BuildingSecureWebApplications

Building a Robust and Secure Web Application With Velocity

draft document – June 11, 2003

Any time you build a web application, it your responsibility as a web developer to ensure that the application does what it is supposed to do, fails gracefully in case of an error, keeps users from gaining access to data they are not supposed to view, and prevents malicious users from interfering with the operation of the application.

While a detailed discussion of how to build a robust web application is an overly complex topic, this short paper touches on several issues that are common in a Velocity-based web application. The paper is written from the perspective of a Velocity developer, who interacts with a group of HTML template designers and a larger pool of end users. Readers are encouraged to send comments, other issues, and relevant design tips to the Velocity users list or directly to the author, Will Glass-Husain.

For general information on building a Velocity web application, consult the discussion of VelocityServlet in the Developer's Guide. You should also review the more recently created VelocityViewServlet in the velocity-tools subproject.

How Velocity Helps the Developer Create a Robust App

In general, Velocity is a web templating tool that is easy for HTML designers to learn and hard for them to misuse. This is due to several factors:

- **Velocity Template Language (VTL) is simple.** With just a few key directives plus a set of application-defined references, there is little to learn (and little to mess up).
- **No non-presentation information is stored in the page file.** This is a contrast to Java Server Page (JSP) files where a page header is often required. If a non-technical web designer removes the header, the page may be disabled. No such header or other technical information is needed in a Velocity template.
- **No Java code is included in a web page.** This removes a common source of errors and confusing design, and makes it difficult for a malicious page writer to call unwanted Java methods.
- **A method exception does not block the creation of a page.** With MethodExceptionEventHandler, a method exception can be caught and logged.
- **An invalid reference does not block the creation of a page.** An invalid reference in a directive will usually be silently ignored. An invalid reference in the page will simply print out the reference verbatim (e.g. $foo).

For these reasons, most Velocity developers will find that pages are rendered in a straight-forward manner with few quirks or problems.

Velocity-Specific Issues Regarding Robustness and Security

A Velocity web application does have several features that you should be aware of when considering security and robustness issues.

- **Velocity is a templating tool, not a framework.** It does not address any issues of authentication, access control, session state, or data persistence.
- **VTL method calls are actually Java method calls.** This means that a poorly designed velocity application can allow template designers to alter the system state, execute direct SQL queries or even instantiate arbitrary Java classes. Potential security consequences of this are discussed in more detail below.
- **VTL references have Java types.** Although this is not visible to the template writer, each reference is a java object with a particular type. If $apple is an integer "1", $orange is a string "1" and $banana is a double "1.0", none of these objects are == according to VTL. This can be confusing to the typical non-technical HTML template designer. (Actually, as of Velocity 1.5, if objects are not of the same type, a comparison of their string values is done. So, $apple and $orange are now == in VTL.)

Best Practices In Building A Secure, Robust Velocity Web Application

Below is a list of best practices that can help you build a robust Velocity web application. They include:

- Review all context references for unwanted methods.
- Encode HTML special characters to avoid cross-scripting vulnerabilities.
- Use an up-to-date and properly configured app server.
- Configure Velocity for production use.

Review all context references for unwanted methods.

References placed in the context by the developer generally have one of two primary purposes.

To provide dynamic information for display in the page (e.g. the name of the current user)
To provide a tool that helps control or reformat the information (e.g. a tool to format numbers)

It's tempting to take existing objects or beans and put them in the context. There is an important caution you should be aware of when doing this. Templates may call any public method from an object that is available in the page context. This means that you should take care to only provide methods that are safe for the template designer to access.

To hide unwanted methods, the developer needs to create a wrapper object. An important note-- it is not enough to subclass the object or to implement an interface that hides these methods. The reasons are:

- Using an interface has no effect as the template designer can call any public method from the class that implements the interface.
- Subclassing is not helpful as the template designer can call a method from a superclass with the VTL $reference.super().badmethod().

Some particular things to be concerned about:

- Do not include objects with any methods that can change application state. This breaks the Model/View/Controller paradigm and can be very difficult to debug.
- Avoid object/relational database mapping objects with methods that can execute SQL queries. Special note to users of the Jakarta projects Torque and Turbine: Generated Torque objects contain a method getPeer that provides access to an instance of org.apache.Torque.util.BasePeer. This class contains a number of methods that allow arbitrary SQL queries to be executed.
- Never put File or similar objects into the context. Always display filesystem information through wrapper objects.

Encode HTML special characters to avoid cross-scripting vulnerabilities.

Any time a web application displays text that was previously input by a user, there is a risk of including inappropriate text. In particular, such text might include HTML tags with JavaScript that will cause the page to behave differently than the author intended. This is a particular problem when the page is viewed by a second user, as the included text might include code to pop open windows, grab cookie information, or even intercept data entered into forms. As this a common web application design issue, a web search will turn up a number of articles with more information on the potential risks. For example, MSDN has a useful note.

The solution is to always escape the HTML special characters before displaying them on the screen. At a minimum, do the following substitutions:

original text	replacement text
<	<
>	>
"	"
&	&

For better readibility, also substitute these:

original text	replacement text
carriage return	< BR >
two spaces	one space followed by

There are four possible ways to do the substitution.

1. You can escape the text when processing the user input, and store it in encoded form. This is the easiest for the template designer, but may cause problems if you need to use the unescaped text in other places.
2. You can create a tool to escape the text when displayed. (Example 1, below)
3. You can create a wrapper object for any text in the context that automatically escapes the string. (Example 2).
4. Daniel Dekany has suggested that Velocity should include a new directive #escape (not yet implemented) which would automatically escape all text within the block. (Example 3).

