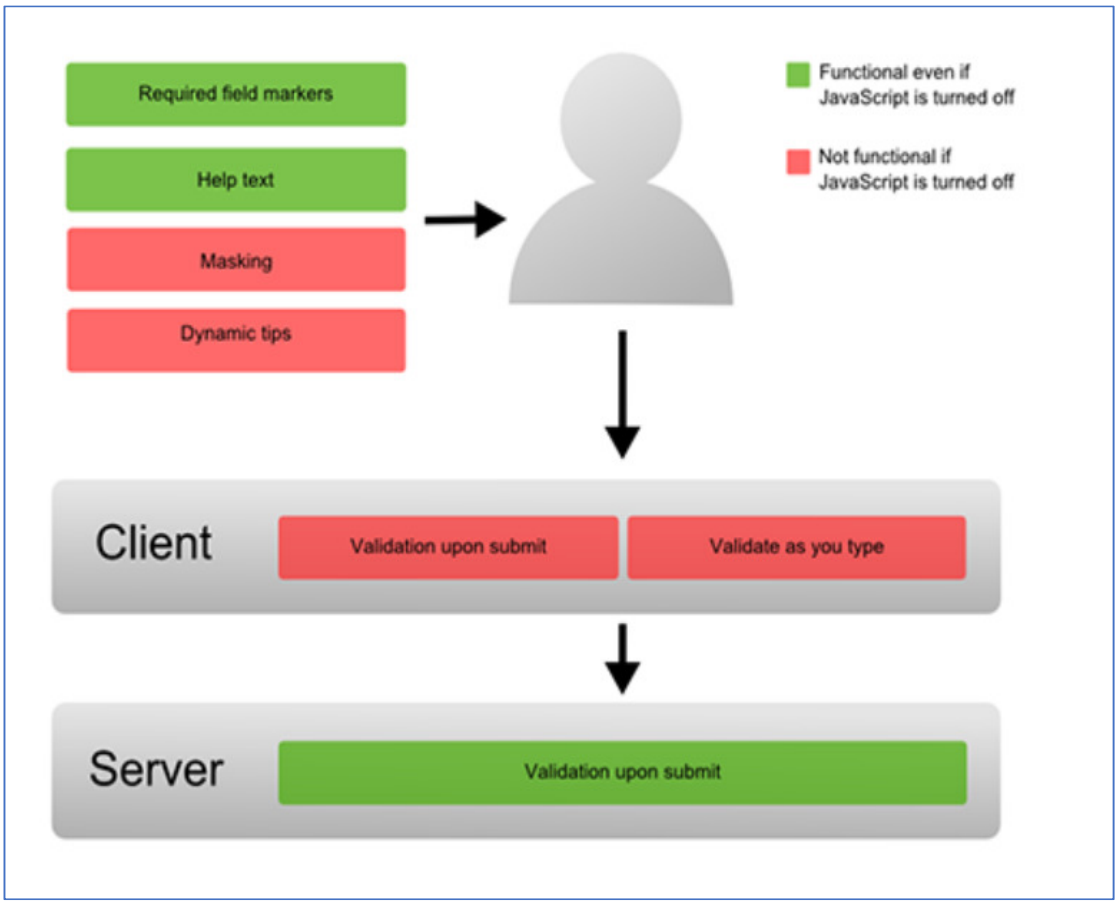
Importance of Server-Side Validations

This article details about the importance of server-side validations, its implementation, usage and how implementing only client-side validation could easily be bypassed by any malicious user/attacker.

It will assist the developers in understanding how actually the attackers look for the loopholes which they can exploit and get access of the Web application or Database.

What Server-Side Validation refers to?

"User's input can be validated on the server and on the client (web browser). Thus, we have server-side and client-side validation."



In the server-side validation,

1- Information is being sent to the server and validated using one of the server-side languages.
2- If the validation fails, the response is then sent back to the client, page that contains the web form is refreshed and a feedback is shown.
3- **This method is secure because it will work even if JavaScript is turned off in the browser and it can't be easily bypassed by malicious users.**
4- **But** complete server-side validation results in a slow response from the server.

✓ **For better user experience, however, you might consider using client-side validation.**

In the client-side validation,

1- This type of validation is done on the client using script languages such as JavaScript.
2- By using script languages, user's input can be validated as they type. **This means a more responsive, visually rich validation by the application.**
3- With client-side validation, form never gets submitted if validation fails.
4- Validation is being handled in JavaScript methods that you create (or within frameworks/plugins) and users get immediate feedback if validation fails.

