**Introducing to Selenium**

History of Selenium:

**Selenium 1 or Selenium Remote Control or Selenium RC**

Selenium RC is a popular UI automation library, allowing developers and testers to automate their interactions with a Web Application Under Test (WAUT) by providing them with the necessary libraries, supported in multiple languages, to program.

In terms of design, Selenium RC chose to use generic JavaScript named Selenium Core to drive the WAUT on a browser. However, the decision of using generic JavaScript that can drive the WAUT on any browser should comply with a security policy named Same-Origin Policy. Every available browser in the market imposes this policy on the websites that are loaded on it.

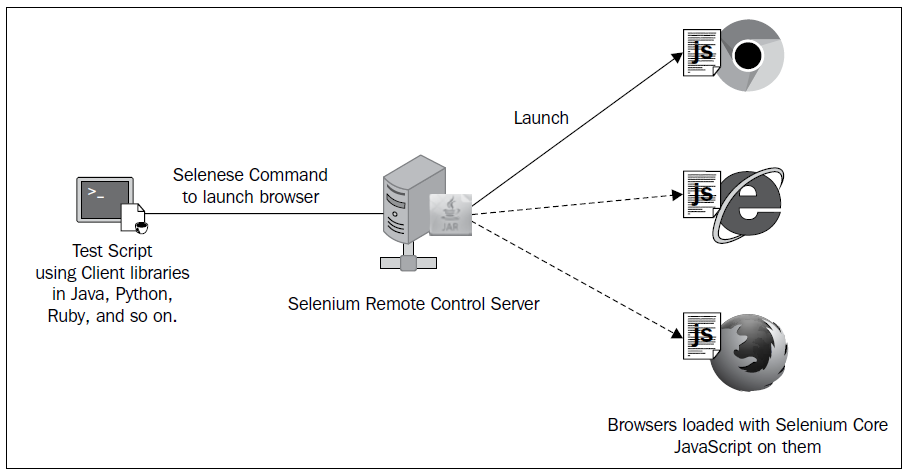
For every website that is loaded on it, the browser creates a separate sandbox for the website’s JavaScript, which restricts the JavaScript to be executed only on it’s respective website domain.

This way, a JavaScript that belongs to one website doesn't execute on another website that is currently loaded on that browser. This security vulnerability, named Cross-site scripting, is the browser's responsibility to restrict. So, coming back to Selenium RC, its generic JavaScript is not allowed, by the browser, to execute on a website (WAUT) that is coming from a different domain.

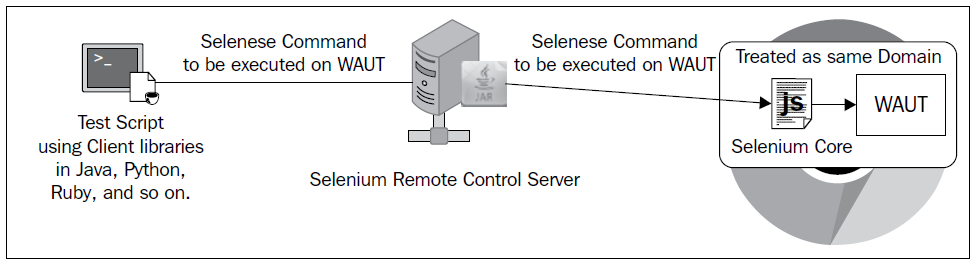
Selenium RC acts as an HTTP Proxy Server. When the test script asks to launcha browser, Selenium RC server launches the browser and injects its JavaScript (Selenium Core) into the browser. All the subsequent requests for the WAUT go through Selenium RC (acting as an HTTP Proxy Server) to the actual web server hosting WAUT. Thus making the browser think that the web application is being served from the Selenium RC's server domain than the actual web server's domain and allowing Selenium Core to execute and drive the web application.

Typically, it works in the following way:

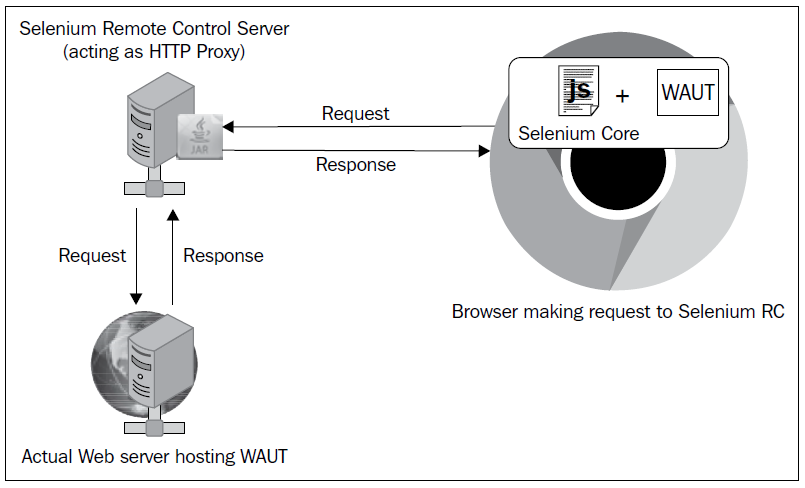
1. A tester or a developer, through his/her test script, can command Selenium RC server to perform certain actions on the WAUT on a certain browser. The way the user can command Selenium RC to perform something is by using the client libraries provided by Selenium RC. These libraries are provided in different languages, such as Java, Ruby, Python, Perl, PHP, and .NET. These commands, which are passed from the test scripts to Selenium RC, are named **Selenese** commands. In a test script, you will have a set of Selenese commands to test a scenario on the WAUT.
2. Once the Selenium RC server receives the command from the test script, it will launch the test script preferred browser, and while launching, it injects the Selenium Core into the browser.



1. Upon loading on the browser, Selenium Core executes all the Selenese commands from the test script, coming through Selenium RC, against the WAUT. The browser doesn't restrict it, because it treats Selenium Core and WAUT as a part of the same domain.



1. Now comes the HTTP Proxy part of the Selenium RC server. All the requests and responses of the browser for WAUT go to the actual web server via Selenium RC server, because the browser thinks Selenium RC is serving WAUT.



1. After execution, Selenium RC will send out the test result back to the test script for developer's analysis.

**Selenium 2 or Selenium WebDriver or WebDriver**

Following are the reason WebDriver is popular:

* To give a better control on the browser by implementing browser-specific implementations.

**How Selenium WebDriver works:**

* A tester or developer, through his/her test script, can command WebDriver to perform certain actions on the WAUT on a certain browser. The way the user can command WebDriver to perform something is by using the client libraries or language bindings provided by WebDriver. These libraries are provided in different languages, such as Java, Ruby, Python, Perl, PHP, and .NET.
* By using the language-binding client libraries, developers can invoke the browser-specific implementations of WebDriver, such as Firefox Driver, IE Driver, Opera Driver, and so on, to interact with the WAUT on the respective browser. These browser-specific implementations of WebDriver will work with the browser natively and execute commands from outside the browser to simulate exactly how the application user does.
* After execution, WebDriver will send out the test result back to the test script for developer's analysis

