Untangling the Woven Web:

Testing Web-based Software

Gary McGraw, Ph.D. David Hovemeyer

Reliable Software Technologies Corporation
21515 Ridgetop Circle, Suite 250
Sterling, VA 20166
(703) 404-9293
http://www.rstcorp.com
gem@rstcorp.com daveho@rstcorp.com

April 1, 1996

1 Introduction

In the early 90's, public interest in the Internet skyrocketed with the introduction of web browsers and hyper text markup language (HTML). As the world-wide web becomes more commercial and people start using browsers to purchase products and do other business, issues of website testing become more and more critical. In addition to web-based commerce, Intranets have also recently become a hot topic.[1] This paper focuses on three general issues in web testing that are applicable to all web-based products: 1) automating remote testing of websites, 2) developing software assessment techniques for Java applets, and 3) issues in website security.

2 Automated Website Testing

As is the case for all software products, part of creating a good website is extensively testing the product. The web presents some interesting new twists on software assessment. One such twist is that web documents present a moving target as they are constantly evolving and changing. More than any other information medium, it is the nature of the web to remain up-to-date. Another twist has to do with the inherently net-based nature of websites. Obviously, websites are "on the net". This means that remotely testing websites over the net is a distinct possibility. In fact, the distributed nature of the web virtually mandates that a web testing environment be net aware. Finally, the interactive nature of many web documents (*e.g.*, forms), makes testing websites more involved than simply following hyperlinks.

Automated web-testing has a history as long (or short?!) as the web itself. In this section we will touch on some existing technologies, discuss general issues in website testing automation, and explore the idea of testing websites with scripts.

2.1 The ideal web testing environment

In order to test websites as thoroughly as possible, a web testing environment should have the following characteristics:

1. *Tests must duplicate as closely as possible the ways in which users interact with websites.* There are many layers of software between the user and a web server. Ideally, a test should be able to duplicate all events that would happen if the test were being executed by an actual user so that it properly tests all of the software layers.

2. *Tests must exercise all features supported by browsers and servers that will be used when the site is in operation.* Web technology is evolving rapidly. In particular, Netscape browsers and servers implement many extensions to published web standards such as HTML and hyper text transfer protocol (HTTP). Ideally, tests should handle these extensions.

[1] The difference between Internet and Intranet is that Intranet networks are "in house" (within an organization), whereas Internet networks connect an organization "to the rest of the world".

3. *Tests should be able to "understand" (at the appropriate level) the test scenarios that they are implementing.* Blindly following hyperlinks is not always adequate. A test should be able to verify that particular features of the web documents it tests are present and function as advertised. When such features are activated, test should ensure that they produce the correct responses. The best way for a test to perform this type of content analysis is for the test to analyze the HTML source code of the documents being sent from the server.

2.2 Why web testing is not easy

The main reason that automated web testing is not easy is because there are many layers of software between the user and the actual documents on a web site:

- User
- Windowing system
- Web browser
- HTTP protocol
- Operating system
- Network
- HTTP server
- HTML document

The test environment must be able to "plug in" at an appropriate level. This is important because there are many types of information that are useful to a test script. For example, all of the following might be critical to the determination of whether or not a test has been passed: the HTTP headers of a received protocol message, the HTML source of the document being accessed, and the pixels of a rendered graphic. In order to access the headers of the HTTP message, a script must have very low-level access (i.e. at the network level). In order to access the HTML source of a document, medium-level access is required. In order to check rendered graphics, high-level access (at the browser level) is required.

Generally speaking, the browser acts as the bridge between the low level network messages and the high level rendered fonts and graphics seen by the user. This means that in theory all of the interesting levels (from the HTTP protocol, through HTML source, up to graphics — and everything in between) are available "inside" the browser. A testing environment which could gain access to the events taking place inside the browser would effectively have access to all this "interesting" information. Unfortunately, browsers have a tendency to hide all of this interesting information.

GUI capture/replay tools provide access to a very high-level of information (including rendered fonts and graphics and GUI interaction) but not medium- and low-level information (HTML, HTTP). This makes it particularly hard for capture/replay approaches to identify HTML document features that should be dealt with in an ideal testing environment.

Another testing approach is to write test scripts that interact directly with the web server — effectively imitating a web browser. While this approach ignores the browser and window system layers of web software, it has the considerable advantage of allowing access

to network-level (HTTP) and document-source–level (HTML) information. Unfortunately, this approach also has the disadvantage of not precisely modeling the interaction between a user and a web server. In addition, a script-based approach requires that the scripting language possess the ability to communicate with a web server in addition to perform analysis on the content of received documents. However, the **perl**[2] scripting language and the **libwww-perl**[3] add-on package provide a very good foundation upon which to implement a web scripting system.