Drawback of Client-Side Validations

Main drawback of client-side validations is,

1- It relies on JavaScript.
2- **If users turn JavaScript off, they can easily bypass the validation.**
3- Therefore, **validation should always be replicated and implemented on both the client and server.**

Practical Attack Scenarios and its Consequences

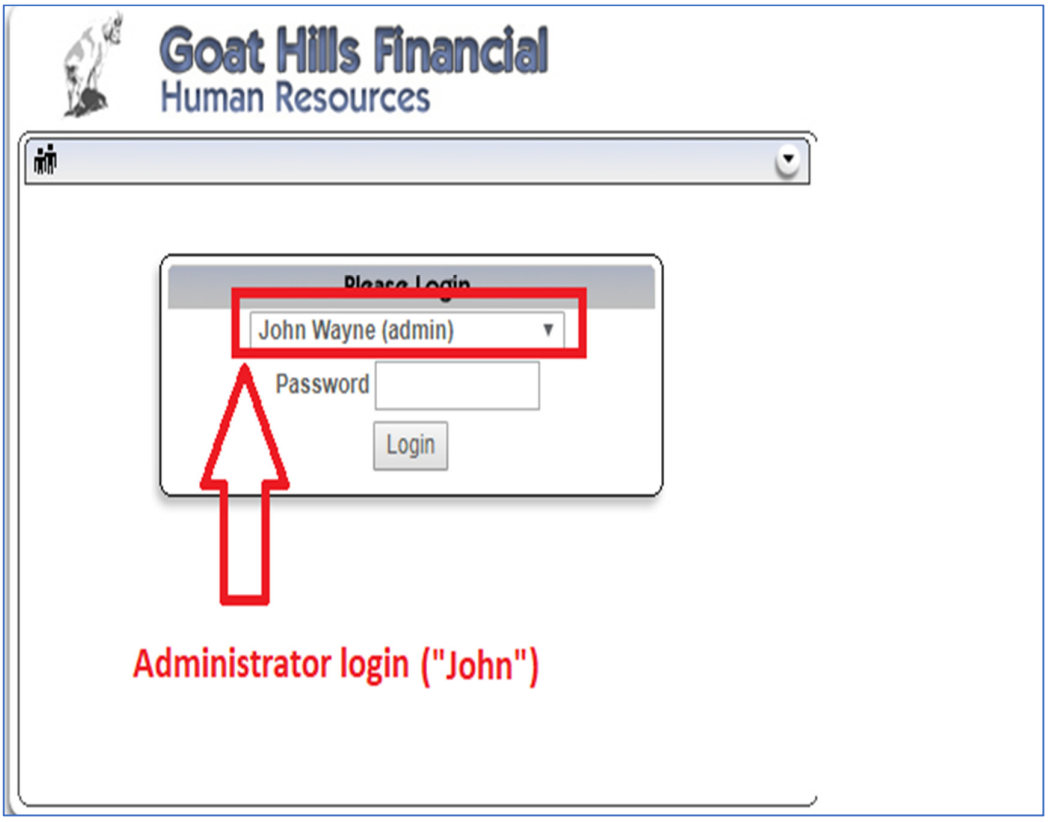
For demonstration purposes, we have taken **WebGoat** application, which is an intentionally vulnerable web app by **OWASP** (Open Web Application Security Project) to understand different security vulnerabilities and their implications.

Example-1: Bypassing Business layer Access Control Mechanisms

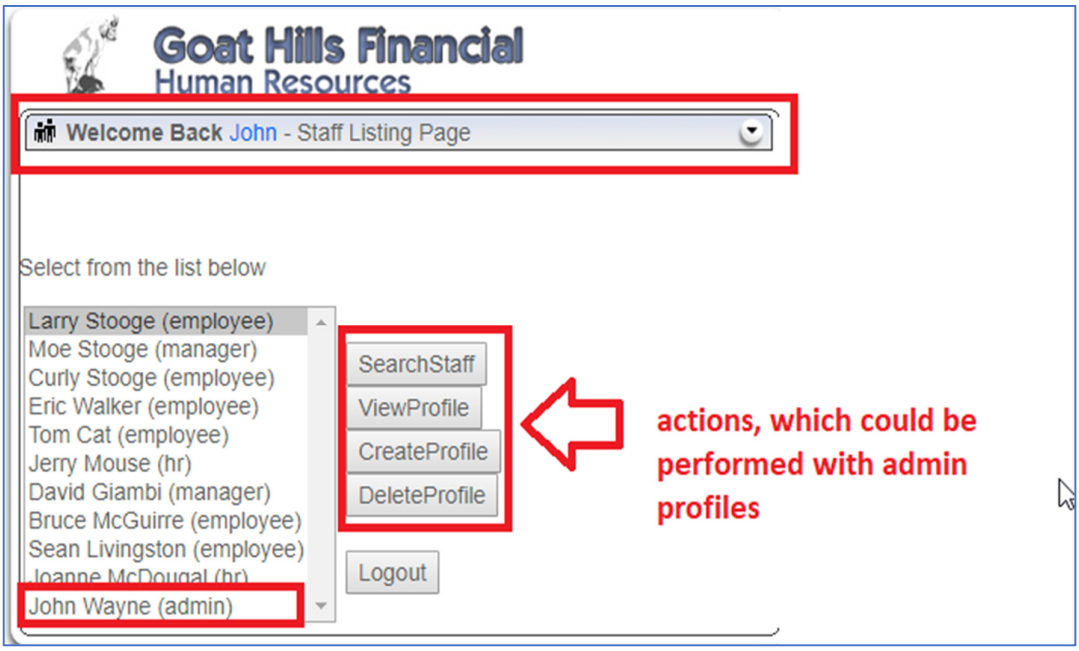
- Application has users with different privileges- **Admin** and **Non-admin** privileged users.
- "John Wayne" is the Admin privileged user.
- "Tom Cat" is a Non-admin privileged user.

