Continuous Integration Testing of Web Applications by Sanitizing Program Input

Bouchaib Falah, Manal Hasri, and Sebastian Schwaiger

***Abstract*— Software testing is known to be the most difficult and expensive phase of the software development life-cycle. When it comes to testing web based applications, this task is even more difficult because of the peculiarities of such applications. Many techniques and tools have been developed to test web applications and automate the different activities involved in this practice, both for the server side and the client side of the application. However, these tests are performed separately; there exists no such a tool that supports integration testing for web applications where server and client side are combined and tested as a group. In this paper, we provide a method that allows testing both parts at the same time. The aim of this method is to sanitize data sent from the server before it is received by the client. Hence, it ensures that the client receives the right response from the server. It supports teams by applying agile development methods such as continuous integration.**

***Index Terms*— Client side, integration testing, server side, web application.**

I. Introduction

Computer technology with its vast diversity has become an essential element in all kinds of human activity. The Internet and computer software grew in less than two decades to achieve the status of the largest information repository in human history. By providing efficient, fast, consistent and authentic tools in the form of internet and computer software, information technology is penetrating human life and is playing an important role in changing lives of so many people around the globe.

In the last decade, a significant growth of the demand of Web-based applications has been recorded, especially to serve business purposes. As more organizations are using the web to offer their services and to be reached by a wide range of customers, their requirements for reliability, security, scalability and accessibility are being stricter. In order to ensure that a web application conforms to these quality attributes, testing becomes a crucial part of the development life-cycle. Testing web based applications is different from testing other software systems because of factors related to performance and user experience [1]. These factors have to be taken into consideration for the application to work correctly in all situations. Some of these factors are:

- Numerous application usage paths are possible depending on the set of tasks that the users want to perform
- Large number of users will access and use the application concurrently
- Users have different backgrounds, different technical skills and some or all functionalities should be self-explanatory
- Different types of browsers might be used to access the application
- Security measures should be more stringent

There exist many techniques for testing server-side and client- side code separately. As an example, Selenium [2] is a tool that allows writing automated user interface (UI) tests for web applications in any programming language against any HTTP website using any mainstream browser. PHPUnit [3] and JsUnit [4] are tools used for testing and automating test operations for web based applications. These tools are efficient in reducing not only the defects of a program, but also the time for detecting and resolving those defects. However, none of these tools supports testing server side and client side as a whole. This is insufficient since, especially for agile development, the back-end code changes at the same time as the front-end code. Hence, faults may occur because of invalid data sent by the server, even if the UI is bug free. The approach that we are suggesting overcomes this problem by sanitizing the data sent from the server. This data is validated before it is fed to the web application and displayed to the user. Therefore, this approach will ensure that the server is sending the right data in response to the client's request. In the remainder of this paper, we will first start by reviewing the related approaches in section 2, sections 3 will present our approach in more details with examples of how it can be

applied to web application integration testing. Section 4 will draw a conclusion and section 5 present some possible future work.

II. RELATED WORK

As for any software system, a web application testing process defines a set of staged testing activities; each one is considering a different testing level [1]. Currently, there is no similar method available to sanitize input data arriving at the client. However, other methods have been explored in order to ensure the correct functioning of web applications by performing unit testing, usually as the first activity, followed by integration and ending by system and acceptance testing [1]. These stages are defined as follow:

- Unit testing: units such as the web pages, scripting modules, forms, applets, servlets, or other Web objects are tested separately at this level
- Client page testing: consists of testing the end user interface of the application
- Server testing: Typical results of server page execution will be data retrieving / storing into a database, or generation of client pages showing results of the elaboration required by the user. Server testing should ensure that these operations are performed correctly
- Integration testing: this phase considers sets of related Web pages in order to assess how they work together, and identify failures due to their coupling.
- System testing: its purpose is to allocate defects at the level of the whole Web application. It is done through black box testing and tries to identify failures in the externally visible behaviour of the application.
- Acceptance testing

Several frameworks related to the testing areas as listed above are the following:

1) PHPUnit

PHPUnit is an infrastructure that allows the programmer to check whether code written in PHP behaves as expected through performing a battery of tests, runnable code-fragments that automatically test the correctness of parts (units) of the software [3]. The results of these tests are reported with details of the tests that failed. The main advantage of using such infrastructure is that tests are fine-grained which allows improving the overall design of the system [3]. However, this infrastructure is only used for testing the back-end functionality of a web application.

