**CCS4352 - Application Security**

**Lab 03 - Exploring Cross-Site Request Forgery (CSRF) Vulnerabilities with Python and Flask**

Name: S.P.A.S. Senarathne  
Registration Number: 22UG1-0345

**Answers to the lab questions:**

**Q1:** What happens when you perform a CSRF attack on the profile page, and why does it succeed before adding protection?  
**Answer:**  
The email address is changed without the user informed consent since it is automatically submitted through the ruse of the malicious page. The vulnerable app knew nothing about the origin of the request and missing among the parameters there was a CSRF token, that is why the request was treated as that of the legitimate form one.

**Q2:** How does the CSRF attack on the password change page work?  
**Answer:**  
The malicious HTML page POSTs a hidden form to the /change-password route with the victim still logged in. The browser automatically sends the session cookie with the request so that the server thinks that the authorized user was changing the password.

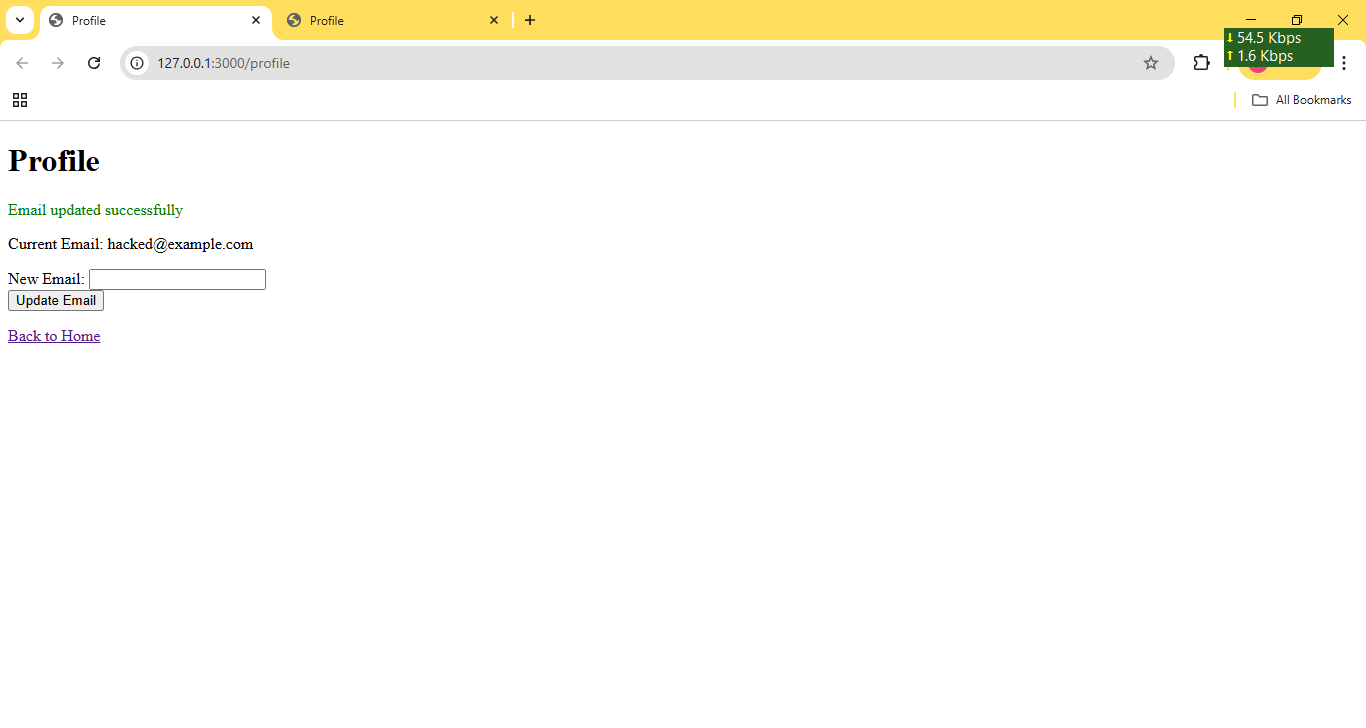
**Q3:** Why do CSRF tokens prevent these attacks?  
**Answer:**  
CSRF tokens are random values that are stored in the session of a user and are embedded in the forms. The attacker cannot guess this token and an incorrect token on any request is rejected because it is verified on the server.

