

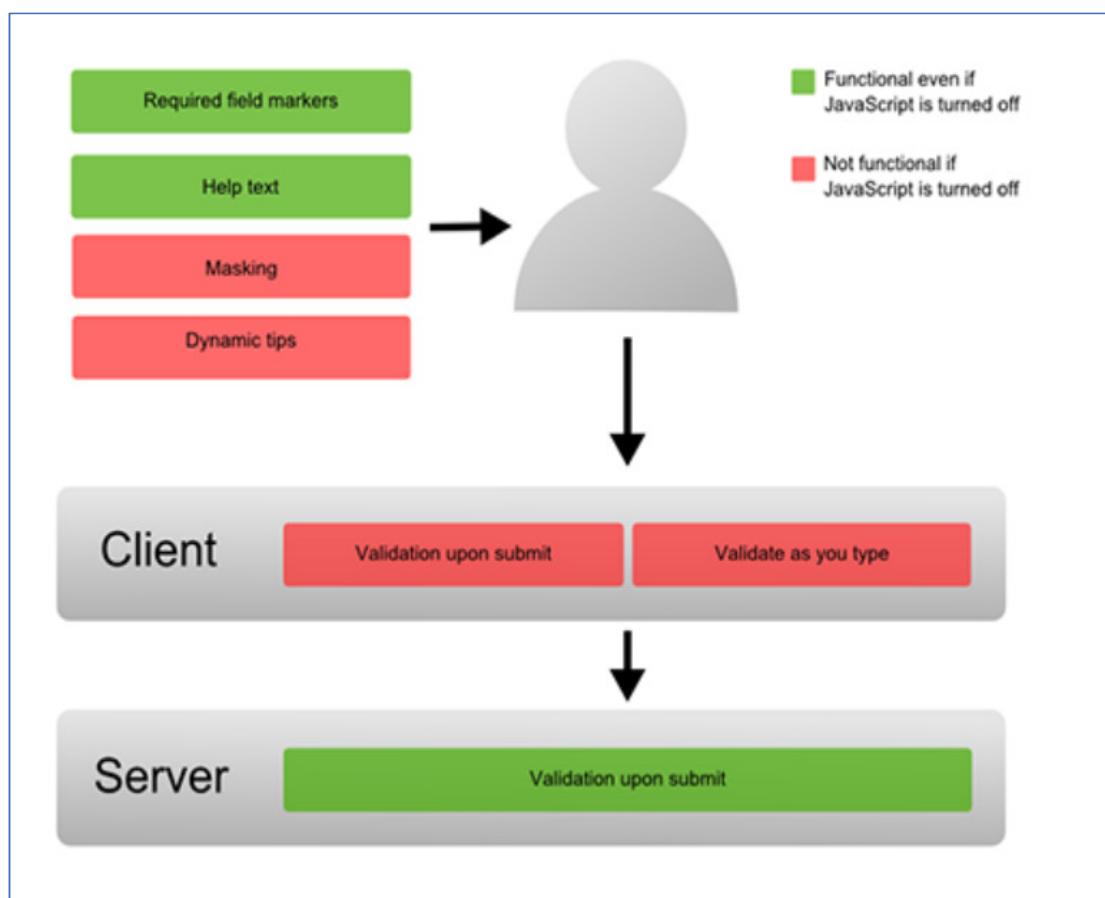
Importance of Server-Side Validations

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

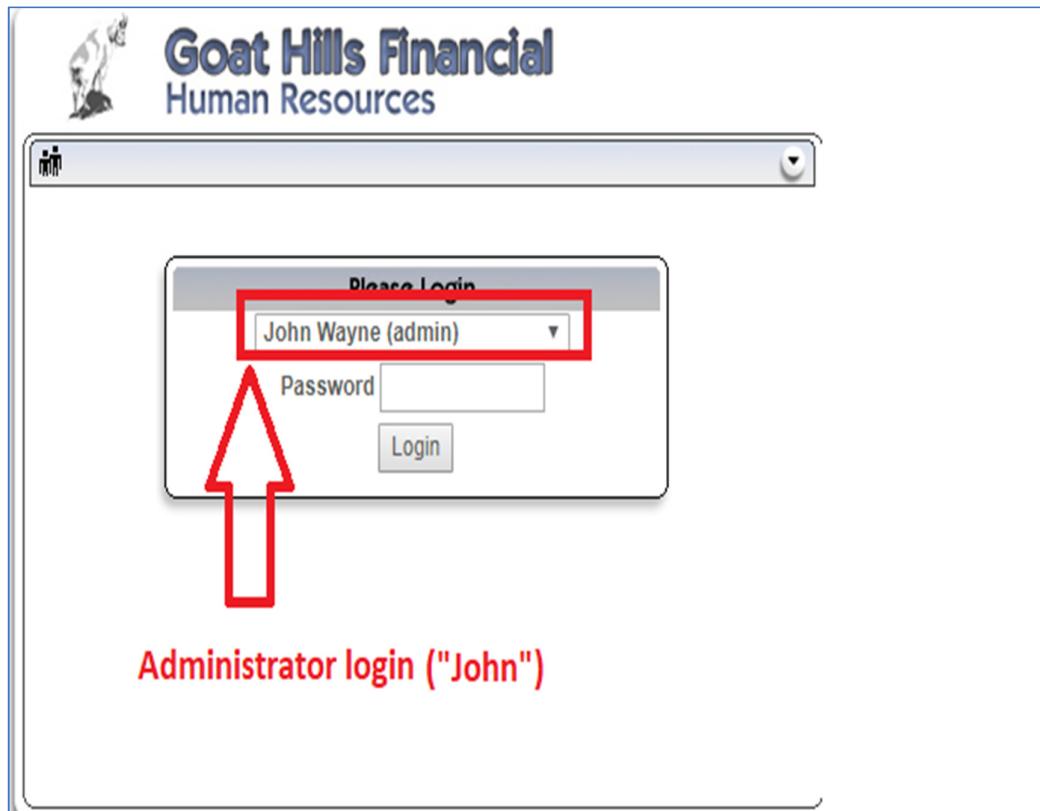
For demonstration purposes, we have taken **WebGoat** application, which is an intentionally vulnerable web app by **OWASP** (Open Web Application Security Project) to understand different security vulnerabilities and their implications.

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".

Welcome Back John - Staff Listing Page

Select from the list below

- Larry Stooge (employee)
- Moe Stooge (manager)
- Curly Stooge (employee)
- Eric Walker (employee)
- Tom Cat (employee)
- Jerry Mouse (hr)
- David Giambi (manager)
- Bruce McGuirre (employee)
- Sean Livingston (employee)
- Joanne McDougal (hr)
- John Wayne (admin)