2) JsUnit

JsUnit is a unit testing framework for client-side code written in JavaScript. It includes a platform for automating the execution of tests on multiple browsers and machines under different operating systems [4]. It provides instant feedback on which test failed on which browser on which operating system and creates logs for each test run. The automation of runs is done through a Stand-alone Test [4]. When run, Stand-alone Test starts each browser in turn and runs the specified test page without any user interaction. In case of failure, a failure message is displayed specifying which test failed in which browser [4]. However, JsUnit is not appropriate for submitting forms that interact with a web server. It is intended to test purely client-side functionality. Hence, any form submission is done through the use of mock objects (in this case, mock forms) in order to create a new unit test.

3) Selenium

Selenium is a test tool that allows you to write auto- mated user-interface tests for web applications in any programming language against any HTTP website using any mainstream browser [6]. It performs automated browser tasks by driving the browser's process through the operating system. Selenium tests run directly in a browser instance, just as if a uses would access them. These tests can be used for both acceptance testing (by performing higher-level tests on the integrated system instead of just testing each unit of the system independently) and browser compatibility testing (by testing the web application on different operating systems and browsers) [6].

III. OUR APPROACH

A. *Theory*

In this paper, we are suggesting an approach that complements already existing front-end and back-end testing techniques. This is achieved by monitoring the traffic retrieved from the server component in response to a client request. Especially we focus on data retrieved during AJAX [5] calls as those cannot be reliably tested with current techniques (to our knowledge no front-end testing framework can intercept AJAX calls). Fig 1 illustrates our approach through AJAX data validator.

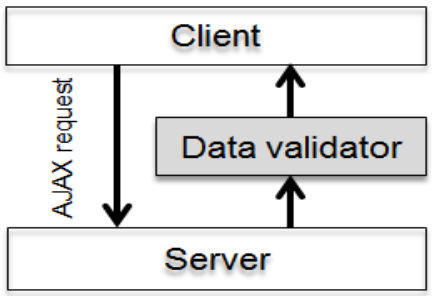

Fig. 1. AJAX Data Validator

B. Selenium User Extension

Our implementation consists of a plug-in for Selenium [2]. It validates the answers the server gives to AJAX requests issued by the web page under test. It's trivial with Selenium to check JavaScript variables for their value, but the need to check every single variable for its content can be tremendously high. We'll overcome that by comparing the data retrieved before they are fed to the application's request handler.

We're assuming that all data transfers between the client and the server are serialized to JSON [6] format, thus enabling us to easily parse and evaluate the packet's content on the client without the need to develop any additional parsing component (current browsers already offer the "JSON" [7] object for the purpose of (de-)serializing JavaScript object in a safe manner). So, the tester simply provides a serialized version of a template object to our plug-in against which the data from the server is matched.

Finally, many web application servers can be configured to automatically transmit JSON (instead of XML for example). Multiple AJAX requests can be tested as every finished request is put in a queue and matched against the first template object contained in the testing queue.

For the sake of simplicity, our demo implementation assumes the use of jQuery [8] in the web page under test. This framework allows us to hook all server calls, both synchronous and asynchronous in a way that is transparent by the web page consuming the framework services.

By implementing the test code as a Selenium User Extension [9], it is callable by Selenium test cases that may already exist for the web page.

To capture the data sent by the server, a small script is injected in the web page's source code by calling JavaScript's eval() method from the Selenium Extension. Direct addition of the callback handler via code execution in the context of the user extension failed so far. Unfortunately jQuery handles requests addressing the server the code was loaded from different from requests that retrieve data from servers out of the domain the calling script was loaded from. This is due to the "same origin policy" [10]. This security related feature disables calls to server from a different domain to prevent CSRF attacks (cross-site request forgery) [11]. To willingly overcome this feature, jQuery supports a technique called "JSONP" [12]. Here the AJAX call is not done with POST but instead new code is loaded by inserting a ¡script¿ tag containing the request parameters in the "src" attribute. It also has to include the name of the callback that is called in the user code. The code retrieved from the server will then contain valid JavaScript code that calls the previously specified callback function passing the actual payload data as function argument. The reader must be aware that this method may only be used for trustworthy servers as any JavaScript code can be sent as response to the script request!

