Chapter 10: Cross-Site Scripting Attack

Problems

10.1. Using LiveHTTPHeader, we find out that the following GET request is used to send an HTTP request to `www.example.com` to delete a page owned by a user (only the owner of a page can delete the page).

```
http://www.example.com/delete.php?pageid=5

GET /delete.php?pageid=5
Host: www.example.com
...
```

Please write a malicious JavaScript program, which can delete a page owned by the victim if the program is injected into one of the victim's page from `www.example.com`.

10.2. Using LiveHTTPHeader, we find out that the following POST request is used to send an HTTP request to `www.example.com` to delete a page owned by a user (only the owner of a page can delete the page).

```
http://www.example.com/delete.php

POST /delete.php HTTP/1.1
Host: www.example.com
...
Content-Length: 8
pageid=5
```

Please write a malicious JavaScript program, which can delete a page owned by the victim if the program is injected into one of the victim's page from `www.example.com`.

10.3. In Listing 10.2, we added a check before sending the Ajax request to modify Samy's own profile. What is the main purpose of this check? If we do not add this check, can the attack be successful? How come we do not have such a check in the add-friend attack (Listing 10.1)?

10.4. To defeat XSS attacks, a developer decides to implement filtering on the browser side. Basically, the developer plans to add JavaScript code on each page, so before data are sent to the server, it filters out any JavaScript code contained inside the data. Let's assume that the filtering logic can be made perfect. Can this approach prevent XSS attacks?

10.5. What are the differences between XSS and CSRF attacks?

10.6. Can the secret token countermeasure be used to defeat XSS attacks?

10.7. Can the same-site cookie countermeasure for CSRF attacks be used to defeat XSS attacks?

10.8. To filter out JavaScript code from user input, can we just look for `script` tags, and remove them?

10.9. ★★
If you can modify browser's behavior, what would you add to browser, so you can help reduce the risks of XSS attacks?

10.10. ★★★
There are two typical ways for a program to produce a copy of itself. One way is to get a copy of itself from outside, such as from the underlying system (e.g., files, DOM nodes) and from the network. Another way is not to use any help from outside, but instead generate a copy of itself entirely from the code. There is a name for this approach: it is called a `quine` program, which, according to Wikipedia, "is a non-empty computer program which takes no input and produces a copy of its own source code as its only output. The standard terms for these programs in the computability theory and computer science literature are *self-replicating programs*, *self-reproducing programs*, and *self-copying programs*." The self-replicating JavaScript program shown in Listing 10.3 is not a quine, because it uses `document.getElementById()` to take an input from the underlying system.

Please write a quine program, and put it in a user's profile in `Elgg`. When anybody visits this profile, the code will be executed, and it prints out a copy of itself in an alert window. The Wikipedia site has examples of quine programs in a variety of programming languages.

If you really want to challenge yourself, please rewrite the code in Listing 10.3, so it is a quine program, and it can do what exactly the code in Listing 10.3 can do, i.e., adding a statement and a copy of the worm to the victim's profile.

10.11. The fundamental cause of XSS vulnerabilities is that HTML allows JavaScript code to be mixed with data. From the security perspective, mixing code with data is very dangerous. XSS gives us an example. Please provide two other examples that can be used to demonstrate that mixing cod with data is bad for security.