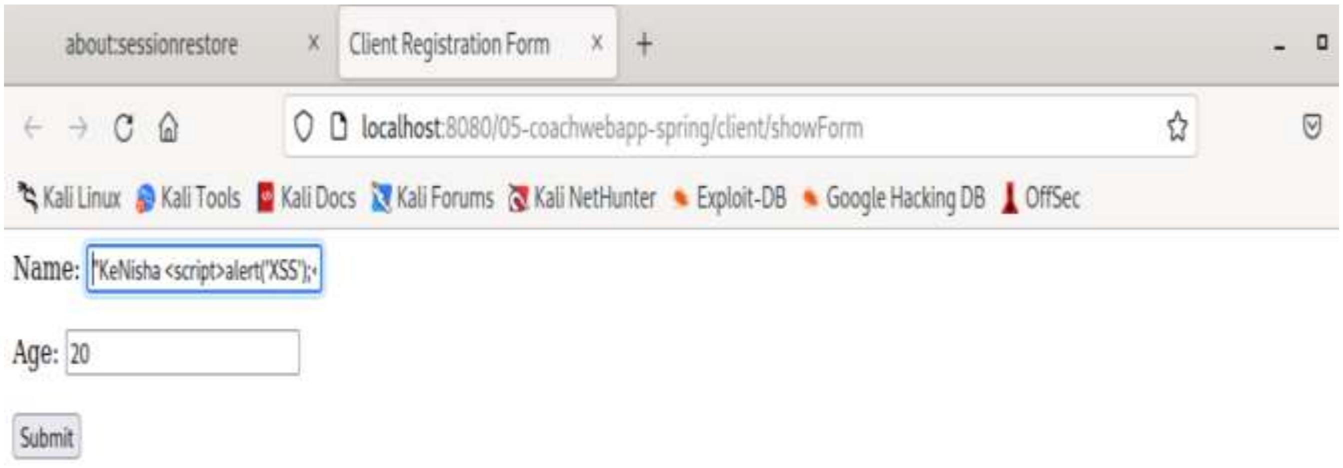
Pass task 4.1P: Injection attack preventions using hibernate validator

Since users could type scripts or tagged input values containing ‘<’ and ‘>’, then I created a regex pattern for a sample HTML input for an interactive web application based on the 05-coachwebapp-spring web application. This reduced the risks of applications being vulnerable to XSS attacks. To assess how effective this regex-based avoidance method described in the previous parts was, I also tried several input patterns.

Input values

First Test

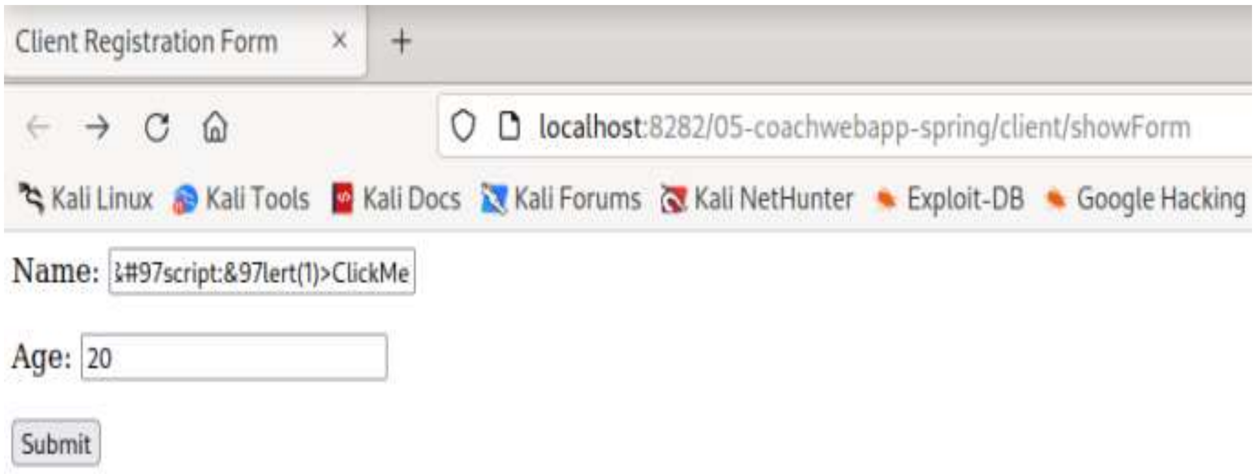
Appling XSS script - name "<script>alert("XSS"); </script>"