It is possible to implement a test environment that combines the strengths of both capture/replay and scripting approaches. The test environment could record both GUI events (through capture/replay) as well as network communication between the browser and server. This type of technology may eventually make all of the characteristics of our ideal testing environment a reality.

2.3 Making testing effective

The possession of an ideal test environment does not necessarily mean that web testing will be as effective as possible. We suggest two general strategies for maximizing the effectiveness of automated web testing: testing interactivity and website consistency.

The first strategy is to concentrate testing efforts on those parts of a website that require the most complicated user interaction. Interactive forms, for example, provide the typical interface to programs running on web servers. Such programs take the information submitted by the user, use it to perform some computation or action, and present the results of the action back to the user. For example, a form might be used to submit a query to a database program and present the results of the query back to the user. Since the form is implemented as a program with complex input and output, it is important to develop a set of test cases that fully exercise the program's functionality and verify its correct operation.

The second strategy is actually a website design strategy. Simply put: consistency is the key. It is important to keep any website's user interface as consistent as possible. Developing a comprehensive set of test cases does no good if the tests are constantly being rendered useless by changes to the structure of the website. Though a site's content may change, its general "look and feel" should not (especially once a reasonable design has evolved). Note that the second strategy applies equally to a website's interactive sections.

2.4 Tools for Web Testing

There are many tools currently available for website testing. This section briefly describes a few of them.

[2] **perl** is a Unix scripting language available at URL `http://www.perl.com`.

[3] The **libwww-perl** library can be found at URL `http://www.sn.no/~aas/perl/www/`.

2.4.1 Link Testers

Link testing programs navigate a web site, checking to see that all hyperlinks refer to valid documents. Since this type of testing can be done completely without human intervention, link testing programs are a very useful part of the web tester's toolkit.

Name of tool	Comments
CheckBot	∘ configurable for sites and paths to check ∘ generates HTML report of search results ∘ `http://dutifp.twi.tudelft.nl:8000/checkbot/`
EIT Link Verifier Robot	∘ configurable for sites, search strategy, etc ∘ generates HTML report of search results
MOMSpider	∘ highly configurable (can specify sites and pages) ∘ free: `http://www.ics.uci.edu/WebSoft/MOMspider/` ∘ does not handle forms or netscape extensions
RST Link Tester	∘ can navigate arbitrary HTML content ∘ handles netscape extensions (HTML and HTTP) ∘ currently in Alpha testing

Table 1: Tools for testing links.

2.4.2 Test script development environments

Software for developing scripts for test scenarios is only now becoming available. Such tools allow scripts to be specified in complete detail and then executed automatically. One example of such a tool is Pure Software's Performix. Pure Performix combines a GUI capture/replay tool with an HTTP capture tool, allowing scripts to control both GUI events and network events.[4] RST also has a script development web-testing product in the works.

3 Testing Java Applets

The power of HTML has been recently and radically expanded with the introduction of Sun Microsystem's Java programming language. Java allows for the development of platform-independent programs that run on webpages. Java significantly expands the ability of web-masters to provide interactive, dynamic web content including: animation, GUI interfaces, complex computations, file I/O and data structures.

As the Java programming environment matures and Java starts to be used to create "real" applications, programmers will require that sophisticated software engineering tools be co-opted from C and C++ development environments for use with Java. Software testing tools make up one important class of these software engineering tools. This section briefly describes some of the issues that we encountered while converting a C/C++ code coverage

[4]See URL `http://www.pure.com/products/pureperformix/index.html` for more information.

tool for use on Java code. Our successful conversion resulted in the first code coverage tool suitable for use in testing Java applets [Binns and McGraw, 1996].

Many proficient C and C++ programmers find coming up to speed in Java a fairly straightforward task. However, these programmers have become accustomed to having a large set of development tools at their fingertips. It is clear that making these tools available for Java is an important step in creating a solid Java development environment. Developers creating such tools are forced to grapple with some fundamental differences between Java and C/C++.