To finally activate the hooks (one for call to the server of the same domain and one for other domains), a Selenium Test method must be called to install the hook. This has to be done once on every page refresh by the test suite to set-up the hooks.

```
Selenium.prototype.doInjectAJAX
CallLogger=
  function(locator, unused) {
  var window =
  selenium.browserbot.getCurrentWindow
  ();
  if(typeof
  window.seleniumAJAXTesterCache
     == "undefined") {
    window.seleniumAJAXTesterCac
    he = []; window.eval(
    "$(document).ajaxComplete
    (" + "function(event,
    xhr, options) {" +
      "seleniumAJAXTesterC
      ache." +
      "push(xhr.response
      Text);" +
    "}); ");

    window.eval(
      "window.jsonpCallb
      ack =" +
      "function(data)
      {" +
        "seleniumAJAXTesterCac
        he." +
        "push(JSON.stringify
        (data));" +
      "}"
    );
  }
}
```

To activate the JSONP hook, jQuery's normal JSONP callback must be replaced with the one that was just added to the web page's code. This has to be done for every AJAX call made, as the callback changes with every new request. The location to do this is the method that takes the request: $.ajax(). It is replaced with a custom version:

```
// Store a reference to
the original method.
var originalMethod =
  selenium.browserbot.
  getCurrentWindow().$.ajax;
selenium.browserbot.getCurre
  ntWindow().$. ajax =
  function(url, settings)
```

```
{
  settings.jsonpCallback =
  "jsonpCallback";
    return originalMethod(url,
    settings);
}
```

To access the data stored in the request cache and match it against templates specified in the test cases, another Selenium User Extension is required. The matching method doesn't simply compare the string representation of the template and retrieved object. It first parses the objects and then serializes them again to ensure, that no white space characters distort the matching process.

```
Selenium.prototype.asser
tAjaxCall =
  function(locator,
  templateObjectAsString)   {

    //  Parse the template object
    var  templateObject =
      JSON.parse(templateObject
      AsString);

    //  Get the oldest result of
    //  an AJAX query
    var  retrievedObjectAsString =
      selenium.browserbot.getCurre
        ntWindow().
        seleniumAJAXTesterCache[0];

        //  Remove  the retrieved
        object
        //  from the AJAX data cache
      selenium.browserbot.getCurren
      tWindow().
      seleniumAJAXTesterCache.shift();

        //  Parse the retrieved
        object
      var retrievedObject
      =  JSON.
      parse(retrievedObject
      AsString);

    //  Serialize the template...
    var stringifiedObject1  =
      JSON.stringify(templateObject);
    //  ...and  retrieved object
    var stringifiedObject2
      =
      JSON.stringify(retrie
      vedObject);

    //check objects for equality
    Assert.matches(
      stringifiedObject1,
      stringifiedObject2);
};
```

Fig. 2 shows the test case as it is shown within the Selenium IDE. First, the data gathering script is injected in the web page by calling the "injectAJAXCallLogger" user extension. Then an AJAX call is triggered (in the demo by virtually clicking on a button with id = btnSync. As the call is asynchronous, we have to have for it to complete. In the current version, this is simply achieved with a timer, more sophisticated version can perhaps directly monitor the browser's AJAX engine. The condition that is waited for (not completely visible in the picture) is: selenium.browserbot.getCurrentWindow().$.active ==0

Command	Target	Value
injectAJAXCallLogger		
click	//*[@id="btnSync"]	
waitForCondition	selenium.browserbo..	5000
assertAjaxCall		{"a":1,"b":2}

Fig. 2. The Selenium test case

Finally, the retrieved data is compared against a template object passed to the validation code by calling "assertAjaxCall". An overview of all control and data flows mentioned in this section is depicted in Fig 3.

C. *Example Web Pages under Test*

When testing web pages, one essential distinction must be made: is the page requesting data via AJAX from the server it was load or from a different domain. According to the situation, different data gathering methods must be used.