Name: "KeNisha <script>alert('XSS');<

Age: 20

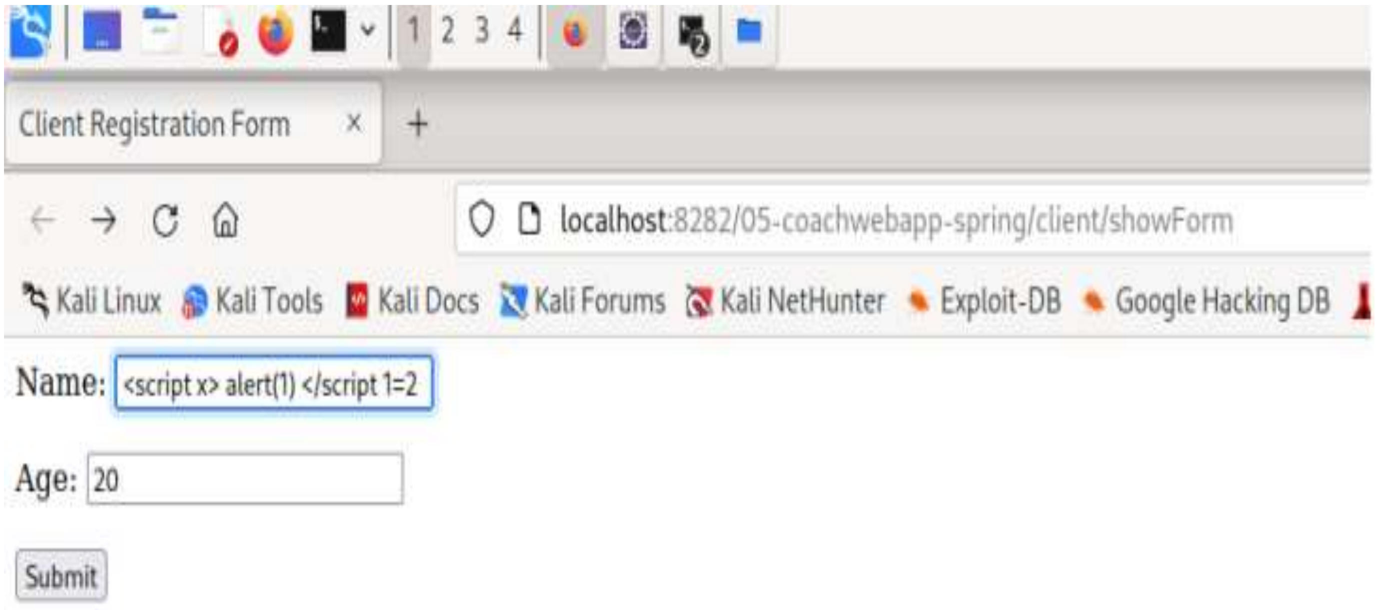
Second Test
obfuscated XSS attack using character encoding - <a aa aaa aaaa aaaaa aaaaaa aaaaaaa aaaaaaaa aaaaaaaaa aaaaaaaaaa href=j&#97v&#97script:&#97lert(1)>ClickMe



Name: &#97script:&97lert(1)>ClickMe

Age: 20

Third Test
XSS attack using a script with additional attributes - <script x> alert(1) </script 1=2