**Q4:** What other security measures could make this app safer?  
**Answer:**

* Implement **SameSite cookies** to prevent cross-site requests from being sent automatically.
* Require **re-authentication** before sensitive actions (like password change).
* Use **CAPTCHAs** to prevent automated submissions.
* Enable **HTTPS** to protect against session hijacking.
* Perform **origin/referrer header checks** to ensure requests come from the same site.

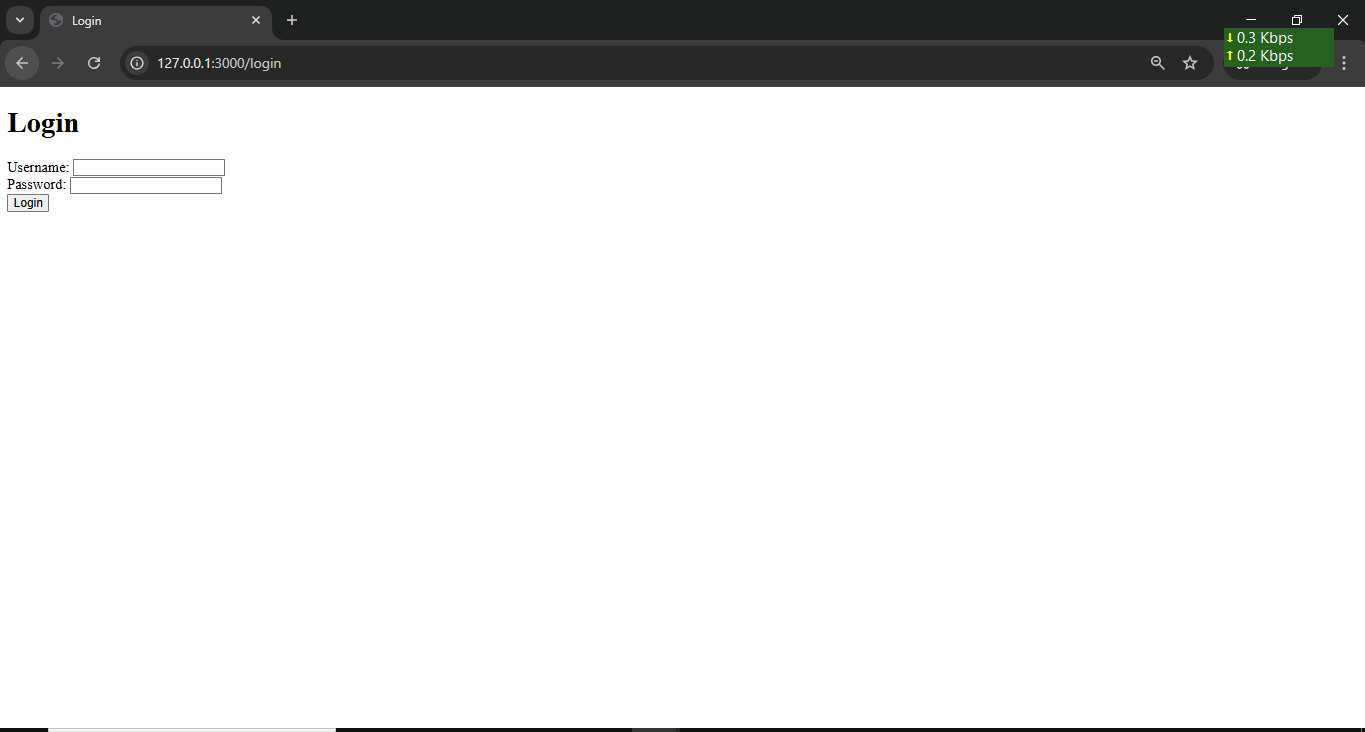
**Screenshots >>>>>>>>>>>>**

Successful CSRF attacks (before fixes):



Failed attacks after adding CSRF protection:

+ Redirect to login page when accessing exploit.html >>>



+ logged as admin >>>

