Automated Client-side Monitoring for Web Applications

Shauvik Roy Choudhary and Alessandro Orso
College of Computing – Georgia Institute of Technology
{shauvik, orso}@cc.gatech.edu

Abstract

Web applications have become very popular today in a variety of domains. Given the varied nature of client-side environments and browser configurations, it is difficult to completely test or debug the client-side code of web applications in-house. There are tools that facilitate functional testing on various browsers, but they cannot mimic all of the possible client-side environments. In modern web browsers, the client-side code can interact with numerous web services to get more data and even to update itself, which can in turn affect the behavior of the client in unforeseen ways. In these situations, monitoring the client-side code allows for gathering valuable runtime information about its behavior. In this paper, we propose a technique for monitoring and detecting failures in client-side code. We also present a preliminary evaluation of the technique where we discuss its efficiency, effectiveness, and possible application scenarios.

1 Introduction

With the advent of new web application technologies, many applications today are being made for the web. These web applications serve a variety of client platforms that range from desktop computers to embedded devices such as mobile phones. On each device, the web application's pages are rendered by a number of user agents, including fully featured browsers, embedded applications (*e.g.,* flash), and custom mobile UI agents with limited functionality. Given this varied nature of user agents, it is often difficult for developers to ensure that their code runs correctly and efficiently in all possible contexts.

Typically, developers test their applications on the most popular browsers before shipping their code. This kind of testing is useful, but necessarily limited to the few deployment scenarios considered; after deployment, when the web application runs on the user platforms, incompatibilities with the environment may result in unforeseen behaviors and, ultimately, application crashes. In these cases, developers can easily get information on the server-side application (*e.g.,* from logs), but they typically have little information about the remotely-running client side, which is the part actually misbehaving.

Existing techniques allows for collecting some client-side runtime information, such as the URL of the page where an error occurred and error messages. This information, however, provides only limited support for debugging problems occurring in the client-side code. To address this limitation, we propose a novel technique for remote monitoring of client-side applications. Our technique transparently injects a client agent into the web application's code and uses a server-side component to collect a variety of monitoring data from the agent and to control it. The server-side component can analyze the collected data and take corrective actions through the agent. Our technique is efficient and flexible, in that it can support a variety of monitoring tasks without imposing too much overhead on the web application being monitored. In the rest of the paper, we describe the technique and present a preliminary evaluation of its efficiency and usefulness.

The main contributions of this paper are: (1) an approach for monitoring client-side code in web applications; (2) a prototype implementation and preliminary evaluation of the approach; and (3) a discussion of some possible application scenarios for our approach.

2 Motivating Example

In this section, we introduce a motivating example that we use to illustrate a possible issue with a web application running in different environments and how our technique could address it. (Due to space limitations, in the discussion we assume a basic knowledge of web technologies.) Consider the following code snippet, from a web application, that renders a client-side UI element based on the value of a cookie named `status`.

```
1. if(getCookie('status') == '') {
2.   setCookie('status','display');
3.   reloadPage();
4. } else if(getCookie('status')=='display') {
5.   updateWithAjaxContent(url, divId);
6. }
```

If the cookie is not set, the code sets its value to `display` and reloads the page. Otherwise, if it is set to `display`, the code calls function `updateWithAjaxContent`. This function makes an Ajax request to the `url` passed as a parameter and populates the data received into the HTML division associated with identifier `divId`, also passed as a

parameter. Consider now a case where cookie `status` has been set by some external entity in the environment (*e.g.,* a third party client or a user-level script) to a different value prior to the execution of the above code snippet. In such a case, the code would not perform any action, which may lead to an unexpected behavior if the programmer did not anticipate this situation.

Note that this is just a possible instance of a more general problem that involves other configurations and environment variables besides cookies. Such variables are in a global space within the browser, can be set in a programmatic way (*e.g.,* by a browser plug-in) or interactively (*e.g.,* by showing a dialog box to the user), and can affect the behavior of web applications running in the browser. Therefore, this and similar problems are not uncommon in today's increasingly feature-rich client environments and are hard to investigate and debug without some access to runtime information about the client-side application.