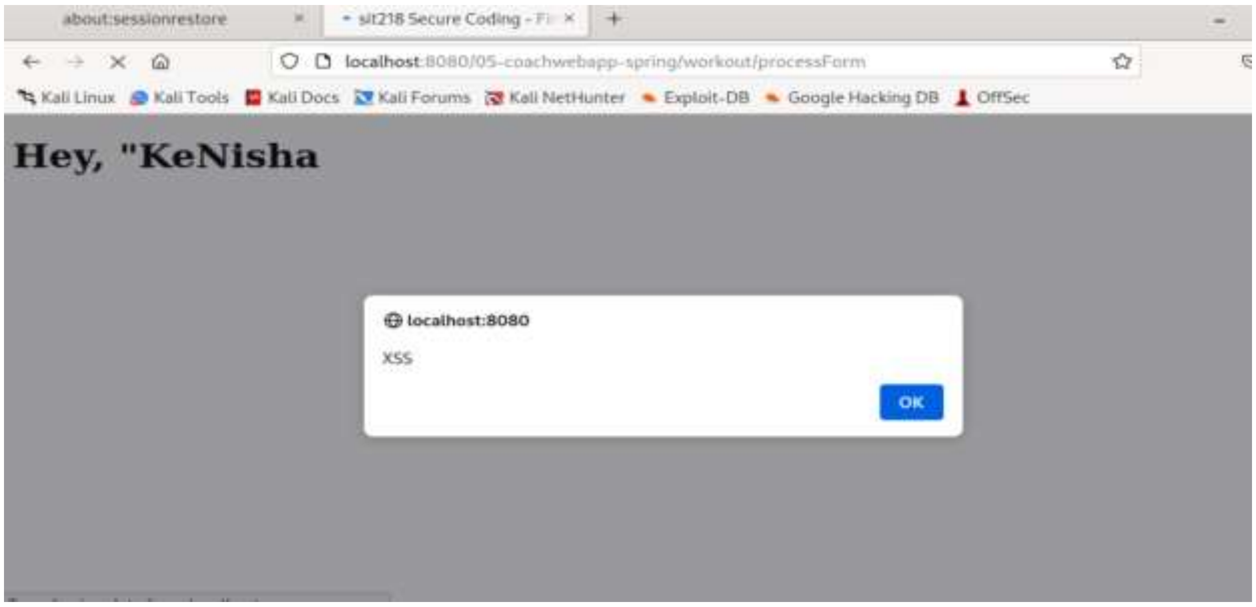
Name: <script x> alert(1) </script 1=2

Age: 20

Outputs of the Test

First Test Output

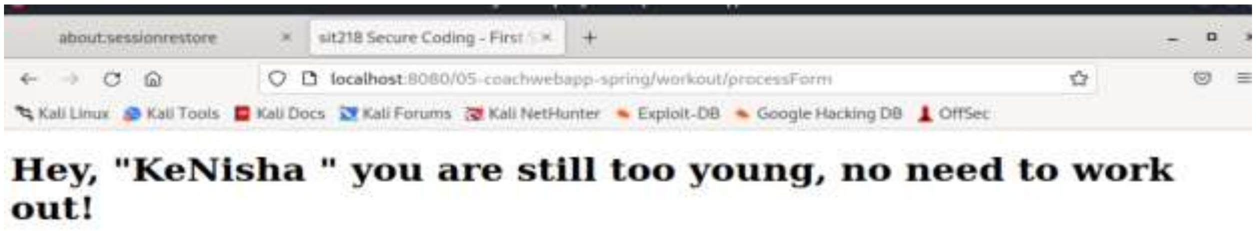
This rather obvious type of an attack did not raise any alarm with the input “regex pattern” and the script illustrated this by running with reckless abandon on the`<script>alert(‘XSS’);</script>` vulnerability. This ensured this that the defense against XSS attempt was low, despite the fact that the attempt was basic, and it used only regex and default script tags.



