Background :

The purpose of this lab is to give you experience with the basic mechanisms of HTML/JavaScript injection underpinning cross-site scripting (XSS) attacks. In this lab you'll provide inputs via an html form to a server-side PHP script that constructs a web page using the provided information. However this script (deliberately) doesn’t do a good job of safely using the user-provided inputs, so it is potentially vulnerable to reflected XSS. You'll get to try this out by providing inputs that cause your JavaScript code to be executed when you view the generated web page.

Benign usage :

The form that is the user interface for creating the web page is hosted next to these instructions at:

https://www-users.cselabs.umn.edu/classes/Fall-2022/csci4271/index.php?page=./labs/08/fruit-form

You may want to open it in a separate tab to experiment with it while also reading these instructions. The five input boxes are filled in with default values which you can see are mostly the names of fruits. If you just submit the form without changing the defaults, you'll see it create a simple HTML page which incorporates the inputs, some in more visible ways than others. Try using the "View Page Source" feature of your web browser to see what the generated HTML looks like: it should be clear where each form input went.
The small program that implements the form and page generation is written in the language PHP. The web server won't show you the source code if you try to access it by URL (instead it executes the script), but you can see the source

code from a CSE Labs machine as the following file:
/web/classes/Fall-2022/csci4271/labs/08/action.php

You can see that PHP is a template-style language: the outer structure of the document is the same as the structure of the final HTML page. But portions enclosed in <?php and ?> contain PHP code. The values entered in the form show up in PHP variables with names like $_GET['fruit']. Some of the values are copied into the HTML page without changes, while some are transformed in an attempt to sanitize them.

You can also confirm how this works by changing the contents of the form and resubmitting it. Try changing the names of fruits, or see if you can think of another color whose name starts with "c" that is recognized in CSS.

Cross Site Scripting (XSS) attscks :

When the goal is just to verify the presence of an XSS vulnerability, rather than maliciously exploit it, the standard "payload" is just JavaScript that opens a popup box with a message; this is similar to overwriting a return address with AAAAAAAA for a buffer overflow or opening a calculator. Different attacks are possible against all five inputs to the script, but for instance to get started with the attack using input a, try to get the browser to execute the following JavaScript code:

alert("XSS a”)

Input a doesn't have any additional defenses or complexities, so all you need is to know the standard way to insert JavaScript code in HTML. You can of course find lots of HTML documentation on the web.

Inputs b and c go in other contexts inside the web page, so your attack will need to be modified accordingly. In these cases it is enough to use a standard injection attack strategy of finishing off the construct that you were inside, then putting in new top-level content. You can also make sure things are more consistent by restarting another of the construct you were already in, though web browsers are forgiving enough that this is rarely needed.
Inputs d and e are more a bit more complex in that the script tries to sanitize them by removing things that you might try to use in your attack. However sanitization to completely block XSS is difficult, and the script doesn't do a good job at all, so you can get around it. You can see from the PHP code that the page is trying to remove things that would be used in an attack via string-replacement operators. There are also a lot of resources on the web about strategies for XSS attacks in the presence of filters; two of our favorites are

this older one from OWASP

another more recent one.

You might also find it helpful to know that all three kinds of ASCII quote marks can be used in JavaScript.