Steps:

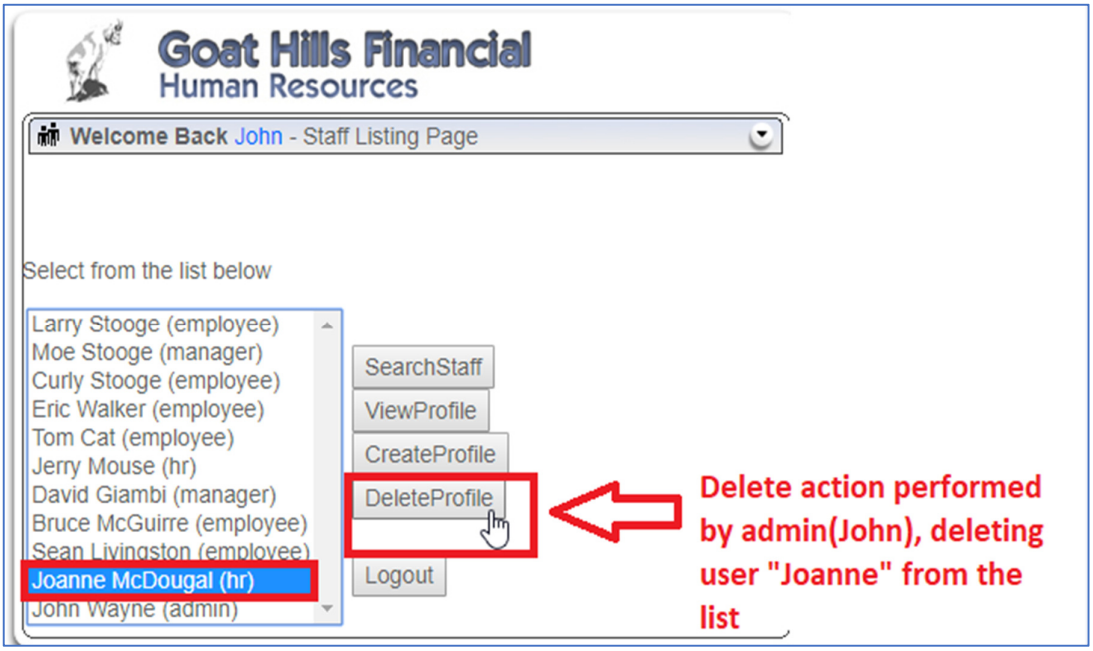
1- Admin user logs into the application.



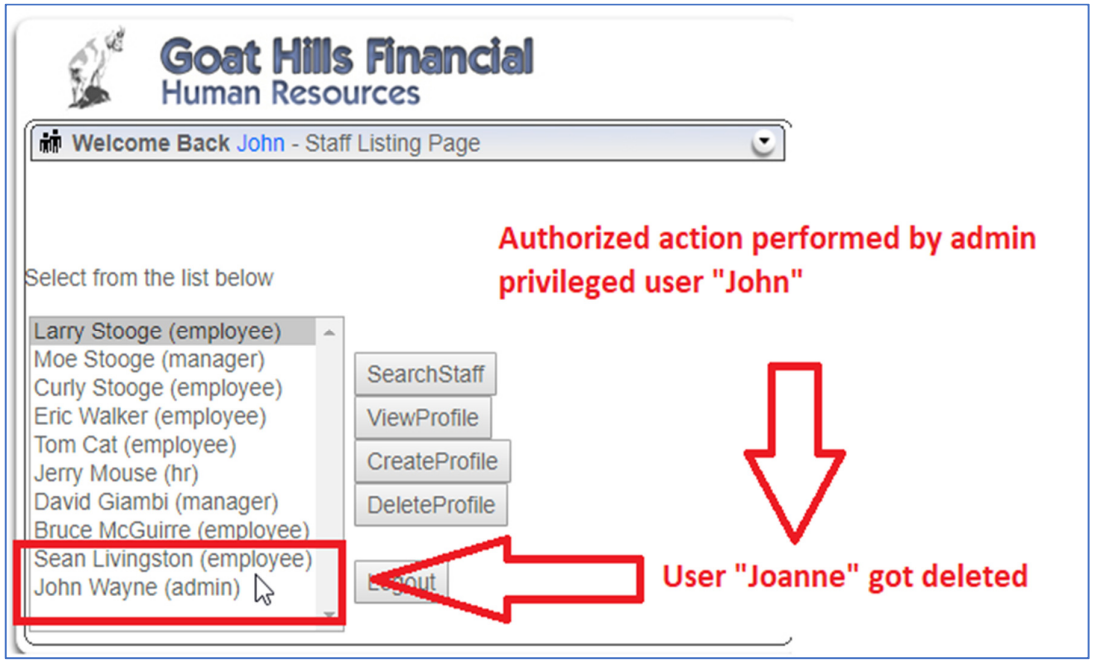
2- The various actions which could be performed by the Admin user “John”.