3 Our Technique and Tool

As we mentioned in the Introduction, our technique is based on using a proxy to transparently inject a client agent into the code of a web application to be monitored. At runtime, the agent can collect a variety of runtime information and send it (all or in part) to a server-side component.[1] The server-side component can then analyze the collected data and possibly take corrective actions through the agent.

Figure 1 provides a high-level view of our technique and tool. The client side simply consists of a Javascript-enabled web browser. On the server side, our technique uses a rewriting reverse proxy that performs three main tasks. When a request is sent from the client browser, the proxy simply directs it to the appropriate web server. When a response is received from the web server, the proxy instruments the code in the page being sent to the client browser to add the client-side agent (*CSA*). Finally, when monitoring data is received from the agent, the proxy directs it to the Command and Control (*CnC*) server. In the following sections, we provide more details about the technique and our prototype implementation.

3.1 Rewriting Reverse Proxy

Reverse proxies are used mainly for load balancing and caching server data. They can also be leveraged to do additional tasks, such as SSL encryption and data compression, and to provide security to web servers by shielding them from direct Internet traffic. For our prototype, we used an open-source non-caching proxy server called TINYPROXY (`https://www.banu.com/tinyproxy/`). The proxy server was configured to route HTTP requests that contain monitoring data (identified using a URL regular-expression)

[1] We are aware of possible privacy issues with the approach, but at this stage of the research we are more concerned with its feasibility.

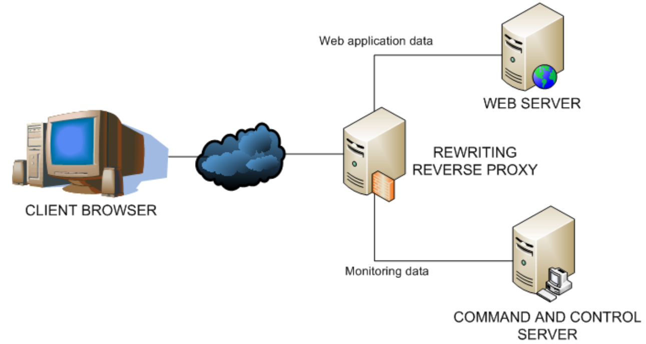

Figure 1. High-level view of the technique.

to the CnC server, while routing all other traffic to the appropriate web application running on the web server.

We also extended TINYPROXY with rewriting capabilities as follows. When a request is made to a web application, the proxy rewrites the server response so as to transparently inject a reference to a CSA located on the CnC server. In this way, the CnC server can get access to all of the resources on the client browser that were loaded by the web application because the CSA is loaded from the same domain (and can thus bypass the browser's cross-domain restrictions).

3.2 Client-side Agent

A CSA is a Javascript program that runs in the user browser. Its main purpose is to connect to the CnC server to provide data and receive commands. Since HTTP does not support push communication, we have implemented this client agent using a polling approach. The initial time interval for the polling is defined in the agent's code, but the CnC server can change such interval using a remote command. In current browsers, the polling operation is performed between normal executions of other client-side scripts. Therefore, if a CSA is performing a computation-intensive operation, it may slow down the client-side code of the web application that is being monitored. This problem, if present, could be addressed by leveraging Web Worker threads, which are a part of the upcoming HTML5 standard [5]. Using this feature, a CSA could be started as an independent worker thread that operates in parallel with the other client-side code.

In a practical scenario where there are multiple CSAs involved, it is necessary to identify the client agents, so as to be able to keep track of the commands sent to them and their responses within a web session. To achieve this goal, our current implementation creates a cookie named `agentId`, sets it to a randomly-generated alphanumeric ID, and associates the ID with a specific agent and session. An alternative approach would be to use the session id cookies set by some web servers (*e.g.,* `PHPSESSID` for PHP and `JSESSIONID` for Tomcat).

```
 1. <commands>
 2.   <cmd>
 3.     <id>8de9</id>
 4.     <name>ALERT</name>
 5.     <param>Hello World!</param>
 6.   </cmd>
 7.   <cmd>
 8.     <id>3bsd</id>
 9.     <name>DUMP</name>
10.     <param>myObj</param>
11.     <param>myArray</param>
12.   </cmd>
13. </commands>
```

