# Towards Security by Construction for Web 2.0 Applications

Benjamin Livshits and Úlfar Erlingsson
Microsoft Research

While security experts routinely bemoan the current state of the art in software security, from the standpoint of the application developer, application security requirements present yet another hurdle to overcome. Given the pressure for extra functionality, "lesser" concerns such as performance and security often do not get the time they deserve. While it is common to blame this on developer education, a big part of the problem is that it is extremely easy to write unsecure code.

## 1 Introduction

By way of illustration, consider an application that prompts the user for her name and sends a greeting back to the browser. The following example illustrates how one can accomplish this task in a Java/J2EE application:

```
ServletResponseStream out = resp.getOutputStream();
out.println("<p>Hello, " + username + ".</p>");
```

However, the apparent simplicity of this example is deceptive: assuming `userName` is supplied as application input, this piece of code is vulnerable to cross-site scripting [2]. This is because executable JavaScript can be embedded into `userName`. When the request is processed within the Web application, this JavaScript will be passed to the client's browser for execution. In summary, the most natural way to achieve the task of printing the user's name is broken: *the default is unsafe*. To make this secure, the developer has to apply input sanitization: he needs to exclude the myriad different ways to pass JavaScript into the application [11], often a tedious and error-prone task. It is, however, very rare that there is a compelling reason to have previously unseen JavaScript code passed to the browser.

## 2 Opportunities

While both secure programming methodologies and static analysis tools have been used to detect "low-level" security issues such as cross-site scripting, we believe that Web software development needs to be elevated to a level where these and other issues are automatically eliminated.

It is our position that, just like in the case of desktop software [6, 9], Web application development frameworks software construction frameworks can expose an excellent opportunity for fundamental improvements in their security. Frameworks and libraries can export security policies that would automatically extend to framework clients. As a result, we could eliminate entire classes of security bugs, leaving developers free to focus on architectural concerns.

Recent interest in enforcing strong security properties on the client side, within the browser sandbox [5, 7, 8, 12], makes us believe that these mechanism will eventually make their ways into mainstream browsers. We think that the moment is ripe to focus on the properties that need to be enforced. Moreover, it creates a great opportunity to automate the process of policy generation by having framework elements expose their security requirements [6, 9].

## 3 Analyzing Dojo Pages

To illustrate what we mean when we talk about security policies, let us consider the Dojo Toolkit, a popular suite of libraries that simplify the development of Ajax applications [4].

### 3.1 Mail Reading Pane

Dojo makes it easy to construct a rich-text email client as shown in Figure 1 by laying out several interface components of predefined Dojo types. E.g., the reading pane at the bottom of the screen is declared as

```
<div id="contentPane" dojoType="ContentPane"
  sizeMin="20" sizeShare="80"
  href="Mail/MailAccount.html" style="padding: 5px">
</div>
```

This declaration allows the pane contents to be loaded from `Mail/MailAccount.html`, an HTML file that can is changed at runtime, depending on the message being selected. However, there is generally no compelling reason to allow content pane contents to contain executable JavaScript. We can assume a "safe default" position and create a simple analysis that will generate no-execute policies for every `ContentPane` on the page. This simple default will go a long way toward preventing cross-site scripting and JavaScript worms, such as Yamanner [3] that propagated through Yahoo! Mail each time

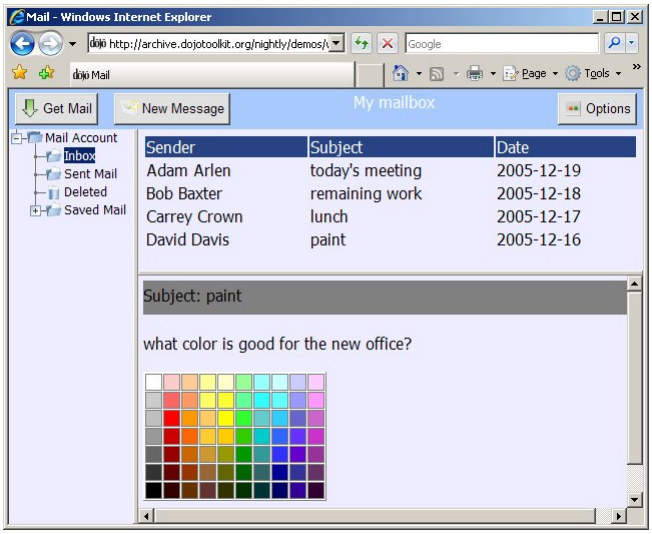


**Figure 1:** Sample mail application constructed using Dojo Toolkit

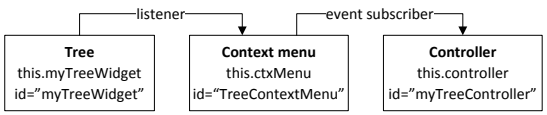


**Figure 2:** Tree widget-menu-controller architecture in Dojo

a user opened a cleverly crafted email message. Moreover, relying on Dojo components to "know" about their own security requirements makes it largely unnecessary to specify policies manually, such with `<noexecute>` blocks in BEEP [7, 8].