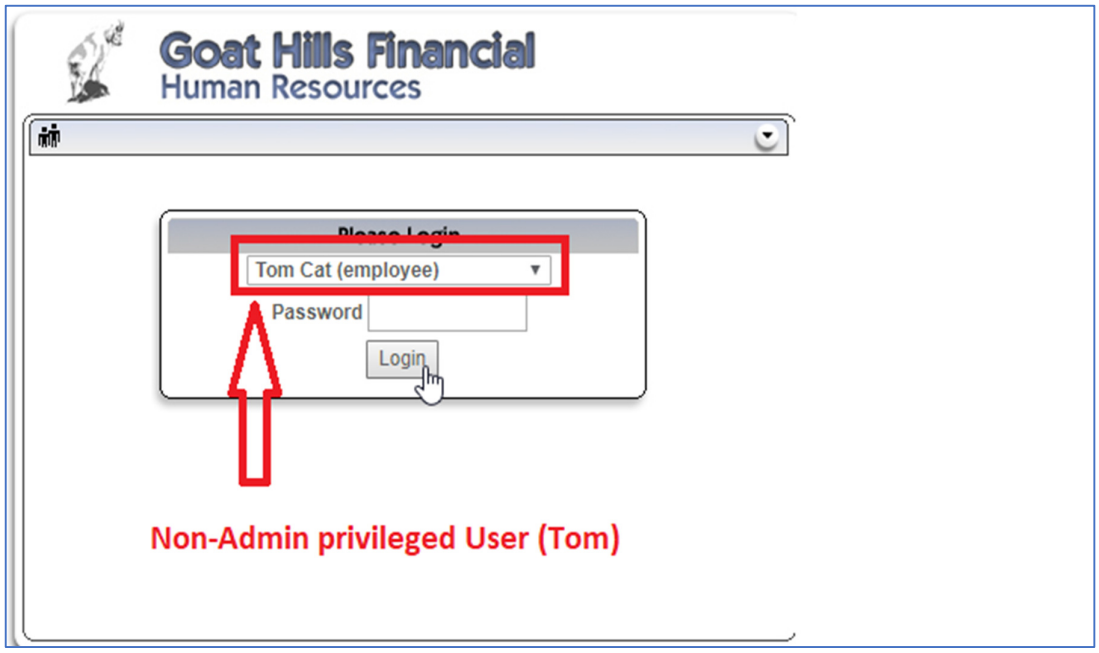
3- Admin user selects any other user in the list and deletes him/her from the list.



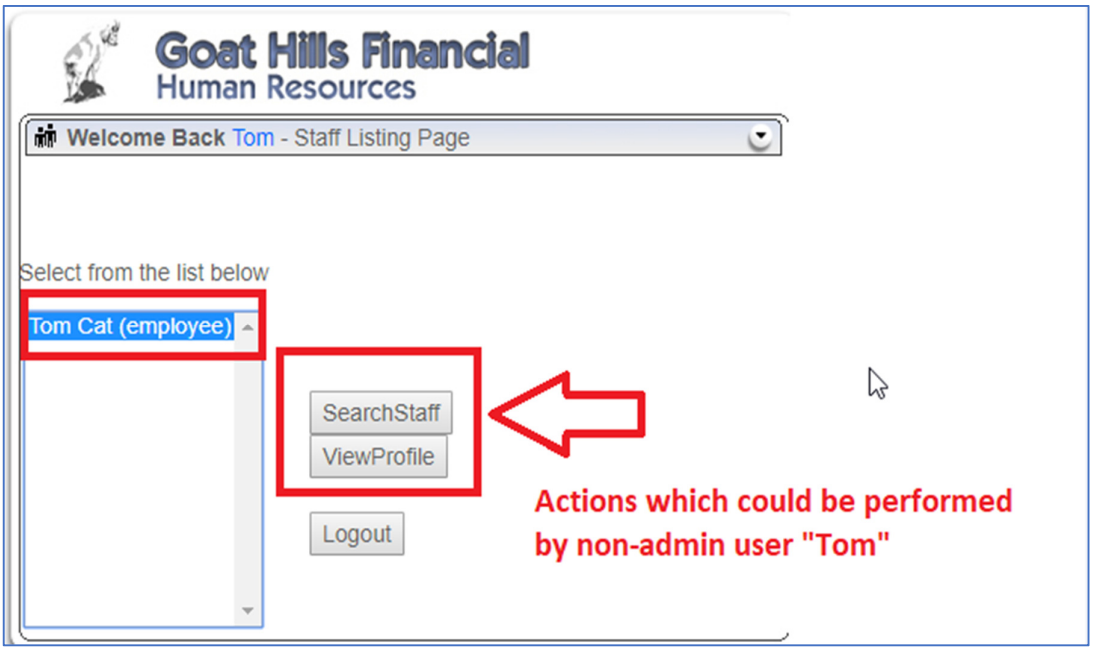
4- User gets deleted from the list as the action is performed with the administrative privileges.