Figure 2. Example of command sent from the CnC server to a CSA.

```
 1. <responses>
 2.   <resp>
 3.     <id>8de9</id>
 4.     <status>1</status>
 5.   </resp>
 6.   <resp>
 7.     <id>3bsd</id>
 8.     <message>{"aString":"Howdy",
                  "aInteger":10,
                  "aBoolean":true}</message>
 9.     <message>[1,"foo","web"]</message>
10.   </resp>
11. </responses>
```

Figure 3. Example of response sent from a CSA to the CnC server for the commands in Figure 2.

3.3 CnC Server

The CnC server has prior knowledge of the client-side program as it has access to the AST of the Javascript source. When a CSA first loads, the CnC server creates an identifier for it, as described in the previous section, and starts queueing commands for the CSA. The CSA fetches these commands and sends back a response. To illustrate, figures 2 and 3 provide some simple examples of commands and responses. Multiple commands can be sent to an agent at a time by wrapping each command in a `<cmd>` tag. In the example, the server sends two commands. The first command results in displaying an HTML alert box to the user with the specified message. On success, the tag `success` of the corresponding response is set to one. In the second command, the server requests a dump of the values of object `myObj` and array `myArray`, and the CSA returns the respective encoded values back to the server in its response.

4 Preliminary Evaluation

The goal of our evaluation is to measure the practicality of our approach in terms of feasibility and overhead imposed on the user. To this end, we used our prototype tool on a set of web applications.

Table 1 lists the 10 web applications that we used for our evaluation. Interactive Test, Number Guess, and Chat Client are sample rich-client applications distributed with the Echo2 Ajax framework (`http://echo.nextapp.com/site/echo2`); Mail and Showcase are demo applications distributed with Google Web Toolkit (GWT – `http://code.google.com/webtoolkit/`); Joomla, Drupal, and Wordpress are open source content-management systems and blogging packages; finally, iGoogle and Amazon.com are commercial web applications. All of our subject applications are characterized by having a large proportion of their code on the client side. The Echo 2 applications are Ajax-based applications that contain a considerable amount of client-side code. The GWT applications are written in Java and then compiled to Javascript and HTML. Joomla, Drupal and Wordpress use large client-side libraries to render their UI elements in the browser. iGoogle and Amazon.com have a great deal of client-side widgets and validation code written in Javascript.

Table 1. Instrumentation overhead.

Subject	Avg. Download Time (μs) Normal	Avg. Download Time (μs) Instrumented
Interactive Test	30,258	30,922
Number Guess	25,625	25,731
Chat Client	29,327	30,782
Mail	27,950	29,018
Showcase	29,318	29,909
Joomla	274,122	280,845
Drupal	171,109	172,730
Wordpress	242,704	250,782
iGoogle	71,461	74,991
Amazon.com	808,693	810,541

To measure the efficiency of the proxy's transparent injection, we randomly chose one page in each of the subject applications and measured the time necessary to instrument it. To gather this information, we collected and compared the loading time for the page with and without injection. To account for transitory effects, we loaded each page 100 times and averaged the measurements. The results are shown in Table 1. As the table shows, the proxy took between 0.1 and 8 milliseconds to inject the agent, which is negligible in the overall loading process. Moreover, the proxy we chose was not designed to be a rewriting proxy and could be further optimized (*e.g.,* by instrumenting only the main page of an application instead of all server-side resources associated with content type `type/html`).

We also measured the overhead imposed by a CSA that iterates through all window objects and inspects all global elements in the window. Table 2 shows the results of this part of the evaluation in terms of number of objects encountered (divided in objects from the same and different domains) and time taken by the agent to iterate through the objects. (Note that, as we discussed in Section 3.1, our rewriting proxy allows both the CnC server and the web server to appear as a single source to the browser, thus overcoming cross-domain restrictions.) As the results show, for

Table 2. CSA Performance.

