTTPS using `request.isSecure()`
- HTTPS / SSL is considered safe (today)

Summary

- Cryptography is the tool for web security
- No changes in JSP required to use HTTPS (web server reconfiguration may be required)
- Web server needs a certificate
- JSP can require clients to use HTTPS

Tag Libraries

- Introduced in JSP 1.1
- Allows users to create their own JSP tags
- Commonly referred to as *custom tags*
- Emphasizes role-based development methodology (creator of tag different from user of tag)
- Integrated in JSP (can alter JSP body response stream)
- Used to encapsulate complex logic and reuse code

Custom Tags

- A Java class implementing the *Tag* interface (boilerplate implementation classes available)
- Defines tag name, tag attributes & tag body interpretation
- Specify a tag description in a XML-based *descriptor* file
- Included in JSP using the `taglib` directive
- Tags may control JSP processing

Tag Interface

- Package `javax.servlet.jsp.tagext`
- Implement directly or extend `TagSupport` / `BodyTagSupport`

```
public interface Tag extends Tag
{
  Tag getParent ();
  void setParent (Tag t);
  void setPageContext (PageContext pc);
  void release ();
  int doEndTag ();
  int doStartTag ();
}
```

IterationTag Interface

- Package `javax.servlet.jsp.tagext`
- Implement directly or extend `TagSupport` / `BodyTagSupport`
- Used to iterate body evaluation (repeats until `doAfterBody()` returns `SKIP_BODY`)

```
public interface IterationTag extends Tag
{
  int doAfterBody ();
}
```

BodyTag Interface

- Package `javax.servlet.jsp.tagext`
- Implement directly or extend `BodyTagSupport`
- Used to gain access to body content

```
public interface BodyTag extends IterationTag
{
  int doInitBody ();
  int setBodyContent (BodyContent b);
}
```

Tag Library Descriptor

- XML-based configuration file
- Provides a mapping from tag names to tag Java classes
- Contains tag library information and tag descriptions
- Tag descriptions direct how the tag is utilized
  (by the JSP engine, include tag body etc)
- Required, one per tag library

Descriptor Content

- `name` - tag alias for use in JSP
  (coupled to tag library namespace)
- `tagclass` - fully qualified implementation class name
- `attribute` - tag attributes (optional)
  (used as tag parameters, values delivered as Strings)
  - name - attribute name
  - required - attribute required to process tag flag
  - rtexprvalue - attribute value from JSP expression flag
- `info` - descriptive information about tag (optional)
- `body-content` - tag body processing directives (optional)
  - EMPTY - no tag body
  - JSP - body contains JSP
  - TAGDEPENDENT - tag processes body itself

Tag Lifecycle

1. Development
   - coding a tag library & writing a descriptor
   - developing JSP pages that uses the tag library
2. Translation - compile time
   - tags and JSP translated to servlets (tag calls inserted)
3. Evaluation
   - request time
   - tag is loaded and methods are called

Tag Development

- Can be done in a separate environment
  (only requires access to the J2EE environment)
- Tag development like any other Java development
- Tags are exported in a JAR file
- Tag library descriptor included in JAR file
  (in the META-INF directory)

Tag Translation

Compile time

1. A call to `doStartTag()` is inserted in Servlet
2. Tag body is translated (JSP inserted in Servlet)
3. A call to `doEndTag()` is inserted in Servlet

Tag Evaluation

Request time

1. `setPageContext()` called, page context provided
2. `setParent()` called, page hierarchy established (used for nested tags)
3. `set`*`Attribute`*`()` is called for attributes
4. `doStartTag()` called, return value directs processing
5. Tag body processed (if so instructed by `doStartTag()`)
6. `doEndTag()` is invoked, return value directs processing
7. `release()` is called to release tag resources (so that tag objects can be reused by thread pools)

Tag Method Return Values

- `doStartTag()`
  - `EVAL_BODY_INCLUDE` - process tag body
  - `EVAL_BODY_BUFFERED` - tag processes body
  - `SKIP_BODY` - do not process tag body
- `doAfterBody()` (`IterationTag`)
  - `EVAL_BODY_AGAIN` - repeat tag body evaluation
  - `SKIP_BODY` - do not repeat tag body evaluation
- `doEndTag()`
  - `EVAL_PAGE` - process rest of JSP
  - `SKIP_PAGE` - do not process rest of JSP

Tag Example

```
public class SimpleHelloWorldTag extends TagSupport
{
  //------------------------------------------------------------
  public int doEndTag ()
    throws JspException
  {
    try
    {
      JspWriter out = pageContext.getOut();
      out.print("(Simple) Hello world!");
    }
    catch (IOException e)
    {
      throw new JspException(e.getMessage());
    }

    return Tag.EVAL_PAGE;
  }
}
```

Tag Library Descriptor Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE taglib
  PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
  "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>

  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>lecture07</short-name>
  <uri>taglibs/lecture07</uri>
  <description>
    lecture07 taglib
  </description>

  <tag>
    <name>SimpleHelloWorld</name>
    <tag-class>lecture07.tags.SimpleHelloWorldTag</tag-class>
    <body-content>empty</body-content>
  </tag>

</taglib>
```

Tag JSP Example

```
<%@ taglib uri="/WEB-INF/lecture07-taglib.tld" prefix="lecture07" %>

<html>
<body>

...

<lecture07:SimpleHelloWorld/>

...

</body>
<html>
```

web.xml

- Web application configuration file
- Specific to Tomcat
- Maps relative tag descriptor URIs to local filenames
- Optional (specify descriptor path in taglib directive)

```
<taglib>
  <taglib-uri>lecture07-taglib.tld</taglib-uri>
  <taglib-location>/WEB-INF/lecture07-taglib.tld</taglib-location>
</taglib>
```

Summary

- Tab libraries are used to extend the JSP tag set
- Very powerful way to reuse Java code in JSP
- Custom tags can be used as any JSP tag
- Tag behavior determined by tag developer
- Custom tags well suited to hide large logic segments
- Custom tags are never visible to web clients

Next Time

- Web Services