Securing your Apache Tomcat installation

Tim Funk

November 2009

Who am I?

- Tomcat committer for over 7 years
- Day job: programmer at Armstrong World Industries.

Why?

```
function search() {
  var q = document.search.q.value.split(/\w+/);
  var where = '';
  for (i=0;i<q.length;i++) {
      where+=" OR sku like '%"+q[i]+"%'"
      where+=" OR name like '%"+q[i]+"%'"
      where+=" OR descr like '%"+q[i]+"%'"
  }
  document.search.extraCriteria = where;
}
--------------------------------------------
String where = "type=1 and (" +
                  request.getParameter("extraCriteria") + ")";
String query = "select * from items where " + where;
ResultSet rs = statement.executeQuery(query);
```

Background

- Based on 6.0.x
  - Much applies to 5.5.x and 4.1.x
- Security:
  - Confidentiality
  - Integrity
  - Availability
- Usability
- Need to balance for your environment

Threats seen in the wild

- Very few Tomcat specific
- Malicious webapp seen from July 2008
  - fex*.war
  - Some variants download and execute code
  - Some variants provide (effectively) a shell prompt
- Infection via insecure manager app
- What security flaw have been reported? http://tomcat.apache.org/security.html

Configuration
Other components

- Tomcat configuration should not be your only line of defence
- OS
  - Limit Tomcat privileges
  - Limit access to Tomcat's files
  - Minimize impact of a successful attack
- Firewall
  - In and out

Configuration Installation

- If you don't need it, remove it
- Web applications
  - docs
  - examples
  - host-manager
  - manager (probably)
  - ROOT

Configuration
Security manager

- Web applications run in a 'sandbox'
  - Prevents System.exit
  - Can limit what files can be accessed
- Some Tomcat testing is performed with this enabled
- Configured in catalina.policy

Configuration
Security manager

- Do you trust your web applications?
- Restricts actions of malicious web applications
- Default policy very likely to break your application
- Don't use it without testing

Configuration server.xml

- Configuration is reasonably secure by default
- Discuss options to make it more/less secure
- Values shown are defaults unless otherwise noted

Configuration server.xml

- Comments
  - Delete components you aren't using
  - Minimize other comments
- <Server ... />
  - port=”-1” (non-default) disables it
  - shutdown should use strong password
- <Listener .../>
  - Native (APR) on Solaris is not stable
- <Executor .../>
  - Nothing to see here

Configuration server.xml

- <GlobalNamingResources .../>
  - Should not be using UserDatabase as primary realm
  - Only used for shared resources
  - Changes will require a Tomcat restart
- <Service .../>
  - Nothing to see here
- <Engine .../>
  - Nothing to see here

Configuration server.xml

- <Connector .../>
  - Do you need HTTP and AJP enabled?
  - address=”...” (defaults to all)
  - allowTrace=”false”
  - maxPostSize=”2097152” only parameters
  - maxSavePostSize=”4096”
  - xpoweredBy=”false”
  - server=”Server: Apache-Coyote/1.1”
    - server =”Microsoft-IIS/5.0”
  - Many more parameters may be set for tuning purposes

Configuration server.xml

- AJP specific
  - request.secret=”...” should be strong if used
  - request.shutdownEnabled=”false” (default)
  - request.useSecret=”false” (default)
  - tomcatAuthentication=”true” (default)

Configuration server.xml

- <Host .../>
  - autoDeploy=”false”
    - default is true
  - deployOnStartup=”true”
  - If both false, only contexts defined in server.xml would be deployed
  - deployXML=”true”
    - Set to false to ignore META-INF/context.xml
  - errorReportValveClass – Override error pages for all webapps in a host. May be helpful when a webapp doesn’t handle all errors.
  - http://tomcat.apache.org/tomcat-6.0-doc/config/host.html

Configuration server.xml / context.xml

- <Context .../>
  - crossContext=”false”
  - privileged=”false”
  - allowLinking=”false”
    - Don't change this on a case-insensitive OS
    - This includes Windows
  - useHttpOnly (default false)
  - caseSensitive (default true)

Configuration
Nested components

- <Valve .../>
  - Always configure an access log valve
  - Remove/archive old log files
  - Typically, one per host
  - Can be configured at Engine, Host, or Context level
  - e.g.:

```
<Valve className="org.apache.catalina.valves.AccessLogValve"
        directory="logs"  prefix="localhost_access_log."
        suffix=".txt" pattern="common" resolveHosts="false" />
```

Configuration
Nested components

- <Valve .../>
  - Use a Remote Address Filter where possible
  - The allow/deny attributes are regular expressions
  - allow is better than deny
  - e.g.:

```
<Valve className="org.apache.catalina.valves.RemoteAddrValve"
        allow=”127\.0\.0\.1” />
```

Configuration
Nested components

- <Realm .../>
  - Don't use these in production:
    - Memory Realm
    - UserDatabase Realm
    - JDBC Realm
  - That leaves:
    - DataSource Realm
    - JNDI Realm
    - JAAS Realm (not widely used)

Configuration
Nested Components

- <Realm .../>
  - There is no account lock-out implemented
    - Brute-force attacks will work (eventually)
  - May be protected by JNDI service if userPassword is not set
  - Suspected infection method for fex* malicious web application

Configuration Nested Components

- <Realm .../>
  - New for 6.0.19
  - Combined Realm
    - Realm nesting
    - Use multiple Realms for authentication
  - LockOut Realm
    - Wraps standard realms
    - Locks accounts after multiple failed logins

Configuration
Nested Components

- <Realm .../>
  - LockOut Realm example:

```
<Realm className="org.apache.catalina.realm.LockOutRealm"
          failureCount="5" lockOutTime="300" cacheSize="1000"
          cacheRemovalWarningTime="3600">
    <Realm className="org.apache.catalina.realm.DataSourceRealm"
              dataSourceName=... />
</Realm>
```

