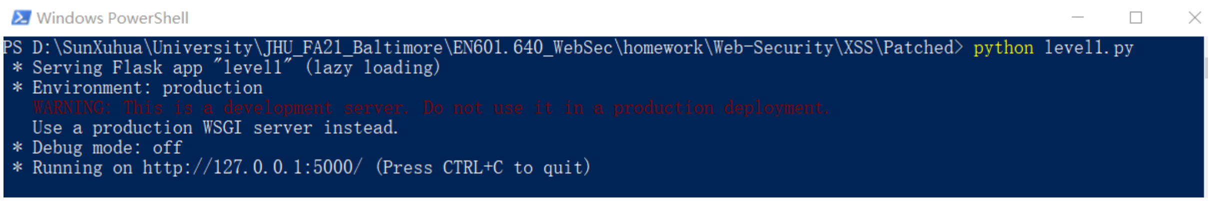
XSS Exploit Rebuilding and Patching

Overview:

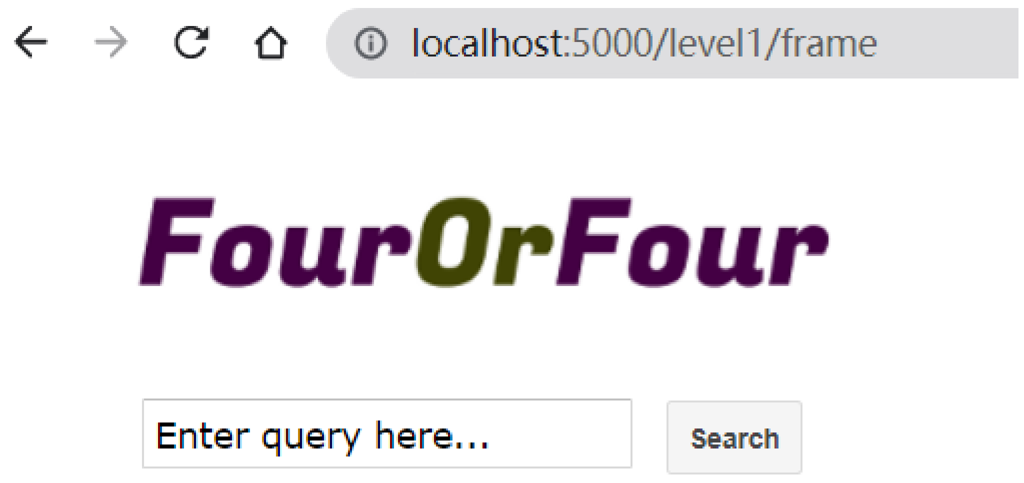
In the following report, we will add patches for every level to prevent the attacks we have used in part1.

For running any level, we can simply use "python level[index].py" in the shell to run and then check the localhost:5000, the content should be the same to what we have seen in the XSS game.

For example, when we want to see level1, we first go to the directory of level1.py and use the shell to run "python level1.py". We then check the localhost:5000 and it will look like the following.



```
PS D:\SunXuhua\University\JHU_FA21_Baltimore\EN601.640_WebSec\homework\Web-Security\XSS\Patched> python level1.py
 * Serving Flask app "level1" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

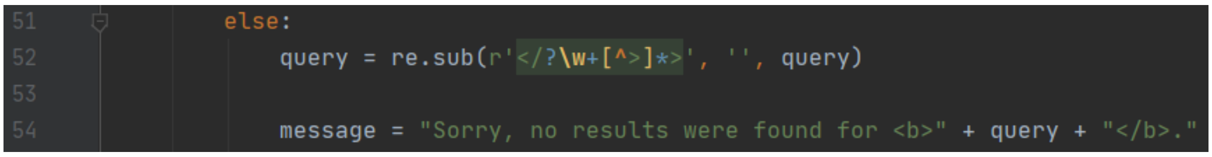


For the following, we will introduce how we patch every level to prevent attacks.

Level 1: Hello, world of XSS

In “level1.py”, we simply add a substitution based on the regular expression after we receives the query to filter the input containing HTML tags.

This can be useful when a user tries to input content including “<script>” and therefore prevents the attack.

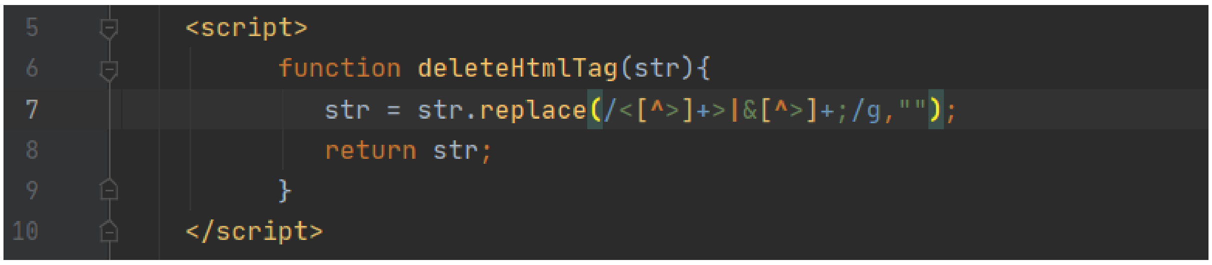