SearchStaff
ViewProfile
CreateProfile
DeleteProfile
Logout

actions, which could be performed with admin profiles

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

Welcome Back John - Staff Listing Page

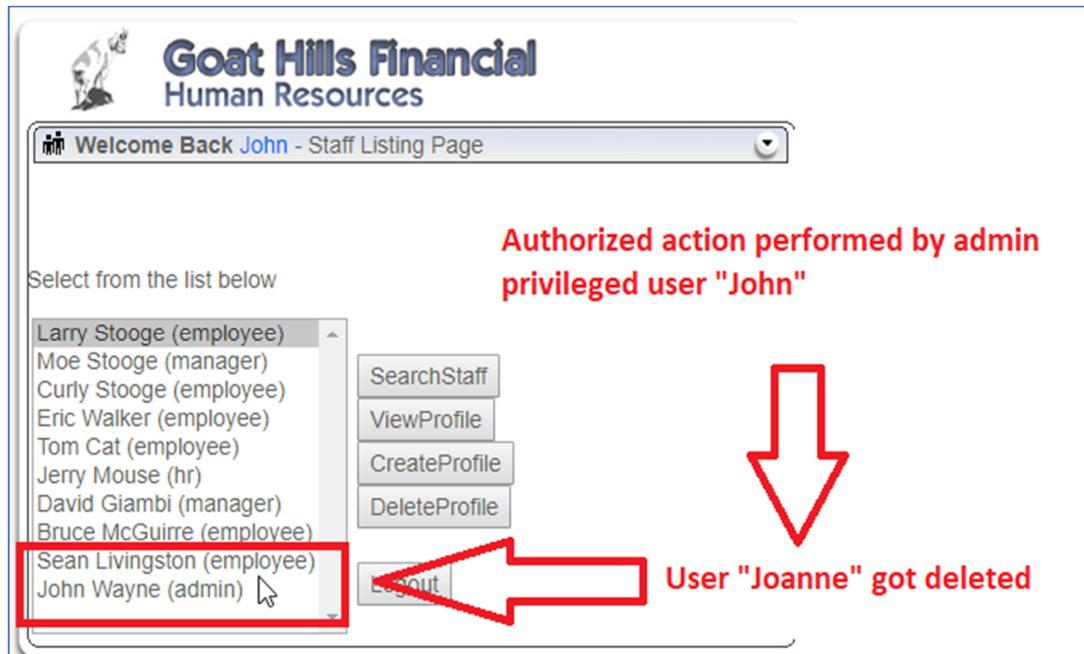
Select from the list below

- Larry Stooge (employee)
- Moe Stooge (manager)
- Curly Stooge (employee)
- Eric Walker (employee)
- Tom Cat (employee)
- Jerry Mouse (hr)
- David Giambi (manager)
- Bruce McGuirre (employee)
- Sean Livingston (employee)
- Joanne McDougal (hr)
- John Wayne (admin)

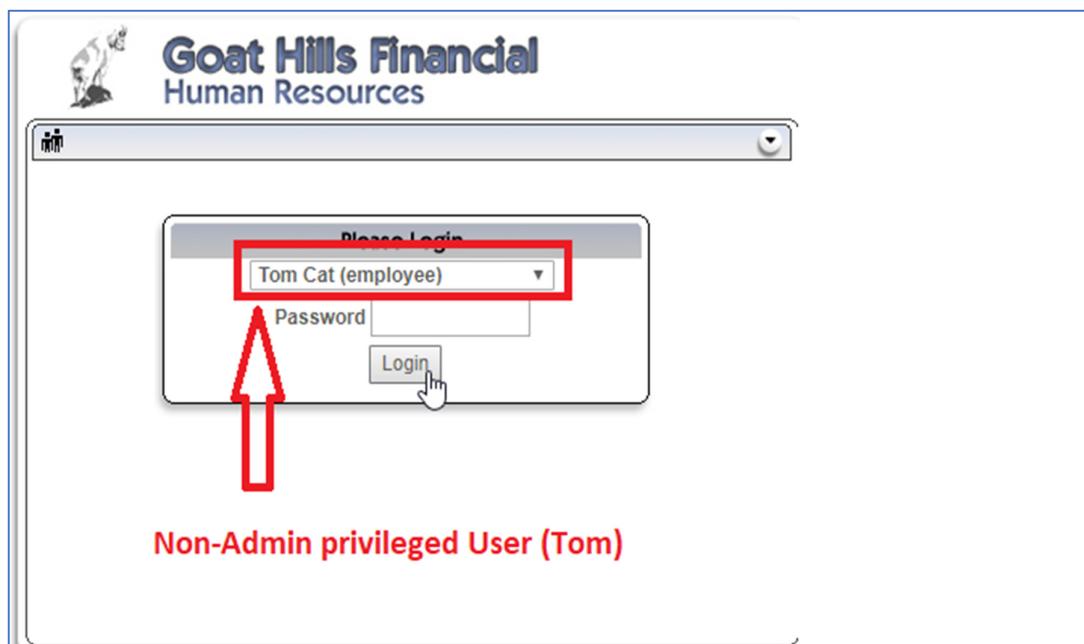
SearchStaff
ViewProfile
CreateProfile
DeleteProfile
Logout