	Same Domain		Cross Domain	
Subject	*# Obj.*	*Time (ms)*	*# Obj.*	*Time (ms)*
Interactive Test	147	2	146	2
Number Guess	144	3	144	2
Chat Client	147	2	146	2
Mail	1,286	9	1,286	10
Showcase	4,490	30	4,490	31
Joomla	229	3	229	3
Drupal	118	1	118	2
Wordpress	176	3	176	3
iGoogle	618	6	602	5
Amazon.com	314	4	303	3

the cases considered the agent was able to monitor the objects' state and provide information to the CnC server fairly efficiently. These results provide initial evidence of the feasibility of the approach.

5 Application Scenarios

In this section, we present some possible application scenarios for our technique. These are just examples, as our approach is generic enough to support many other usages.

5.1 Remote Error Detection & Debugging

Javascript errors can be caught by surrounding the Javascript code with a try-catch block or by associating the `window.onerror` event to a utility function. The CnC server could check if error handling code is present and, if so, redirect the handler to a CSA that performs some operation, such as trying to recover from the error or logging relevant data, before calling the original error-handling routine.

5.2 Metrics Collection

A CSA can be helpful in collecting and reporting a number of dynamic metrics on the client side, such as code coverage and client-activity profiles. Periodic collection and reporting of metrics to the CnC server would allow for monitoring client-side execution. Moreover, the server could dynamically instruct the CSA to collect more detailed information for relevant parts of the code (*e.g.,* hot spots).

5.3 Memory Profiling

Using a CSA, the CnC server could get the count of live variables, objects, and arrays in the client-side code and use it to calculate the total memory usage at specific points in time. A CSA could also count the number of instances of one type of object or the length of the complete DOM tree.

5.4 Security of Web Services

There are many issues related to enforcing service-level agreements within web services [3]. In particular, monitoring web services requires end-to-end connectivity. Our approach could help in the case of web services that are accessed through client-side Javascript applications. For instance, our technique could dynamically check for interface mismatches [4] or monitor scripts loaded from external sources that might lead to web-security attacks.

6 Related Work

The existing approach most closely related to ours is AjaxScope [1, 2]. AjaxScope identifies Javascript code in web pages at runtime and performs dynamic source-level instrumentation to distribute test and analysis tasks among all users of a web application. Although it shares some technical similarities with our CSA-based technique, AjaxScope has a different goal. Most importantly, AjaxScope does not allow for providing commands to the instrumented Javascript, which limits the applicability of the approach.

7 Conclusion

In this paper we presented our approach for automated monitoring of client-side code, a proof-of-concept prototype tool that implements our approach, and an initial empirical evaluation. Our results, although preliminary, provide initial evidence of the feasibility of the approach.

In future work, we will perform a more thorough empirical evaluation. We will also study ways to optimize the approach by leveraging characteristics of the new HTML standard, such as Web Worker threads. In general, we believe that advances in web technologies will have a dual effect: on the one hand, they will introduce additional complexity on the client-side execution and call for more monitoring and control over it; on the other hand, they will provide features and capabilities that help support such monitoring and control through approaches such as the one we propose.

References

[1] E. Kiciman and B. Livshits. AjaxScope: A Platform for Remotely Monitoring the Client-side Behavior of Web 2.0 Applications. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles*, Oct. 2007.

[2] E. Kiciman and Helen J. Wang. Live Monitoring: Using Adaptive Instrumentation and Analysis to Debug and MaintainWeb Applications. In *Proceedings of the 11th Workshop on Hot Topics in Operating Systems*, May 2007.

[3] Heiko Ludwig Web Services QoS: External SLA and Internal Policies Or: How do we deliver what we promise? In *Proceedings of the 4th International Conference on Web Information Systems Engineering Workshops*, WISEW'03

[4] W. G. Halfond and A. Orso. Automated Identification of Interface Mismatches in Web Applications. In *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, Nov. 2008.

[5] Ian Hickson, Google. Web Workers – Draft Recommendation `http://www.whatwg.org/specs/web-workers/current-work/`, January 2009.