Example 1: Velocity tool to escape the text.

```
$HTMLText.setText($textFromUser).Escaped
$HTMLText.setText($textFromUser).EscapedMultiLine
```

The first line would display the raw text $textFromUser with all HTML characters escaped. The second line would translate any carriage returns into
 statements. (Note: you will not want to do this in all cases. For example, you should include the carriage returns as entered in the textarea field in a form.)

Example 2: Using a text wrapper object.

```
$textFromUser.Escaped
$textFromUser.EscapedMultiLine
$textFromUser.PlainText
```

In this example, $textFromUser is defined in the context as a wrapper object with properties Escaped, EscapedMultiLine, and PlainText. The first property displays the text with the HTML codes escaped. The second also encodes carriage returns. The final property returns the text exactly as entered. If none of the three properties are given, the default text displayed should be Escaped.

Example 3: Using the #escape directive (unimplemented).

```
#escape("html")
  ... Tons of HTML here...
  ... interpolations will be implicitly escaped
#end
```

Use an up-to-date and properly configured app server.

As with any web application, make sure your Velocity-based app is running on an application server using the latest updates or service packs. Read up on security-related configuration settings and keep track of published vulnerabilities. Several issues that the author has encountered include:

- Configure your application server to allow a sufficient maximum heap size for the Java Virtual Machine (JVM) using the -Xmx java command line option. Without this setting, your application is likely to throw an OutOfMemoryError.
- When using a web server and Apache Tomcat, you must often explicitly disallow user access to WEB-INF, META-INF, and other system directories.
- Apache Tomcat versions prior to 4.1.12 could be tricked into using the DefaultServlet to display the source code for Velocity pages, Java Server Pages, or any template in the web tree.
- It's good practice to configure a Java Security Manager to restrict access to files (outside of the web tree and template paths) and dangerous methods such as System.exit() and getClassLoader. This is technically involved and is not fully supported by the current implementation of Velocity (1.3.1). (Most notably, some Velocity classes require access to the ClassLoader, while others should be explicitly restricted. If you're not familiar with these issues, this thread on the Velocity users list might be helpful to get started.

These are only general guidelines-- be aware of issues for your specific system.

Configure Velocity for production use.

A web application installed on a production server is typically configured very differently than while under development. Be sure to take advantage of Velocity's [developer-guide.html#Velocity Configuration Keys and Values extensive configuration options] to create a robust and professional deployment.

Some suggestions:

- Create an EventCartridge and Event Handlers to catch method exceptions. Log the exception, but allow the page to continue processing.
- Turn on caching of Velocity pages. In addition to speeding up the rendering process, this avoids a "memory leak" caused in certain circumstances by excessive page introspection.
- Create custom error pages that match the look and feel of your application. Display this error page after catching ParseErrorException and ResourceNotFoundException (thrown by VelocityServlet.getTemplate). Include an user friendly error message (passed to the page as a Velocity reference) and log the technical details for the developer / sysadmin.

Working with Untrusted HTML Template Designers

(note: some specific code references below are out of date. see list of enhancements for version 1.5 WGH - 10/7/2005).

In many Velocity applications, a small group of people (or a single person) work together on developing a web application. In such a case the focus of the developer is primarily on creating a user-friendly and secure application for the end-user. The developer provides the template designers with a set of simple technical guidelines for page design, most notably a list of available VTL references tools, and sets up CVS or FTP access to the web tree. In this common scenario, both the developers and the template writers have responsibility for ensuring the application works smoothly and securely.

In other Velocity applications, a larger pool of template designers create the templates, perhaps from outside the developer's organization. Often, the template designers do not have direct access to the CVS tree or the web file system. Instead, they might upload their templates to the web application through a web administration interface. In this case, the template writers should be regarded as "untrusted". Special care must be taken to ensure the integrity of the web application, regardless of what template is uploaded to the system.

While it's not entirely clear how many Velocity web applications fall into this category, the author has corresponded with half a dozen developers who have created this type of application. The author himself runs a Velocity-based web site in which hundreds of HTML template designers have their own account with the ability to upload templates.

Some notes on developing a web-application with "untrusted" template designers:

- As discussed above, only provide safe references in the context. Methods should not be able to change the app state, execute SQL queries or access the File system.
- Review the potential use of #include and #parse. In the author's application, templates and private user data for all accounts were originally stored in parallel directories in a web tree all in the same resource path. This meant that any template user could use #include to display template or data from another user. Serge Knystautas has suggested one possible solution to this dilemma which is to create a custom resource loader that loads templates based on the current user. Another solution is to use a patch to Velocity developed by the author which allows the developer to control the actual template returned by #include and #parse by using an event handler. (Typically, this would involve restricting each account to only include pages in that account). (update: new event handler included in source code tree, to be released in version 1.5 - WGH)

In addition, the comments above about configuring a security manager become critical in this type of application. Developers should be aware that template designers have the ability to call getClassLoader() returning a ClassLoader which could be used to instantiate any class and call any method in the default configuration. The author has proposed a patch to Velocity that would restrict this dangerous capability. (update: patch accepted and scheduled for version 1.6 - WGH)

Clearly the safest path is to restrict template design to a small group of trusted template writers. However, Velocity remains a useful tool in the case where a larger pool of users can upload templates. In such a case you must be even more careful to consider issues around system integrity and security.

Acknowledgements

The author would like to thank various members of the velocity-users mailing list for providing comments on these and related issues, particularly those involved in this discussion thread.