Delete action performed by admin(John), deleting user "Joanne" from the list

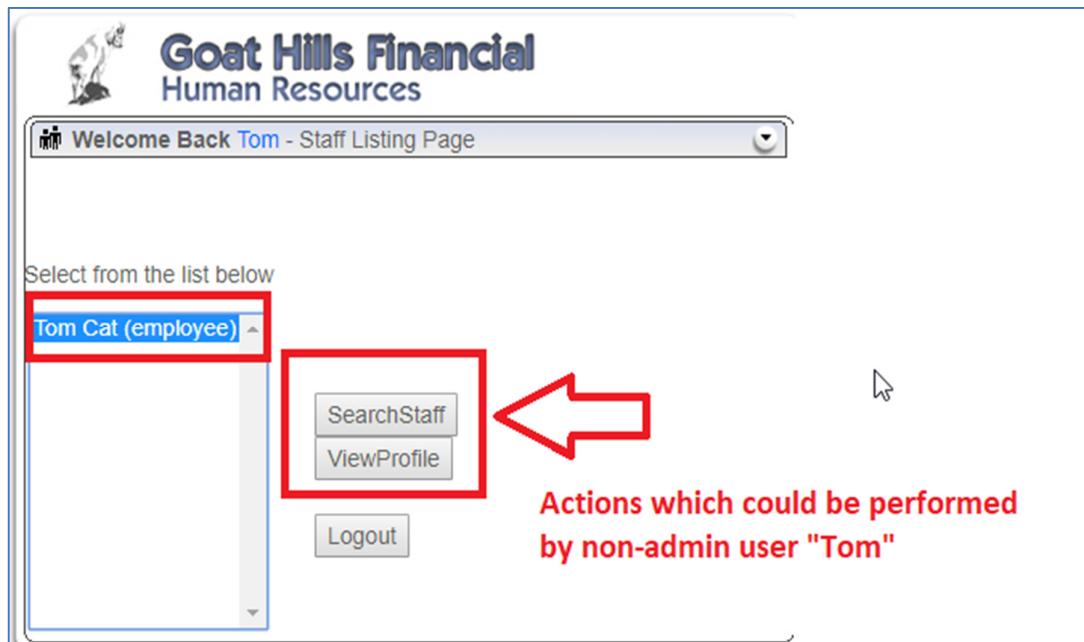
- 4- User gets deleted from the list as the action is performed with the administrative privileges.



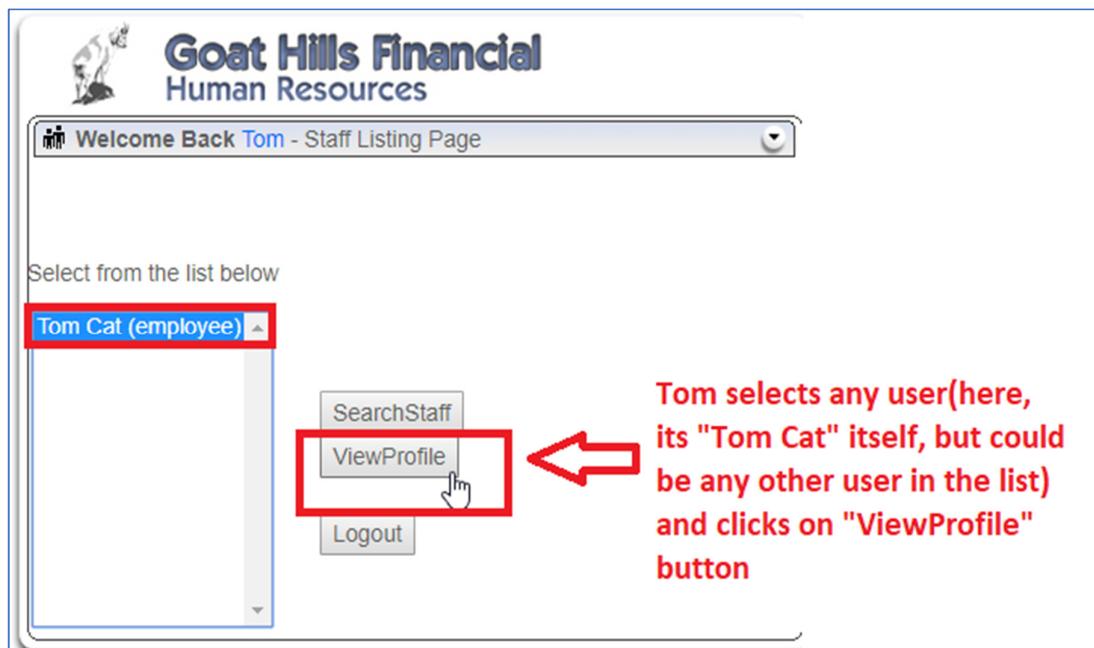
- 5- Non-Admin user logs into the application.



- 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”.