1) Same origin AJAX request: The JavaScript snippet that requests data from a web server can simply use jQuery's $.ajax() functionality with default parameters:

```
function
testSameDomainAjax()  {
  $.ajax("data.php",   {
    success:
       function(data,
       textStatus,
       jqXHR) {
      data  =  JSON.parse(data);
      //work with the data
      $("#a").text(data.a);
      $("#b").text(data.b);
    }
   });
}
```

The PHP script, that is queried for data is the following:

```
<?php
  $arr  =  array  ('a'=>1,'b'=>2);
  echo json_encode($arr);
?>
```

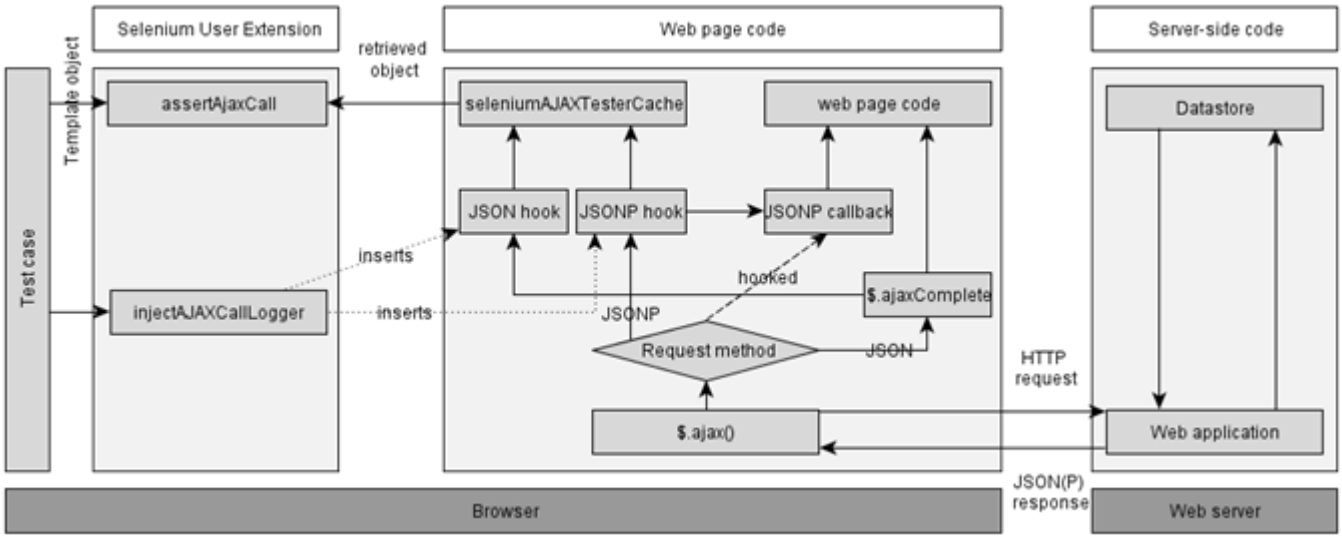

Fig. 3. Control Flow and Data Flow

2) Cross domain AJAX request Requesting data from a different domain requires more effort. A callback function has to specify that it will be called when the data has been retrieved.

```
function   testGeonames()  {
$.ajax("http://api.geona
mes.org/"  +
"countrySubdivisionJSON?"  +
"lat=33.536511&"   +
"lng=-
4.746094&username=...",
{
dataType:   'jsonp',
success:
function(data,
  textStatus,  jqXHR){

//work  with  the
data
  $("#countryName").text
(
data.countryName);
$("#countryCode").text

(data.countryCode);
}
});
}
```

IV. CONCLUSION

Software testing is a crucial element of the software development process. Testing is an intellectually challenging activity that needs a lot of planning before we start testing. It occurs near the end of the software development life cycle. Thus, it is often rushed and frequently not done well. In addition, it is a very costly and time consuming activity.

Web applications are becoming the most desired way to do shopping. So, in order to satisfy the customer with an error free application, it is fairly important to perform integration testing as this will remove the bad experience from customers. Many effort and experiences are required to understand the testing techniques to develop a web application. Therefore, a significant growth of the demand of Web-based applications has been recorded and documented, especially to serve business purposes.

In this work, we presented a neat plug-in for Selenium to hook every AJAX call made by the web application under test to verify requested data before it is processed by the application. This helps to narrow down the location of a fault (whether it is situated in server- or client-side code). This is especially important for agile teams that work on both client and server- side code in parallel.

In a nutshell, our approach helps these teams fulfilling the requirement of continuous integration testing their artifact. Beyond that, the developed approach can be used to for regression testing, too. For example, every new product version can be tested if its output still complies with the specification and the API documentation (e.g. regression testing). Furthermore, the front-end can directly work with a live back- end system without the need to develop mock objects, as the produced data is verified before it is fed to the JavaScript code reducing the effort required set up an application's testing environment.