One good thing about the Java programming language and environment is that many of the development tools that other commercial-grade languages have enjoyed can be adapted for use in Java. This is a direct result of the common ground that Java shares with C++. With this in mind, Reliable Software Technologies recently took on the task of enhancing a version of the popular C/C++ code coverage tool the PiSCES Coverage Tracker^{tm} so that it could operate on Java code.[5] Throughout the development of our Java coverage tool, we discovered that many of the standard techniques for performing dynamic analysis on C and C++ programs were not amenable to the Java programming environment.

Before we go on, we must briefly introduce the concept of code coverage [Myers, 1979]. The PiSCES Coverage Tracker for Java^{tm} (Java Tracker) is a code coverage measurement tool. Given a set of test cases and the source code for a program (in the current case a Java applet or application), Tracker measures which parts of the source code are "exercised" during program execution on the test cases. Code coverage is meant to aid in the development of a thorough and rigorous set of test cases and helps a programmer ensure that every aspect of a piece of code is exercised during the testing phase. The C/C++ version of Tracker provides many sophisticated coverage measurement techniques: function, branch, condition/decision, and multiple condition coverages. The Java Tracker prototype currently supports function and branch coverages. Function coverage measures which subset of defined functions have been exercised during execution over a given test set. In the Java case, this amounts to a coverage measurement over methods associated with a class. Function coverage is the most basic level of coverage. Branch coverage (also known as decision coverage) measures the number of explicit branches that are exercised during testing on a test set (e.g., which specific branches of a case statement have been tested). Branch coverage returns numbers that are relative to the total number of branches in the source code.

Coverage measures are achieved by a software tool that is inserted into the compilation pipeline. In this pipeline, a pre-processor is invoked on the source code. The original source code itself is left intact, but new code is inserted by the coverage tool. The inserted code performs the work of recording what pieces of the original program have been executed. This process is known as "code instrumentation". Coverage information is stored dynamically as the instrumented program is executed. For this reason, coverage measurement is known as a dynamic software analysis technique.

In the development of Java Tracker, we were forced to change the way coverage instrumentation is integrated into the compilation pipeline. This was a direct result of divergent

[5]The PiSCES Coverage Tracker^{tm} and the PiSCES Coverage Tracker for Java^{tm} are part of Reliable Software Technologies Corporation's PiSCES Software Analysis Toolkit$^{(r)}$.

build environments for Java and C/C++. But not only is the compile-time pipeline different, the run-time environment is different too.

The Java applet/application run-time environment is different in many critical ways from the usual C and C++ environments. The main difference has to do with the level on which the language allows access to the machine. C and C++ can access a machine at the "bare metal" level, meaning that many interesting programming tricks can be performed in order to improve tool performance. This is important in a competitive environment where the performance of a tool is critical to its utility and ultimately to its commercial success. For Java, on the other hand, access to the machine is severely limited (mostly because of security concerns).

The same features that make Java so attractive to developers turn out to make creating software engineering tools problematic. For example, in C and C++, a tool can easily trap signals, start child processes, communicate with the operating system, etc. Nearly all of these kinds of low-level functions are restricted in Java, severely so for an applet running across a network. We have explored some possible solutions to these problems in the development of Tracker, but no clear-cut answers are available, and much work remains to be done.

We have been forced to address many problems head-on throughout the development of the PiSCES Coverage Tracker for Java^{tm} (now available free on the net at `http://www.rstcorp.com/java.html`). As we do more Java development and craft solutions to these problems, more questions are raised and better solutions are discovered. We encourage all Java developers to devise a reasonable and thorough testing strategy before releasing their applets. The Java Tracker should prove useful to such efforts.

4 Website Security

One of the fundamental concerns that Internet users have (and rightly so) is site security. Web-based documents and services have no special immunity to attacks. In fact, several very serious security flaws have recently been revealed that cast doubt on many of the claims made by vendors regarding the security of their web browsers and other products. The Java language bugs are the most serious and involve security holes that have not yet been patched. In this section we will briefly mention several such security holes. The purpose of this list is to stress the essential role that software testing should play in web-product development. The fact is that security concerns are especially important for web-based products because such products have more exposure to attack than usual.

Netscape encryption bugs: Encryption plays a key role in setting up secure transactions for web commerce. As such, Netscape has made a major effort to enable safe encrypted communication between websites and clients. Unfortunately, Netscape's implementation of the RSA encryption algorithm was buggy. David Wagner of Berkeley discovered a mistake in Netscape's pseudo-random number generator that could be exploited to break the encryption. The Netscape encryption algorithm has since been patched [Levy, 1996].