```
    else:
        query = re.sub(r'</?\w+[^>]*>', '', query)

        message = "Sorry, no results were found for <b>" + query + "</b>."
```

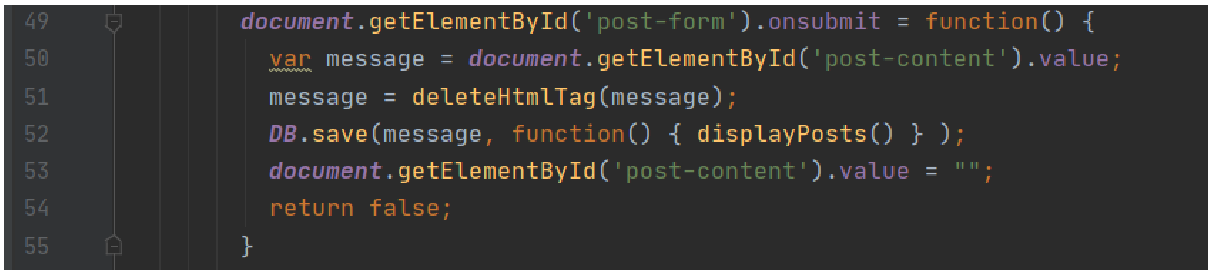
Level 2: Persistence is key

Likewise, we use the regular expression here to filter the input and apply it to the message we have received. Therefore, any possible HTML tag, including the image, will be eliminated at this stage.

We patch this in “level2_index.html”.



```
<script>
    function deleteHtmlTag(str){
        str = str.replace(/<[^>]+>|&[^>]+;/g,"");
        return str;
    }
</script>
```

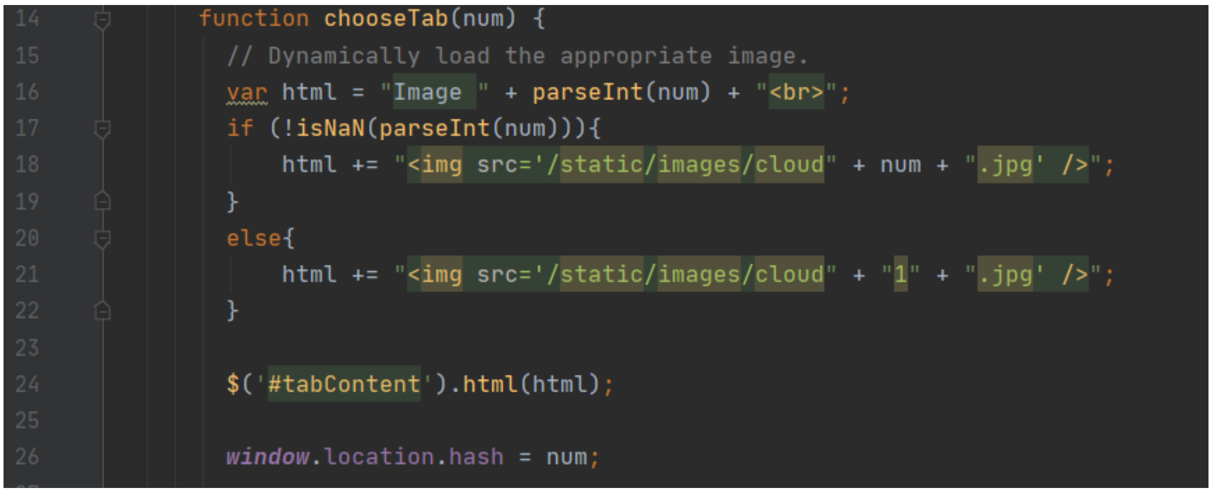


```
document.getElementById('post-form').onsubmit = function() {
  var message = document.getElementById('post-content').value;
  message = deleteHtmlTag(message);
  DB.save(message, function() { displayPosts() } );
  document.getElementById('post-content').value = "";
  return false;
}
```

Level 3: The sinking feeling

Since what we need is only numbers here, we use the if-condition to judge whether the input is a number. If not, we do not accept that and always display the first picture as default.

We patch this in “level3_index.html”.

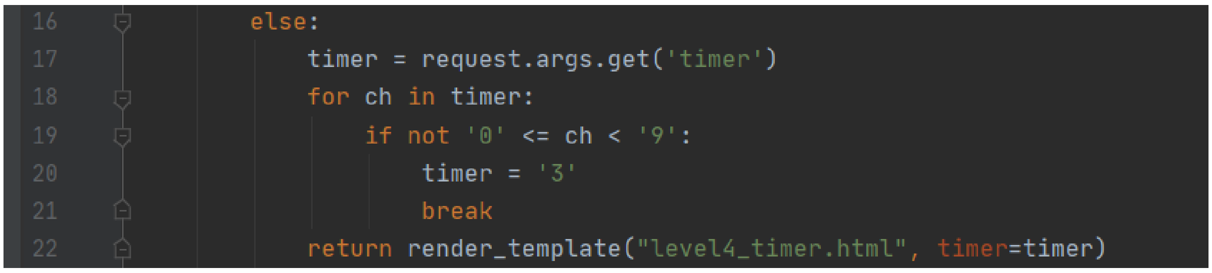