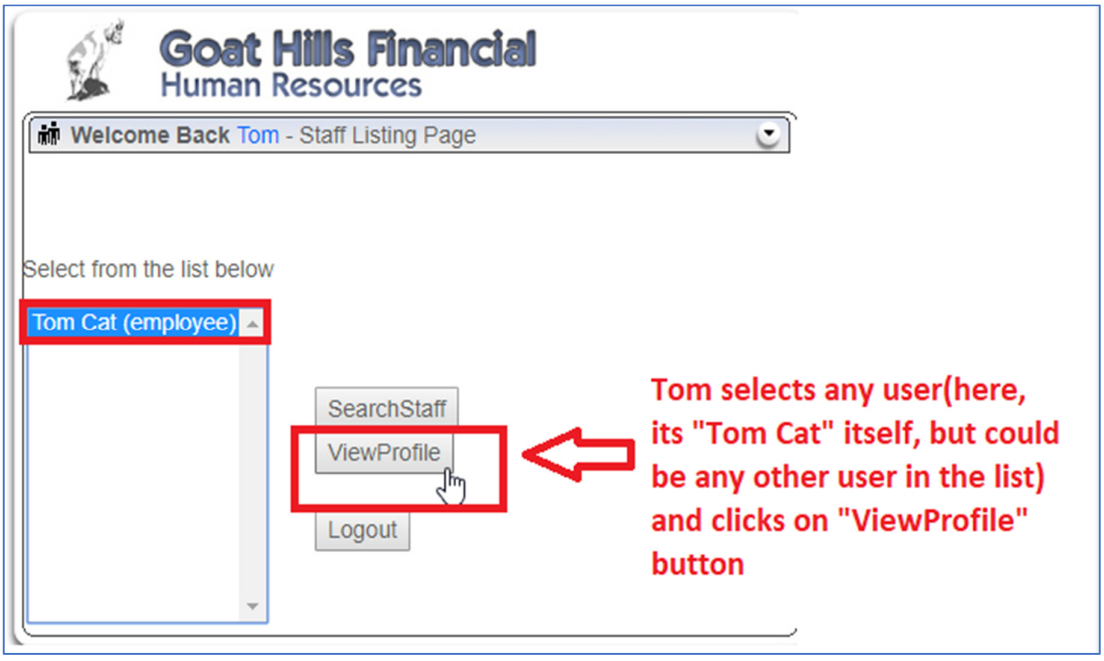
5- Non-Admin user logs into the application.



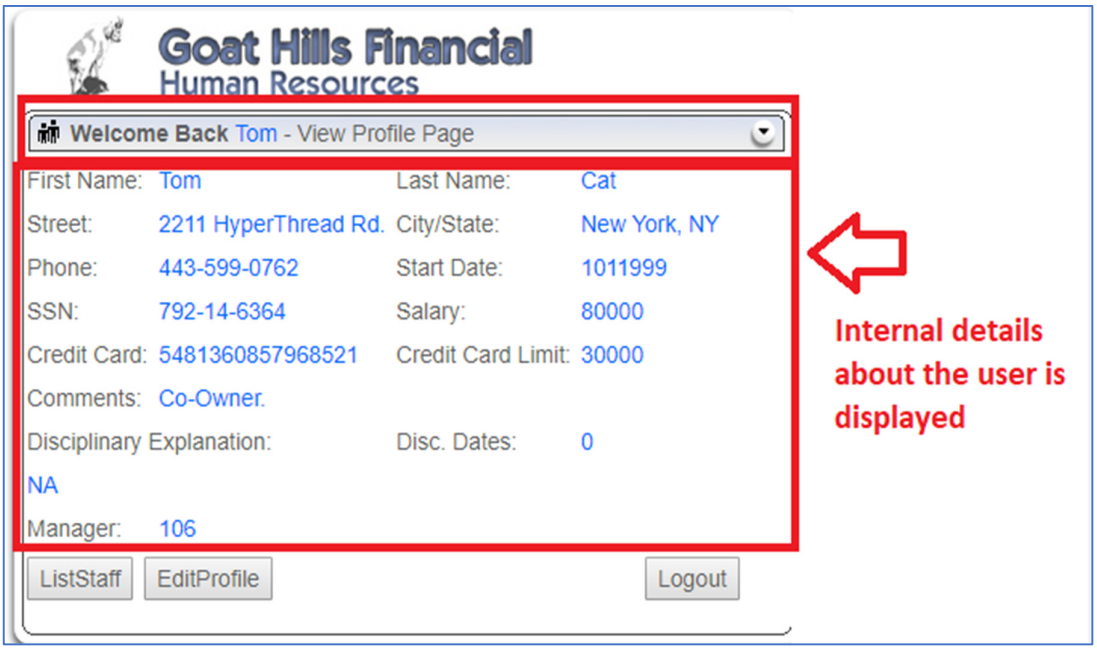
6- The various actions which could be performed by the Non-Admin user “Tom”.