V. FUTURE WORK

Our prototype currently can only test for the identity of two objects. It would be a valuable improvement if we could test the data for compliance to a range of constraints (data type, value range) instead of just the simple identity. An example for JSON data, that cannot be verified yet, is a call to the Twitter API. Every result returns data that is unique to the request. Therefore, the static matching cannot be used and a more sophisticated technique must be developed.

```
$.ajax("http://search.
  twitter.com/" +
  "search.json?callback=?&
  rpp=5&q="  +
  "from:ladygaga",
  {
    dataType:
    'jsonp',
    success:
    function(data
    ,
       textStatus, jqXHR) {
     //work with the data
    }
  }
);
```

By adding the ability to compare the objects using regular expressions, changing parameters in every request can be filtered out or validated. The most sophisticated approach on top of the current work would be to specify not a template object in the test case but metadata that describe the attributes of the server's response (like the count, the type and the names of the expected parameters allowing in-depth inspection and validation of the data. This can parse all kinds of responses where regular expressions are not powerful enough.

Another way to improve our work is through testing our technique on real-time search engines in order to evaluate its usefulness in the assessment of the site quality. Our technique is expected to allow a deep insight in the internal functioning of the Web applications, as well as evaluating the accuracy of the results received by the end-users. Conducting such experiments would be a great improvement to our work, since it will allow not only testing the effectiveness of our technique, but also how complementary it is to all other existing techniques.

REFERENCES

[1] G. A. Di Lucca and A. R. Fasolino. (2006, Dec.) Testing web-based applications: The state of the art and future trends. [Online]. Available: www.sciencedirect.com

[2] Selenium. (2012) What is Selenium? [Online]. Available: http: //seleniumhq.org/

[3] S. Bergmann. (2012, Dec.) PHPUnit Manual. [Online]. Available: http://www.phpunit.de/manual/current/en/phpunit-book.pdf

[4] J. Schaible. (2007, Dec.) JsUnit Manual. [Online]. Available: http://jsunit.berlios.de/index.html

[5] Refsnes Data. (2012) AJAX Tutorial. [Online]. Available: http://www.w3schools.com/ajax/default.asp

[6] (2012) JSON Tutorial. [Online]. Available: http://www.w3schools.com/json/default.asp

[7] Mozilla Developer Network. (2012, Aug.) JSON. [Online]. Available: https://developer.mozilla.org/en-US/docs/JSON

[8] jQuery Foundation. (2012) jQuery is a new kind of JavaScript Library. [Online]. Available: http://jquery.com/

[9] Selenium. (2012) User-Extensions. [Online]. Available: http://seleniumhq.org/docs/08 user extensions.html

[10] Mozilla Developer Network. (2011, Nov.) Same origin policy for JavaScript. [Online]. Available: https://developer.mozilla.org/en-US/ docs/Same origin policy for JavaScript

[11] J. Blatz. (2007, Dec.) CSRF: Attack and Defense. [Online]. Available: http://www.mcafee.com/us/resources/white-papers/ wp-csrf-attack-defense.pdf

[12] K. Simpson. (2012, Nov.) Defining Safer JSON-P. [Online]. Available: http://json-p.org/

Dr. Bouchaib Falah: Offering more than 20 years of combined experience developing and implementing computer science and technical/math curriculum for different colleges and universities as well as web designs for multimillion-dollar organizations in USA and researcher in different projects, Dr. Bouchaib Falah is currently an Assistant Professor at Al Akhawayn University, teaching graduate and undergraduate software engineering courses, School of Science and Engineering. Beside teaching high school level math in Morocco and college mathematics and computer science at Harrisburg Area Community College in Pennsylvania, Suny Orange Community College in New York, Pennsylvania State University in Pennsylvania, Central Pennsylvania College in Pennsylvania, Concordia College in Minnesota, and North Dakota State University in North Dakota, Dr; Bouchaib Falah has an extensive industrial experience with Agri-ImaGIS, Synertich, and Commonwealth of Pennsylvania Department of Environmental Protection. He holds a doctoral degree in Software Engineering from North Dakota State University, in 2011, a master degree in Computer Science from Shippensburg University, in 2001, and a bachelor degree/teaching certificate from Ecole Normale Superiere in Morocco, in 1990.