Bypassing Netscape's security manager: Godmar Back, a Computer Science student at the University of Utah, has published a webpage explaining how to circumvent Netscape's security manager. It is possible to disable Netscape's security manager by replacing it with a Null manager so that Netscape allows file access through Java applets. (Note that Java applets are not supposed to be able to read and write arbitrary files.) This can be done by changing a few bytes of Netscape's executable file with any sort of hex editor. [Back, 1996]

Jumping firewalls with Java: Researchers at Princeton have discovered a serious security problem with Netscape Navigator's 2.0 Java implementation. (The problem is also present in the 1.0 release of the Java Development Kit from Sun.) A Java applet is normally allowed to connect only to the host from which it was loaded. However, this restriction is not properly enforced. A malicious applet can open a connection to an arbitrary host on the Internet. At this point, bugs in any TCP/IP-based network service can be exploited. If the user viewing the applet is behind a firewall, this attack can be used against any other machine behind the same firewall. The firewall will fail to defend against attacks on internal networks, because the attack originates behind the firewall. Netscape has made a patch available to cure this security problem [Dean et al., 1996].

CGI security holes: CERT reports that a security vulnerability has been reported in example Common Gateway Interface (CGI) code, as provided with the NCSA `httpd 1.5a-export` and APACHE `httpd 1.0.3` (and possibly previous distributions of both servers) [CERT, 1996]. The example code contains a library function that contains a vulnerability. As a result of this problem, a remote user may retrieve any world readable files, execute arbitrary commands, and create files on the server with the privileges of the `httpd` process that answers HTTP requests. This may be used to compromise the http server and under certain configurations gain privileged access. Several workarounds are possible and are listed in the CERT Alert [CERT, 1996].

Major Java Security Hole: [Dean et al., 1996] The same researchers at Princeton who discovered the Firewall Jumping problem have discovered another serious security flaw in the Java programming language. This one allows a malicious Java applet running under Netscape Navigator (version 2.0 or 2.01) to execute arbitrary machine code. Using this hole, they have implemented an applet that exploits the flaw to remove a file. No fix for this hole has been issued as of March 28, 1996 [Dean et al., 1996].

It is clear that security concerns for web-based products are legitimate, and are not just so many cries of "wolf". The existence of such holes points out a real need for some sort of automated security testing geared toward use on web products. We believe that testing for security is an important research endeavor that the software testing community should explore. At RST, an ARPA-sponsored research project into security testing is underway. One important application of our work will be a tool that can assess website security.

5 Conclusions

In this paper, we have addressed three major issues of testing web-based software: automating web testing, developing testing techniques for Java applets, and website security testing. As companies migrate toward web-based computing (including both Internet and Intranet usage), these issues will become even more important than they are today. Testing web-based software is clearly an area that is ripe for the application of software testing expertise.

Acknowledgments

Part of Dr. McGraw's research on security issues is sponsored by ARPA under contract number **F30602-95-C-0282**. Mr. Hovemeyer's work on web testing is partly funded by NIST's Advanced Technology Program under award number **70NANB5H1160**. THE VIEWS AND CONCLUSIONS CONTAINED IN THIS DOCUMENT ARE THOSE OF THE AUTHORS AND SHOULD NOT BE INTERPRETED AS REPRESENTING THE OFFICIAL POLICIES, EITHER EXPRESSED OR IMPLIED, OF THE ADVANCED RESEARCH PROJECTS AGENCY, THE DEPARTMENT OF COMMERCE, OR THE U.S. GOVERNMENT.

References

[Back, 1996] Back, G. (1996). How to bypass netscape's security manager. Web document at URL: `http://www.cs.utah.edu/ gback/netscape/bypass.html`. Also posted to `comp.lang..java` 1/29/96.

[Binns and McGraw, 1996] Binns, A. and McGraw, G. (1996). Building a java software engineering tool for testing java applets. In Proceedings of the Intranet96 NY Conference, New York.

[CERT, 1996] CERT (1996). Cert alert ca-96.06. CERT stands for Computer Emergency Response Team.

[Dean et al., 1996] Dean, D., Felton, E., and Wallach, D. (1996). Java security: From hotjava to netscape and beyond. Web documents at URL: `http://www.cs.princeton.edu/ ddean/java`. Also see the related CERT Alert document CA-96.05.

[Levy, 1996] Levy, S. (1996). Wisecrackers. Wired Magazine, 4(3):128–134 and 196–202.

[Myers, 1979] Myers, G. J. (1979). *The Art of Software Testing.* Wiley-Interscience, New York, NY.