Welcome Back Tom - View Profile Page

First Name:	Tom	Last Name:	Cat
Street:	2211 HyperThread Rd.	City/State:	New York, NY
Phone:	443-599-0762	Start Date:	1011999
SSN:	792-14-6364	Salary:	80000
Credit Card:	5481360857968521	Credit Card Limit:	30000
Comments:	Co-Owner.		
Disciplinary Explanation:	Disc. Dates: 0		
NA			
Manager:	106		

ListStaff EditProfile Logout

- 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.

```
GET /WebGoat/attack?Screen=940&menu=200&stage=1&employee_id=105&action=ViewProfile HTTP/1.1
Host: 127.0.0.1:8080
Accept: /*
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
Referer: http://127.0.0.1:8080/WebGoat/start.mvc
Accept-Language: en-US,en;q=0.8
Cookie: JSESSIONID=370F5A6BA3FE7EB4B253C34EE83146E6
Connection: close
```

The action performed on clicking "View Profile" button

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

```
GET /WebGoat/attack?Screen=940&menu=200&stage=1&employee_id=105&action=DeleteProfile HTTP/1.1
Host: 127.0.0.1:8080
Accept: /*
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
Referer: http://127.0.0.1:8080/WebGoat/start.mvc
Accept-Language: en-US,en;q=0.8
Cookie: JSESSIONID=370F5A6BA3FE7EB4B253C34EE83146E6
Connection: close
```

Malicious User or attacker modifies the action button to "DeleteProfile" and tries to delete any of the other user in the list

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

User "Tom" is able to successfully
* You have completed Stage 1: Bypass Business Layer Access Control
* Welcome to Stage 2: Add Business Layer Access Control
bypass the client side validation
and delete any user in the list

"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: Bypassing Path Based Access Control Scheme

- 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.

Steps:

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

The 'guest' user has access to all the files in the lessonPlans/en directory. Try to break the access control mechanism and access a resource that is not in the listed directory. After selecting a file to view, WebGoat will report if access to the file was granted. An interesting file to try and obtain might be a file like WEB-INF/spring-security.xml. Remember that file paths will be different depending on how WebGoat is started.

* File is already in allowed directory - try again! ==> C:\Raj-PEXA\Tools\WebGoat
Application\extract\webapps\WebGoat\plugin_extracted\plugin\BackDoors\lessonPlans\en\BackDoors.html

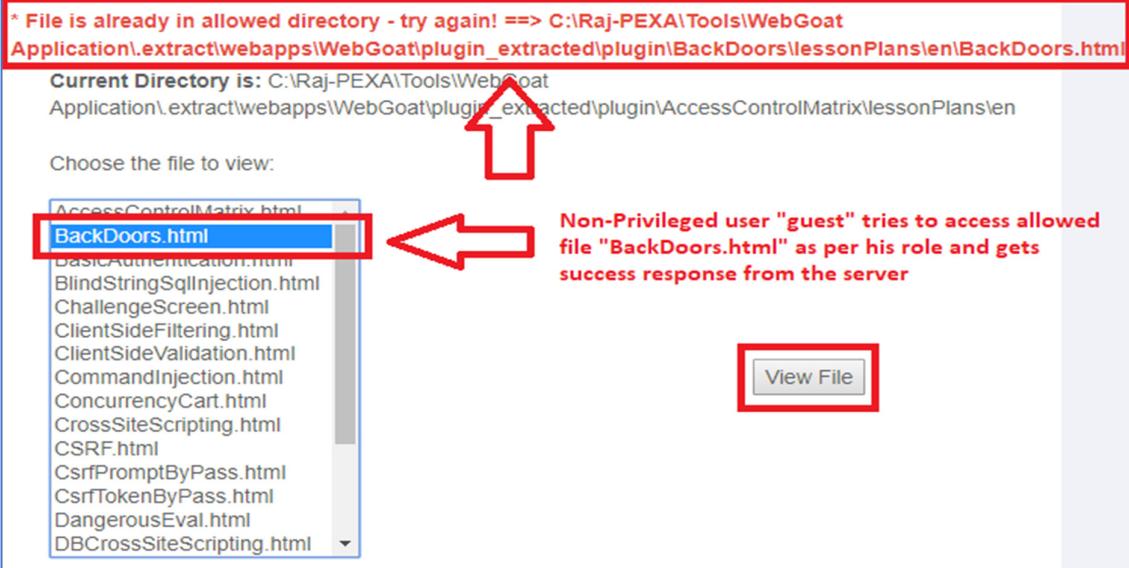
Current Directory is: C:\Raj-PEXA\Tools\WebGoat
Application\extract\webapps\WebGoat\plugin_extracted\plugin\AccessControlMatrix\lessonPlans\en

Choose the file to view:

AccessControlMatrix.html
BackDoors.html
BasicAuthentication.html
BlindStringSqlInjection.html
ChallengeScreen.html
ClientSideFiltering.html
ClientSideValidation.html
CommandInjection.html
ConcurrencyCart.html
CrossSiteScripting.html
CSRF.html
CsrfPromptByPass.html
CsrfTokenByPass.html
DangerousEval.html
DBCrossSiteScripting.html

Non-Privileged user "guest" tries to access allowed file "BackDoors.html" as per his role and gets success response from the server

View File



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

A screenshot of a web browser window. The URL bar shows a POST request to "/WebGoat/attack?Screen=92&menu=200&File=BasicAuthentication.html&SUBMIT=View+File HTTP/1.1". The page content displays the response: "Allowable file "BackDoors.html"". A red arrow points from the question mark in the URL bar down to the word "Allowable".

