Level 1:

The vulnerable source code is:

```
 def get(self):
   # Disable the reflected XSS filter for demonstration purposes
   self.response.headers.add_header("X-XSS-Protection", "0")

   if not self.request.get('query'):
     # Show main search page
     self.render_string(page_header + main_page_markup + page_footer)
   else:
     query = self.request.get('query', '[empty]')

     # Our search engine broke, we found no results :-(
     message = "Sorry, no results were found for <b>" + query + "</b>."
     message += " <a href='?'>Try again</a>."

     # Display the results page
     self.render_string(page_header + message + page_footer)
   return
```

This is triggered because the user input from the “query” parameter is directly embedded into the HTML, I inserted <script> alert(“”) </script>.

Level 2:

The vulnerable source code is:

```
function displayPosts() {
        var containerEl = document.getElementById("post-container");
        containerEl.innerHTML = "";

        var posts = DB.getPosts();
        for (var i=0; i<posts.length; i++) {
          var html = '<table class="message"> <tr> <td valign=top> '
            + '<img src="/static/level2_icon.png"> </td> <td valign=top '
            + ' class="message-container"> <div class="shim"></div>';

          html += '<b>You</b>';
          html += '<span class="date">' + new Date(posts[i].date) + '</span>';
          html += "<blockquote>" + posts[i].message + "</blockquote";
          html += "</td></tr></table>"
          containerEl.innerHTML += html;
        }
      }
```

The innerHTML used bans the keyword “script”, but one can still execute code by inputting <img src=“” onerror=alert(“”)> in the message box.

Level 3:

The vulnerable source code is:

```
function chooseTab(num) {
        // Dynamically load the appropriate image.
        var html = "Image " + parseInt(num) + "<br>";
        html += "<img src='/static/level3/cloud" + num + ".jpg' />";
        $('#tabContent').html(html)
```

We can close the <img src='/static/level3/cloud" + num + ".jpg' />" and add the script we need to execute by changing the url to https://xss-game.appspot.com/level3/frame#'/><script>alert(1)</script>

Level 4:

The vulnerable source code is:

```
<img src="/static/loading.gif" onload="startTimer('3');" />
```

The user input is not sanitized, and we can input 3');alert('1 in the timer box to first close the onload function and execute our code.

Level 5:

The vulnerable source code is:

```
<a href="confirm">Next &gt;&gt;</a>
```

After we click Signup,
The Next button is linked to the confirm page, and we can modify the url to https://xss-game.appspot.com/level5/frame/signup?next=javascript:alert(), and then click on next to execute the JavaScript code.

Level 6:

The vulnerable source code is:

```
// Load this awesome gadget
scriptEL.src = url;
```

Although the website uses regular expressions to check the url, we can still use an external file that contains desired js code to execute an alert. I inputted this into the URL box https://xss-game.appspot.com/level6/frame#//google.com/jsapi?callback=alert.