```
function chooseTab(num) {
  // Dynamically load the appropriate image.
  var html = "Image " + parseInt(num) + "<br>";
  if (!isNaN(parseInt(num))){
      html += "<img src='/static/images/cloud" + num + ".jpg' />";
  }
  else{
      html += "<img src='/static/images/cloud" + "1" + ".jpg' />";
  }

  $('#tabContent').html(html);

  window.location.hash = num;
```

Level 4: Context matters

Likewise, since we only need number input for the seconds variable, we add the if-condition to judge the input. We do not accept any input other than numbers and will treat it as the default 3 seconds.

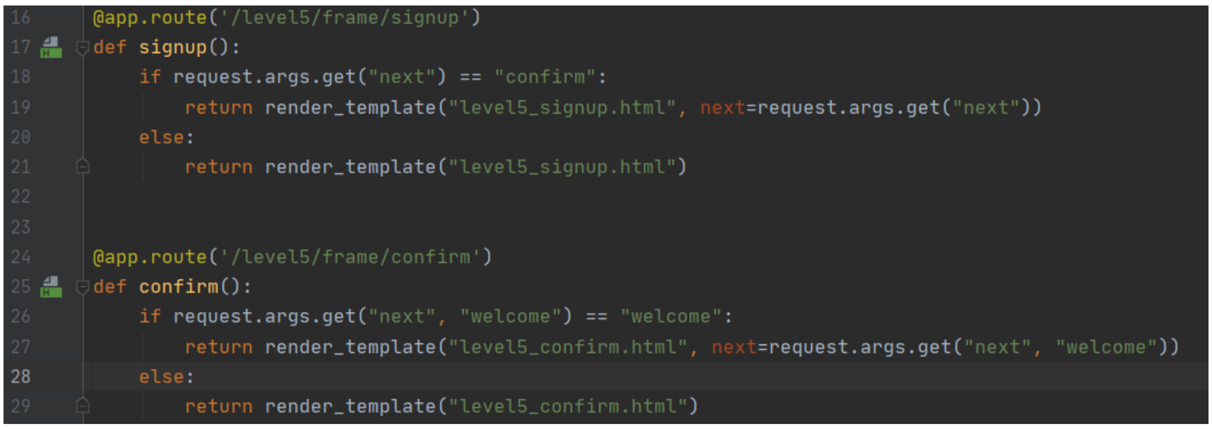
This time, we do this in the backend "level4.py"



```
    else:
        timer = request.args.get('timer')
        for ch in timer:
            if not '0' <= ch < '9':
                timer = '3'
                break
        return render_template("level4_timer.html", timer=timer)
```

Level 5: Breaking protocol

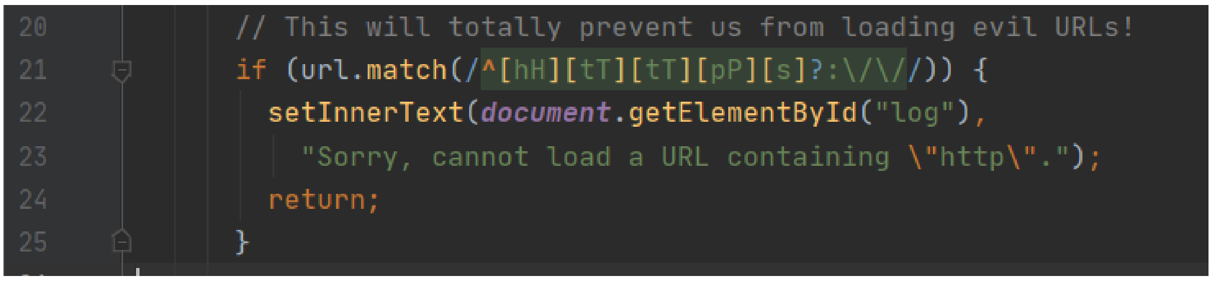
In "level5.py", we add if-conditions to judge whether the parameter of "next" is legal. In this case, only keywords like confirm, welcome are legal. Therefore, we banned the input of Javascript code here.



```
@app.route('/level5/frame/signup')
def signup():
    if request.args.get("next") == "confirm":
        return render_template("level5_signup.html", next=request.args.get("next"))
    else:
        return render_template("level5_signup.html")


@app.route('/level5/frame/confirm')
def confirm():
    if request.args.get("next", "welcome") == "welcome":
        return render_template("level5_confirm.html", next=request.args.get("next", "welcome"))
    else:
        return render_template("level5_confirm.html")
```

Level 6: Follow the rabbit

In “level6_index.html”, we extend the filter into a more sensitive way that it can detect any combination of http or https, regardless of the uppercase or lowercase. Therefore, we can prevent the using of such protocol even if the browser will translate uppercase letters into lowercase ones.



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