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

A screenshot of a web browser window. The URL bar shows a POST request to "/WebGoat/attack?Screen=92&menu=200&File=../../../../WEB-INF/spring-security.xml&SUBMIT=View+File HTTP/1.1". The page content displays the response: "Malicious attempt by the attacker to access unauthorized files from the file system by performing "Directory traversal" attack". A red arrow points from the question mark in the URL bar up to the word "Malicious".

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

A screenshot of a web browser window. The title bar says "* Access to file/directory ".../WEB-INF/spring-security.xml" denied". The page content shows the current directory as "C:\Raj-PEXA\Tools\WebGoat\...". Below it, a message says "Access to the file is denied by the server initially". On the left, there is a list of files: AccessControlMatrix.html, BackDoors.html, BasicAuthentication.html, BlindStringSqlInjection.html, ChallengeScreen.html, ClientSideFiltering.html, ClientSideValidation.html, CommandInjection.html, ConcurrencyCart.html, CrossSiteScripting.html, CSRF.html, CsrfPromptByPass.html, CsrfTokenByPass.html, DangerousEval.html, DBCrossSiteScripting.html. A red arrow points from the title bar down to the word "denied".

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



`GET /WebGoat/attack?Screen=92&menu=200&File=../../../../WEB-INF/spring-security.xml&SUBMIT=View+File HTTP/1.1
Host: 127.0.0.1:8080
Accept: */*
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
Referer: http://127.0.0.1:8080/WebGoat/start.mvc
Accept-Language: en-US,en;q=0.8
Cookie: JSESSIONID=370F5A6BA3FE7EB4BC53C34EE83146E6
Connection: close`

Malicious attempt by the attacker to access unauthorized file through "Directory traversal" attack

- 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.



* Congratulations! Access to file allowed. ==> C:\Raj-PEXA\Tools\WebGoat Application\extract\webapps\WebGoat\WEB-INF\spring-security.xml
* Congratulations. You have successfully completed this lesson.

Current Directory is: C:\Raj-PEXA\Tools\WebGoat Application\extract\webapps\WebGoat\plugin_extracted\plugin\AccessControlMatrix\lessonPlans\en

Choose the file to view:

AccessControlMatrix.html
BackDoors.html
BasicAuthentication.html
BlindStringSqlInjection.html
ChallengeScreen.html
ClientSideFiltering.html
ClientSideValidation.html
CommandInjection.html
ConcurrencyCart.html
CrossSiteScripting.html
CSRF.html
CsrfPromptByPass.html
CsrfTokenByPass.html
DangerousEval.html
DBCrossSiteScripting.html

Malicious User is successfully able to access any unauthorized file through directory traversal attack due to lack of improper server-side validation.

View File



Viewing file: C:\Raj-PEXA\Tools\WebGoat Application\extract\webapps\WebGoat\WEB-INF\spring-security.xml

```
xmlns:beans="http://www.springframework.org/schema/beans"  
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
xsi:schemaLocation="http://www.springframework.org/schema/beans  
http://www.springframework.org/schema/beans/spring-beans-3.2.xsd  
http://www.springframework.org/schema/security  
http://www.springframework.org/schema/security/spring-security-3.2.xsd">
```

Content of unauthorized file

```
login-page="/login.mvc"
default-target-url="/welcome.mvc"
authentication-failure-url="/login.mvc?error"
username-parameter="username"
password-parameter="password"
always-use-default-target="true"/>
```

DB username/Password are disclosed

Contents of unauthorized file accessible to attacker



```
<user-service>

<user name="guest" password="guest" authorities="ROLE_WEBGOAT_USER" />
<user name="webgoat" password="webgoat" authorities="ROLE_WEBGOAT_ADMIN" />
<user name="server" password="server" authorities="ROLE_SERVER_ADMIN" />
```

- 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/>