DOI:10.1145/1461928.1461948

Technical Perspective
Tools for Information to Flow Securely and Swift-ly

By Dan Wallach

BACK IN THE old days of the Web (before 1995), Web browsers were fairly simple devices. They downloaded HTML, laid out the text, and loaded a few images. There was neither JavaScript nor Java nor Flash. There was only the beginning of e-commerce sites, where all the code ran exclusively on the server. While security vulnerabilities certainly existed in both browsers and servers, the server's Web interface was simple enough that an auditor could at least look at it and reason about its security.

Today, it's a different world. With powerful client-side JavaScript and asynchronous Web requests (called "Ajax"), we now have Web "applications" that have significant portions of their state on the client side. This gets even further complicated by "mash-ups," where code from many Web sites might interact within a single Web browser. Building systems like this typically requires careful engineering of the whole system; the server side must be secure even if a non-conforming client is making arbitrary requests.

Meanwhile, a new generation of tools, such as the Google Web Toolkit (GWT), promise to simplify the client-server programming process by blurring the distinction between the client and server. You just write one monolithic program and draw a line through it saying "these parts go on the client and these parts go on the server." This sounds great for improving developer productivity, particularly by abstracting away the inconsistencies and peculiarities of each Web browser's JavaScript runtime system. Because the RPCs are generated automatically, possible information leaks, security holes, or a host of other issues could well present themselves, and the source code is sufficiently abstract so that it's no longer obvious how to audit such a system for correctness.

This concern motivates the following research paper, "Building Secure Web Applications with Automatic Partitioning," where the authors describe a tool they built—Swift—that provides a general-purpose programming language, an extension of Java, for building partitioned Web applications. The secret sauce in Swift is its handling of annotations, placed by the programmer, which declare security properties for objects and variables within the program. These annotations speak toward secrecy or integrity constraints on the data.

For example, say you've got a list of passwords (or hashed passwords or whatever else) on the server and you want to validate a client-supplied password. Clearly, you want to perform that comparison on the server side, such that an attacker cannot access other passwords or impersonate other users. But how do you guarantee such a thing? Swift lets us declare the list of passwords to be "sensitive." We don't want to disclose it to any user. With such annotations, the program partitioning system can figure out that the password-checking logic can only happen on the server side, satisfying the information flow constraints.

Sounds easy, right? Not really. In fact, there's an important problem. Information flow systems are really good at saying "no." You validated the password, and now you want to let the user know. Unfortunately, that very fact is sensitive information because it was derived from sensitive information. We can't release that to the user? That's a problem. Clearly, we need to carve out exceptions to the rules in order to get anything useful done. Swift allows a programmer to make these sorts of exceptions in a controlled fashion, but those will still need to be carefully audited.

Information flow technologies, whether operating statically like Swift, or operating dynamically like the "tainting" mechanism used in Perl, are clearly an important mechanism for building and maintaining secure Web applications. One only has to look at the never-ending parade of cross-site scripting, cross-site request forgery, SQL injection, and other such Web attacks, none of which rely on traditional buffer overflows, to recognize the importance of high-level automated systems built into the development tool chain to improve our assurance that systems are secure. Manual, labor-intensive code audits by security experts cannot scale to support the vast number of new Web applications being deployed each and every day.

The challenge for the research community, with sophisticated tools like Swift, is to simplify the development process, making it easier to get the security labels written properly. Ultimately, our ability to prove that a system is secure, whether Web-based or anything else, is limited by our ability to understand the security model and convince ourselves that the labels we've written and properties we've derived from those labels are consistent with our high-level security goals. Swift takes us a big step closer to achieving those goals.

Dan Wallach (dwallach@cs.rice.edu) is an associate professor in the Department of Computer Science at Rice University, Houston, TX.