7- Note that Non-Admin user can only select any user in the list (though, here we have only one employee in the list but could be more) to view his/her profile.



8- View Details of the employee on legit action by the non-admin user “Tom”.



9- Now, rather than performing legit action, disgruntled non-admin user (“Tom”) performs malicious action. Clicks on the **“ViewProfile”** button and intercepts the request made to the server.

10- He then modifies the **“action”** parameter to **“DeleteProfile”** and **forwards the modified request to the server.**

11- The response below from the server shows that the employee was deleted successfully.

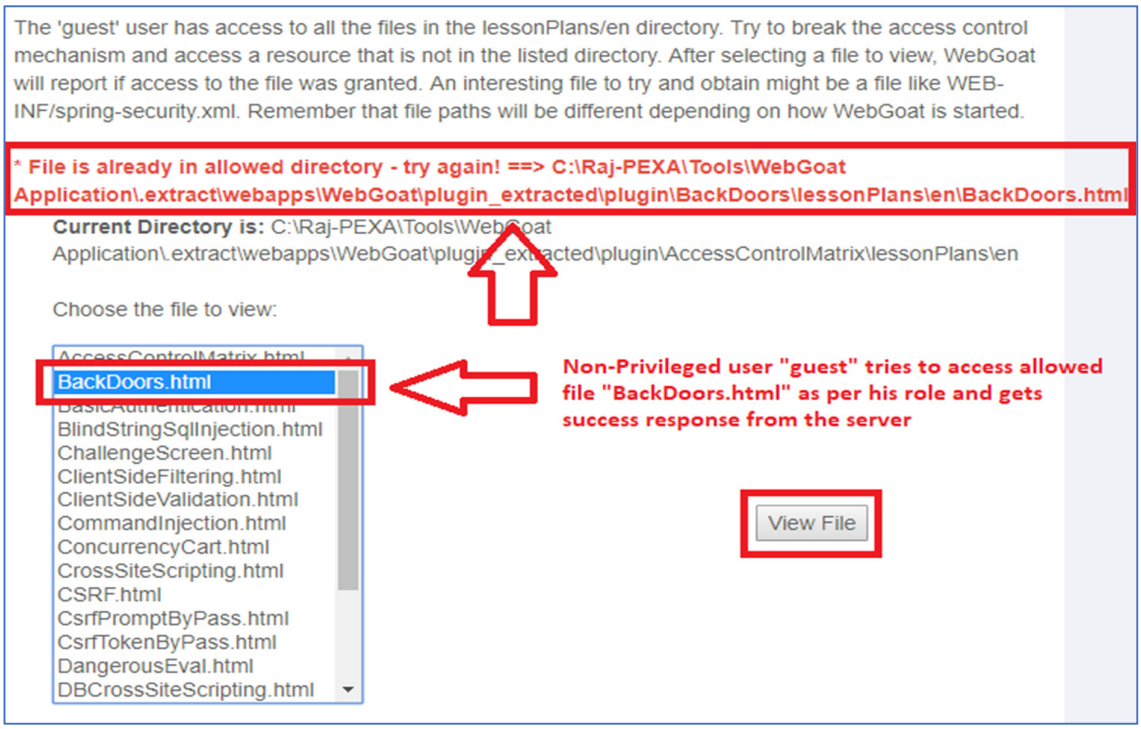
"The above demo shows that the non-admin user "Tom" could perform the malicious activity (deleting other users) because of improper server-side validations."

Example-2: <u>Bypassing Path Based Access Control Scheme</u>

- Application has users with different privileges to access different files within the File-System.
- Users can access the files based on their roles and permissions.
- Malicious User attempts to access unauthorized files out of his roles/permissions.