Hey, "KeNisha

localhost:8080

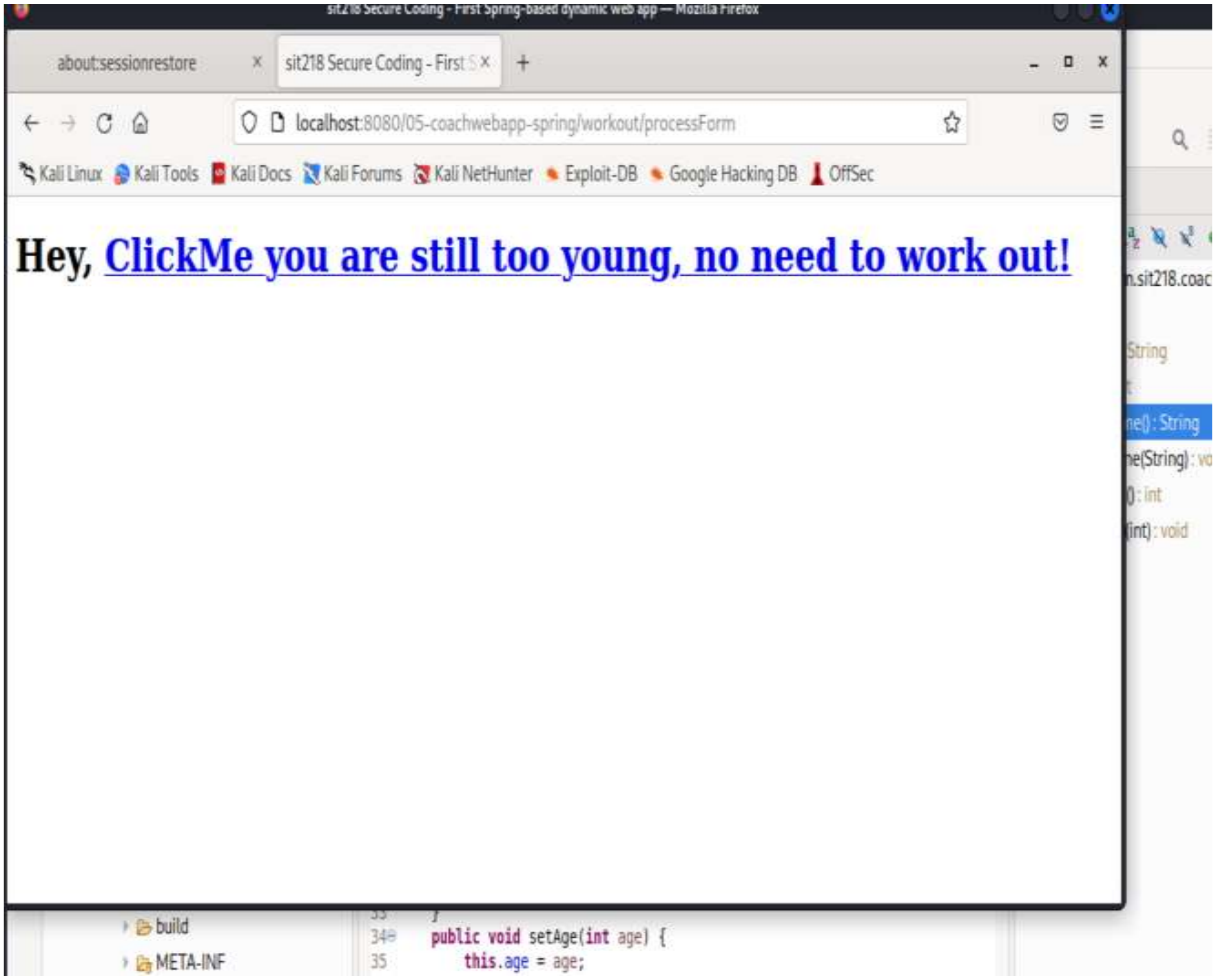
XSS



Hey, "KeNisha " you are still too young, no need to work out!

Second Test Output

Raw ”<” and ”>” were what the regex was seeking. Consequently, the input <a href=j&#97v&#97script:There were no filtered protests, ClickMe did not get filtered and none of the coded, concealed forms. This showed that they are ready for an even more complex form of the XSS attack.