Configuration
Nested Components

- <Loader .../>,
- <Resources .../>,
- <Manager .../>
  - entropy=”this.toString()”
    - This has been shown to be predictable
    - APR provides real random value
  - randomClass=”java.security.SecureRandom”
  - sessionIdLength=”16”

Configuration System Properties

- org.apache.catalina.connector. RECYCLE_FACADES=”false”
- org.apache.catalina.connector. CoyoteAdapter.ALLOW_BACKSLASH=”false”
- org.apache.tomcat.util.buf. Udecoder.ALLOW_ENCODED_SLASH=”false”
- org.apache.coyote. USE_CUSTOM_STATUS_MSG_IN_HEADER=”false”
- STRICT_SERVLET_COMPLIANCE=”true”

Configuration
JNDI Resources

- server.xml or context.xml
- Why is the password in plain text?
  - Tomcat needs the plain text password to connect to the external resource
  - Encrypting the password means Tomcat would need a decryption key – back to the original problem

Configuration JNDI Resources

- What are the risks of a plain text password?
  - Remote information disclosure
    - Is the resource remotely accessible?
  - Local information disclosure
    - If an attacker has console access you probably have bigger issues
    - They can replace any Tomcat code and read the plain text password from memory

Configuration JNDI Resources

- Solutions to the plain text password issue:
  - Enter the password at Tomcat start
    - Needs custom code
    - Password still in memory
    - Tomcat restart requires manual input
  - Encode the password
    - Needs custom code
    - Encoding is not encryption
    - May prevent some accidental disclosures

Configuration web.xml

- Default servlet
  - readonly=”true”
  - listings=”false”
- Invoker servlet
  - Evil
  - bypass security constraints
  - may expose servlets packaged in 3rd party jars
  - will be removed from 7.x onwards

Configuration manager web.xml

- Manager webapp is just a webapp
  - You can edit web.xml to allow for different roles per action
  - You can disable parts of it
  - Consider adding <transport-guarantee>CONFIDENTIAL</transport-guarantee>

Configuration manager web.xml

Example: Allow reload, but disable install

```
REMOVE
 <servlet-mapping>
    <servlet-name>Manager</servlet-name>
      <url-pattern>/install</url-pattern>
  </servlet-mapping>
```

Configuration manager web.xml

Example: Allow reload, but for role installer

```
<security-constraint>
  <web-resource-collection>
    <url-pattern>/install</url-pattern>
  </web-resource-collection>
  <auth-constraint>
     <role-name>installer</role-name>
  </auth-constraint>
 </security-constraint>
```

Configuration manager webapp

- Extra lockdown considerations
  - Consider RemoteAddrValve to only allow localhost
  - Then ssh tunnel

```
<Valve className="org.apache.catalina.valves.RemoteAddrValve"
           allow="127\.0\.0\.1" />

ssh  -L 80:localhost:80 myserver.net
```

Monitoring

- Most monitoring tools also provide management functionality
  - Is the benefit of monitoring worth the risk of exposing the management tools?
- Options
  - manager
  - LambdaProbe
  - JMX
  - commercial

Monitoring

- Always configure an access log
- Always use a Remote Address Filter
- Configure strong passwords
  - Adm!n, @dmin, @adm!n are not strong passwords

Policy / Procedure / Process Review logs

- Which logs?
  - Access logs
  - Application logs
  - Tomcat logs
- What do you do if you find an attempted attack?
- What do you do if you find a successful attack?

Policy / Procedure / Process

- Do you monitor Tomcat vulnerability announcements?
- What do you do if one affects you?
- Impact will be:
  - configuration change
  - patch
  - upgrade
- Likely to require a Tomcat restart

Policy / Procedure / Process
Avoiding downtime

- Restart = downtime
- Using a cluster minimises downtime
  - one httpd instance
  - two Tomcat instances
  - mod_proxy_http / mod_jk / mod_proxy_balancer
  - sticky sessions (optional)
  - session replication (optional)

Your webapp

- File Locations
  - When possible – put everything in WEB-INF
    - Templates
    - Configuration
    - Anything not directly requested by the browser
  - Precompile your JSP’s and do not deploy them
    - When
    - Anything not directly requested by the browser

Your webapp

- Don’t trust the incoming data
  - Be aware of what your parameters should be
  - Be also of what they should not be
- Have bad incoming data?
  - Try to detect early
  - Return a 404, 400, or 301 as needed
  - mod_security can handle obvious incoming data violations before the request is made to Tomcat
- See OWASP or {search engine} for the top # web vulnerabilities

Your webapp

- Configure Error pages
  - CONFIGURE THEM! Otherwise, you will disclose that you are
    - running Tomcat
    - possibly the version
    - and worst of all – you may let users see stack traces or JSP code snippets

```
<error-page>
  <error-code>500</error-code>
  <location>/WEB-INF/500.jsp</location>
</error-page>
```

Your webapp

- Error pages / messages
  - Do not expose stack traces or the message from a Throwable
    - `No out.println(e.getMessage())`
  - HTML escape any messaging on the page which was user provided
    - `Invalid Zip Code <c:out value='${param.zip}' />`
  - Log your errors.
    - Make sure someone examines them and fixes them, or logs them correctly.
    - Logs full of noise are logs which are not examined – clean them up and log correctly.

References / Resources

http://www.eu.apachecon.com/presentation/materials/78/2009-03-26-SecuringApacheTomcat.pdf

http://code.google.com/p/securetomcat/

http://www.owasp.org/index.php/Securing_tomcat

http://wiki.apache.org/tomcat/FAQ/Security

Questions