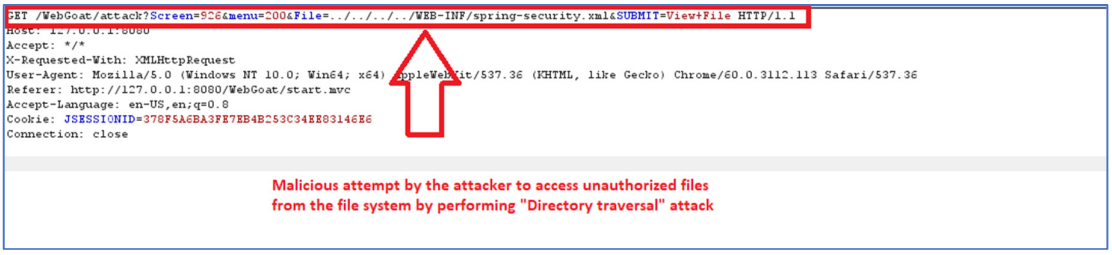
<u>Steps:</u>

1- User logs into the application and tries to access allowable "BackDoors.html" file.

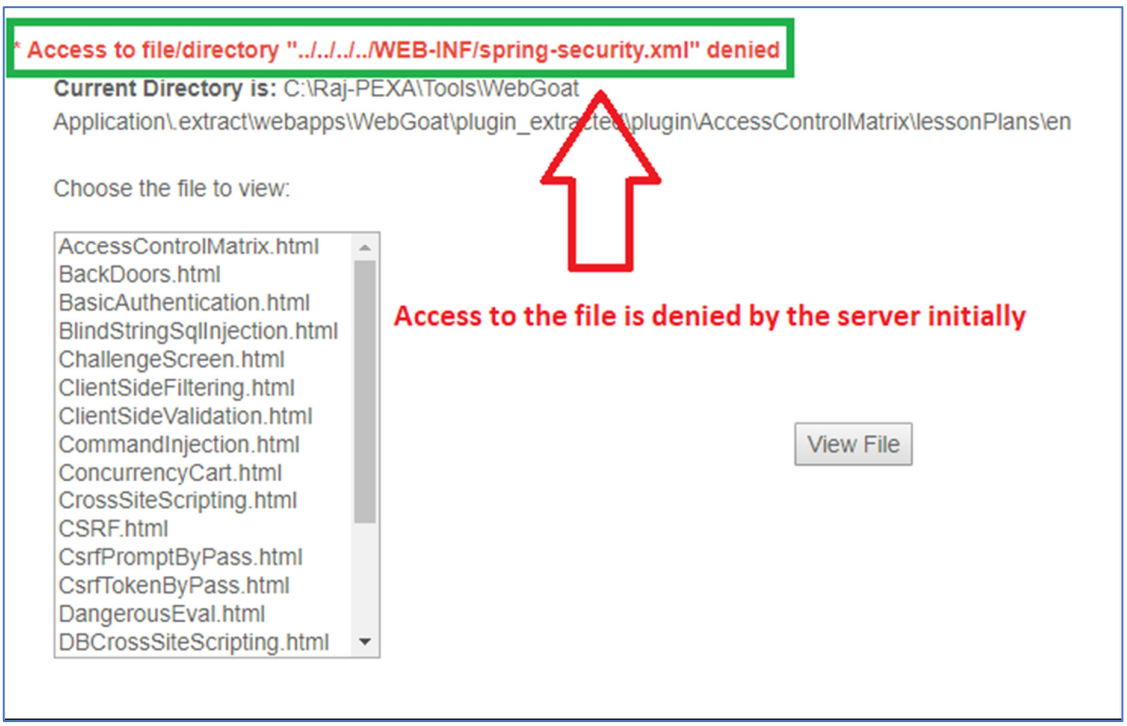


2- The file is accessible to the user “guest”.

3- Now the guest user tries to perform malicious activity and tries to access **unauthorized file “spring-security.xml”** by performing **“directory traversal” attack.**