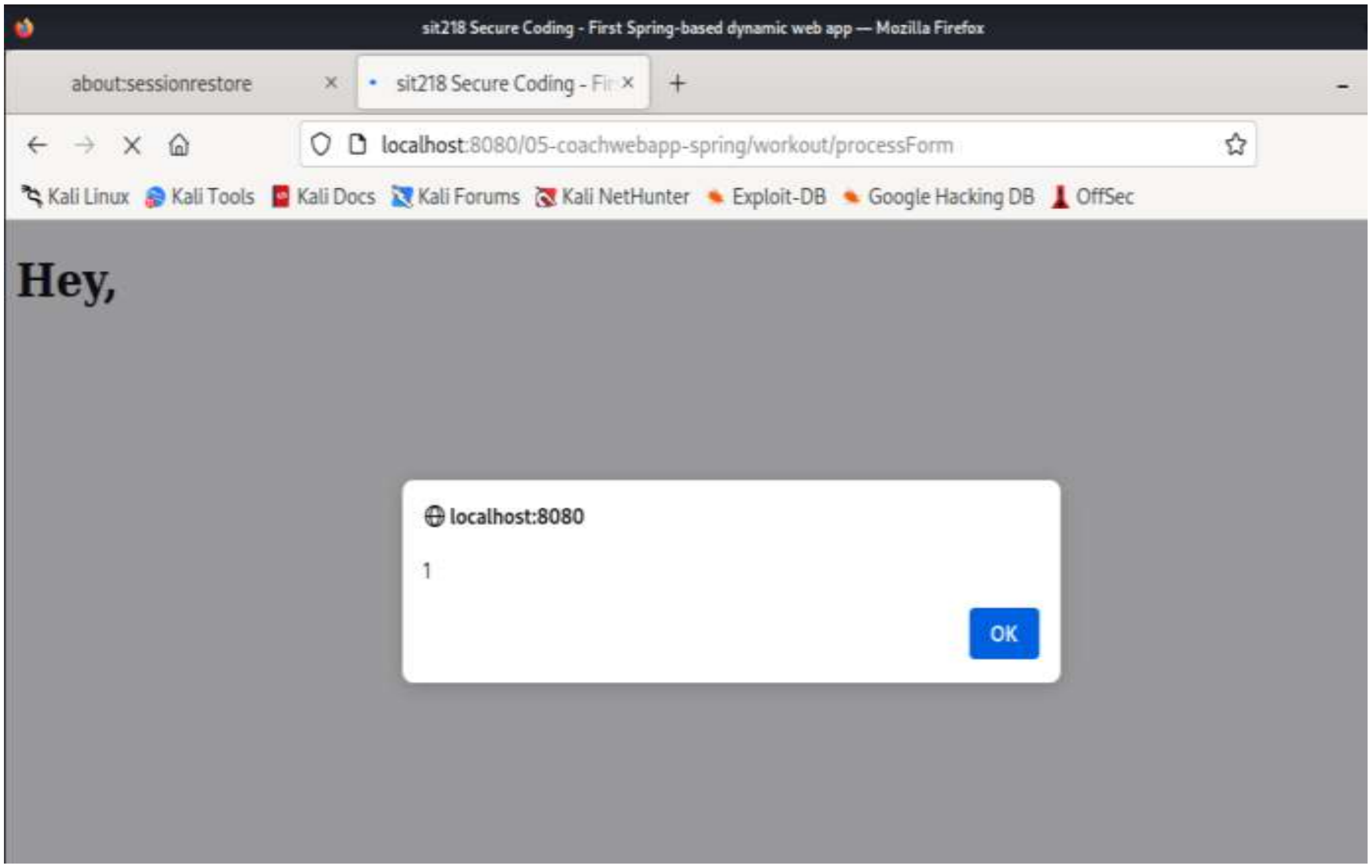

sit218 Secure Coding - First Spring-based dynamic web app — Mozilla Firefox
about:sessionrestore
sit218 Secure Coding - First S
localhost:8080/05-coachwebapp-spring/workout/processForm
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
build
META-INF

Hey, ClickMe you are still too young, no need to work out!

```
public void setAge(int age) {
    this.age = age;
```

Third Test Output

Also, the regex and that this filter ignored minor differences in the script tag format did not pertain to the input <script x> alert(1) </script 1=2.


sit218 Secure Coding - First Spring-based dynamic web app — Mozilla Firefox
about:sessionrestore
sit218 Secure Coding - Fir
localhost:8080/05-coachwebapp-spring/workout/processForm
Kali Linux
Kali Tools
Kali Docs
Kali Forums
Kali NetHunter
Exploit-DB
Google Hacking DB
OffSec
OK

Hey,

localhost:8080

1

An explanation of the inputs that were skipped

- ✓ Insufficient Regex Coverage
  - In this example, it failed to impose only another type of input, and effectively barred <and >. If one uses encoded characters, or simply divides the script tag, then this will not be hard to do at all

    .
- ✓ Tag Obfuscation
  - New trends in XSS attacks include tag splitting for example splitting the tag, \scr<script>ipt> whereby using a pattern match, does not detect them.
- ✓ Character Encoding
  - Another easy method for the attacker is to encode the script tags (like <script>) because regex does not recognize such encoded characters.
- ✓ DOM-based XSS
  - Using regex for filtering is performed on the server side; however, this will not work as some XSS attacks can make it beyond the server side checks and start in the browser’s DOM.

Suggestions for Development

- ✓ Suggestions for Whitelist:

    - To fields such as name, it is recommended to set a relatively limited number of input types. Stable, predictable characters are admitted only because it guarantees the absence of instability problems with characters that can become a problem in the future.

- ✓ Coding of Outputs
    - If the user input on a webpage has to be allowed, then one should not format it in a way the browser may interpret as an application script. This terminology is known as output encoding. It is useful to have a concept of input encoding and output encoding as two extremes.

- ✓ Utilizing Security Libraries

    - One should access special libraries created for the purpose of XSS prevention such as OWASP Java Encoder. It supersedes simple regular expression patterns utilized when filtering most such libraries are proficient at cleansing input.

- ✓ Output Encoding with Context Awareness

    - If you want to remain tuned to the input without running the script, use the right encoding for the environment of the input which may include HTML, JavaScript and others.

It is, however, important to understand that even with the regex pattern providing basic input validation cross site scripting attacks could not be fully prevented. To solve this, in order to completely secure the program, one must incorporate input whitelisting, output encoding, and security libraries. Still higher protection from cross-site scripting attacks will be provided by this multiple-level security strategy.

Reference

'5 methods for Bypassing XSS Detection in WAFs' (2022) Volkis [Preprint]. https://www.volkis.com.au/blog/bypass-xss-in-wafs/.