CSCI 3330 Assignment 5

Deadline: October 28, 2025, 10:00pm

1 Background

In this assignment, you will be developing and executing XSS attacks on Web sites under real-world conditions. Specifically, by completing this assignment, you will:

- Craft and test XSS payloads under different filtering conditions, iteratively minimizing proofs of concept to isolate exact blocking rules or bypasses.
- Evaluate the effectiveness and limitations of regex-based XSS filters.
- Propose strategies to mitigate XSS vulnerabilities comprehensively.

2 AI policy

You may use AI to learn background concepts. You may not use AI to generate, verify, or iterate on solutions for this assignment.

Allowed:

- Concept definitions and explanations: "What is a XSS attack?", "What does the alert() function do in JavaScript?", "What is the onloadstart HTML attribute¿', "What does the body tag do in HTML?"
- Generic examples not derived from this assignment: "Walk through an example of a XSS attack," "Walk through an example of calling the function `alert('xss')` from an HTML page using JavaScript," "Walk through an example of reverse engineering a comment filter that triggers an HTML block page."
- Non-generative tooling: man pages, official documentation, calculators, programming languages, and trying something for yourself.

Handle with care:

- Tools which improve grammar or clarity.
- IDE autocompletion or documentation lookup that does not inject solution content.

Figure 1: An example `alert('xss')` box.

Not allowed:

- Any hint, outline, or solution for these specific questions (including rephrased or "similar" versions), e.g., "Help me with the attack in Q2," "Write a XSS attack to call `alert('xss')`," or "Check my answers."
- Pasting assignment text or numbers into AI to tailor examples.
- Using AI to generate any text, code, calculations, or any other artifacts that you submit.

Ask me questions instead! When in doubt, don't use AI, because **you won't have access to it on tests**, for which these assignments are your preparation.

3 Attacks

In this section, you will implement XSS attacks under a variety of conditions. You are encouraged to consult the XSS Cheat Sheet. Pay special attention to the examples featuring some variation of `alert('xss')`, as this is what you will try to execute in your attacks. In the following attacks, you are ***NOT*** allowed to load resources from external servers (e.g., `src="http://barf.com/bletch.js"` since it generates a network request to barf.com, an external server), so you may wish to skip over trying those examples in the cheat sheet unless you are interested in learning more about them.

1. Execute a XSS attack on this comment submission form that calls the JavaScript code `alert('xss')`. When successful, your browser should display an alert box such as the one in Figure 1. Since this form is not a real comment submission form, you may only preview your comment, and you cannot actually post it anywhere and attack anyone else with it. However, you can preview your comment to verify whether the attack would be effective (i.e., whether `alert('xss')` runs when the preview renders). This submission form performs no sanitization or escaping. What comment did you submit to execute the attack?

2. Execute a XSS attack on this comment submission form that effectively calls the JavaScript code `alert('xss')`. This submission form is subject to a common Web Application Firewall (WAF)'s default filters.

   (a) What is a comment that successfully executes the attack on the Q1 submission form but is blocked on this one?

   (b) What kinds of strings do this filter's rules appear to be targeting? Do all of the rules make sense in the context of blocking malicious attacks? **Hint:** to learn more about what is triggering a blocked attack, try removing characters in the blocked attack until it is no longer blocked (even if it no longer triggers XSS). By removing characters until you find a minimal example that reproduces a block message, you have identified a rule that was targeting your attack. However, keep in mind that there may be *multiple* rules preventing your attack at any given time, and so you may need to repeat this process multiple times for an attack to identify all of the rules that may be targeting it if you find that removing the offending string does not prevent your attack from being blocked.

   (c) What is a comment that successfully executes the attack on this submission form? **Hint:** one way that I found to execute JavaScript was to use the `onloadstart="..."` event attribute (which is a newer HTML event attribute that may not have existed or been popular when the WAF's filter was devised), as other event attributes such as `onload` appear to be blocked. But, if you wish to pursue the same path, to get this to work, you may need to do some background research on what HTML element you can assign this attribute to and what other attribute(s) you need to assign to this element for this event to trigger. Furthermore, normally you would put your JavaScript payload verbatim into the value of this attribute, but you may find that you have to discover creative ways of calling `alert()` in JavaScript due to additional filtering that is being applied.

3. Execute a XSS attack on this comment submission form that effectively calls the JavaScript code `alert('xss')`. This submission form is subject to the exact same XSS filter that my learning management system used when I was a student.

   (a) What is a comment that successfully executed the attack on the Q1 submission form but is blocked on this one?

   (b) What is a comment that successfully executes the attack on this submission form? (**Hint:** consult the XSS Cheat Sheet.)

   (c) Thus far in this assignment, we have used `alert('xss')` as a benign proof-of-concept of arbitrary JavaScript code execution. If your learning management system (Canvas) used this XSS filter, how could you weaponize your payload to change your grades? For example,

what JavaScript would you want to include in a message to your instructor? Provide a brief outline in a few sentences.

4. Execute a XSS attack on this comment submission form that effectively calls the JavaScript code `alert('xss')`. After I reported the vulnerability you exploited in the previous question, the vendor "fixed" the vulnerability by deploying the XSS filter featured in this problem.

   (a) What is a comment that successfully executes the attack on this submission form?

   (b) Did you have to modify your attack from the previous question, or did the same one work unmodified? (If the latter, you likely developed a different exploit than the one that I had originally developed.)

   (c) How do you think they attempted to fix the exploit given that it is still vulnerable? Will fixing the filter in this manner ever be adequate?

   (d) In a few sentences, what is the proper way to sanitize user input if we want to allow *some* HTML content?

4 Mid-class Survey

Complete the mid-class survey here. To preserve anonymity, do **not** email me your answers to this survey as part of your submission.

5 Submission

Email your answers to the questions in Section 3 in a single `*.txt` or `*.pdf` file attachment to my email address in the Syllabus.