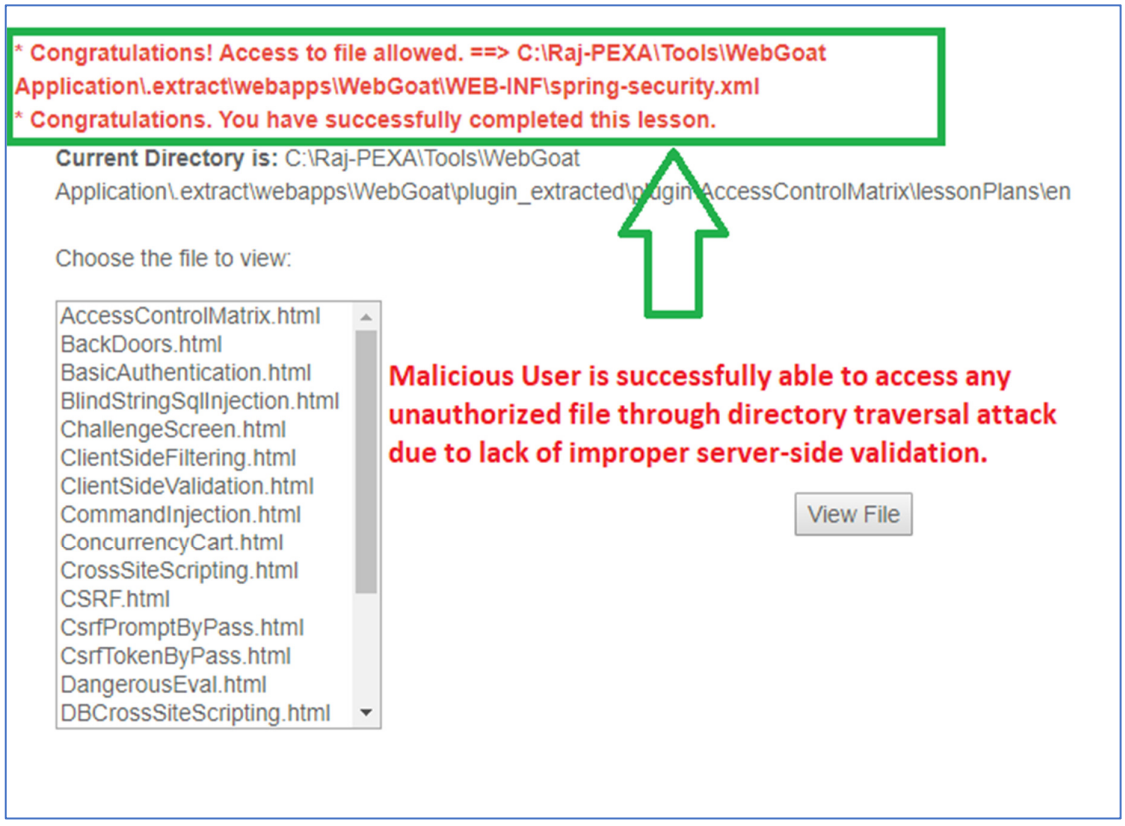


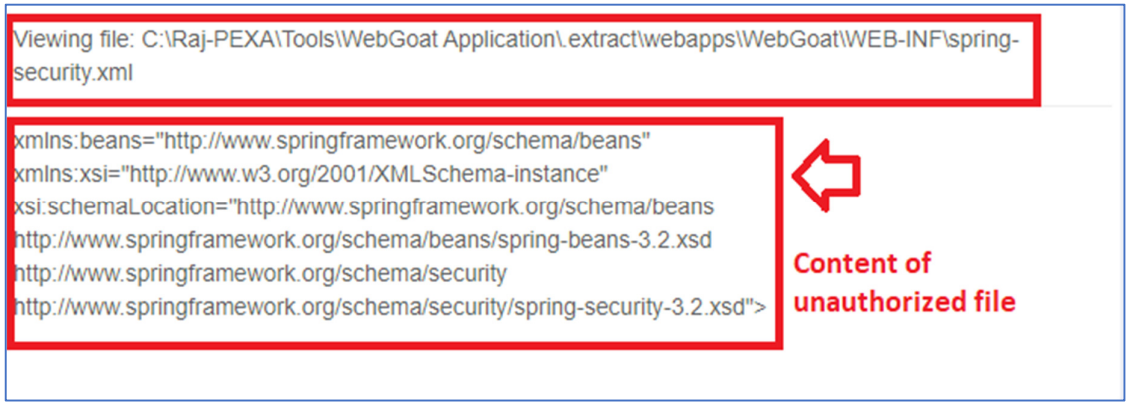
4- Guest user is denied accessibility based on either file path not found or could be also due to roles/permissions.



5- Guest user again tries to access the unauthorized file “spring-security.xml” by further performing directory traversal using **../ (dot dot slash)** concept.

6- This time, user gets successful response from the server and able to access/view the restrictive file “spring-security.xml” within the file-system.





7- The above attack reveals the database username/password to the guest user acting as an attacker in this case.

"The above demo shows how guest user could perform the malicious directory traversal attack to gain unauthorized access to restrictive file 'spring-security.xml' meant only for admins.

The vulnerability got exploited because of improper server-side validations."

Remediation Techniques

1- **Never rely on Client-Side validations** for security perspective.
2- **Proper server-side validations must be implemented** to validate every user request.
3- **Assume all user inputs as malicious.**
4- Apart from implementing client-side validations for any functionality, **the same should also be replicated and validated at the server end** to prevent any malicious attempt if attacker bypasses the client-side validations.

Conclusion

Thus, **by combining both server-side and client-side methods, we can get the best of the two: fast response, more secure validation and better user experience.**

"Be aware that any JavaScript input validation performed on the client can be bypassed by an attacker that disables JavaScript or uses a Web Proxy.

Ensure that any input validation performed on the client is also performed on the server."

Key Notes

1- For additional reading and better understanding, please refer below links

https://www.owasp.org/index.php/Input_Validation_Cheat_Sheet
https://www.smashingmagazine.com/2009/07/web-form-validation-best-practices-and-tutorials/