### 3.2 Tree Widgets

Similar default policies can be produced for other widgets. Consider the `Tree` widget in the Dojo toolkit that allows one to create multi-level trees. Node labels support HTML and are explicitly specified within the tree as follows:

```
<span class="dojoTreeNodeLabelTitle"
  dojoattachevent="onClick: onTitleClick"
  dojoattachpoint="titleNode"><b>HTML label</b>
</span>
```

Right-clicking on tree nodes allows the user to remove them, add new children, etc. Moreover, labels have event-processing code attached to them to support mouse-clicks, etc. The safe default is to ensure that HTML within the node label declaration does not execute JavaScript code *outside* the Dojo toolkit. A stronger safe default is to assert that code within the tree declaration cannot affect anything outside of the declaration. Enforcement of this policy would be particularly useful as a protection against RSS injection if the tree widget is used to display an RSS feed in a mashup page such as `live.com`.

### 3.3 Context Menus and Tree Widgets

The static analysis of Dojo pages required so far was extremely simple: we just needed to find places in the code with which to associate our static policies. Dojo also supports context menus activated on the right mouse click that can be associated with individual trees. However, the relationship between a tree and its associated menu is less trivial to encode: as Figure 2 shows, it involves creating a *controller* that mediates processing of GUI events by subscribing to context menu events. The abbreviated version of code responsible for context menu creation is shown in Figure 3. A more elaborate static analysis would recover the relationship between the tree, context menu, and its controller, and then issue a policy which would ensure at runtime that context menu-initiated actions are not allowed to affect the DOM *outside the tree* for isolation of different components within a mashup page.

## 4 Conclusions

Recently we have seen a strong trend towards using rich APIs for Ajax application development. Just as in the context of traditional desktop applications, this shift toward a rich frameworks and APIs exposes opportunities for both runtime enforcement and bug finding [1, 6, 9]. Combined with runtime enforcement in the context of the browser [5, 7, 8, 12], the use of frameworks can *automatically* result in significantly more secure applications without additional code annotation burden

```
var DemoTreeManager = {
  djWdgt: null, myTreeWidget: null, ctxMenu = null,

  addTreeContextMenu: function(){
    ctxMenu = this.djWdgt.createWidget("TreeContextMenu",{});
    ctxMenu.addChild(this.djWdgt.createWidget(
      "TreeMenuItem",{caption:"Add Menu Item",
        widgetId:"ctxAdd"}));
    document.body.appendChild(ctxMenu.domNode);
    /* Bind the context menu to the tree */
    ctxMenu.listenTree(this.myTreeWidget);
  },
  addController: function(){
    this.djWdgt.createWidget("TreeBasicController",
      {widgetId:"myTreeController", DNDController:"create"}
    );
  },
  bindEvents: function(){
    dojo.event.topic.subscribe("ctxAdd/engage",
      function (menuItem) { addNode(menuItem.getTreeNode(),
        "myTreeController"); }
    );
  },
  addNode: function(parent,controllerId){
    this.controller = dojo.widget.manager.
                getWidgetById(controllerId);
    var res = this.controller.createChild(parent, 0,
                { title: "New node" });
  },

  init: function(){
    /* Initialize this object */
    this.djWdgt = dojo.widget;
    this.myTreeWidget = this.djWdgt.manager.
                    getWidgetById("myTreeWidget");
    this.addTreeContextMenu(); this.addController();
    this.bindEvents();
  }
};
```

**Figure 3:** Context menu creation code

being placed on the developer. While in this paper we only explore the Dojo toolkit, similar opportunities exist in a staggering variety of Ajax libraries and frameworks [10].

## References


[1] T. Ball, E. Bounimova, B. Cook, V. Levin, J. Lichtenberg, C. McGarvey, B. Ondrusek, S. K. Rajamani, and A. Ustuner. Thorough static analysis of device drivers. In *Proceedings of the European Systems Conference*, 2006.

[2] CGI Security. The cross-site scripting FAQ. `http://www.cgisecurity.net/articles/xss-faq.shtml`.

[3] E. Chien. Malicious Yahooligans. `http://www.symantec.com/avcenter/reference/malicious.yahooligans.pdf`, Aug. 2006.

[4] Dojo Foundation. Dojo, the JavaScript toolkit. `http://dojotoolkit.org`, 2007.

[5] Ú. Erlingsson, B. Livshits, and Y. Xie. End-to-end Web application security. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, May 2007.

[6] S. Hallem, B. Chelf, Y. Xie, and D. Engler. A system and language for building system-specific, static analyses. In *Proceedings of the Conference on Programming Language Design and Implementation*, pages 69–82, June 2002.

[7] T. Jim, N. Swamy, and M. Hicks. BEEP: Browser-enforced embedded policies. Technical report, Department of Computer Science, University of Maryland, 2006.

[8] T. Jim, N. Swamy, and M. Hicks. Defeating script injection attacks with browser-enforced embedded policies. In *Proceedings of the International World Wide Web Conference*, 2007.

[9] M. Martin, B. Livshits, and M. S. Lam. Finding application errors and security vulnerabilities using PQL: a program query language. In *Proceedings of the Conference on Object-Oriented Programming, Systems, Languages, and Applications*, Oct. 2005.

[10] Open Source Applications Foundation. Survey of AJAX/JavaScript libraries (24 libraries surveyed). `http://wiki.osafoundation.org/bin/view/Projects/AjaxLibraries`, 2007.

[11] RSnake. XSS cheat sheet for filter evasion. `http://ha.ckers.org/xss.html`.

[12] D. Yu, A. Chander, N. Islam, and I. Serikov. JavaScript instrumentation for browser security. In *Proceedings of Conference on Principles of Programming Languages*, Jan. 2007.