

Ontrack Task4.1 Start Activity

Input validation is the process of ensuring that user-provided data meets the expected format, type, and constraints before further processing. It is an important step in building robust and secure software applications.

Launch the webapp activity

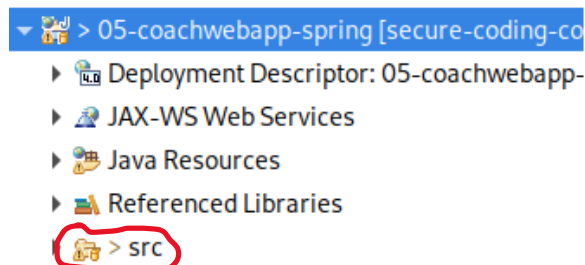
1. Start eclipse

```
cd /opt/eclipse/eclipse
```

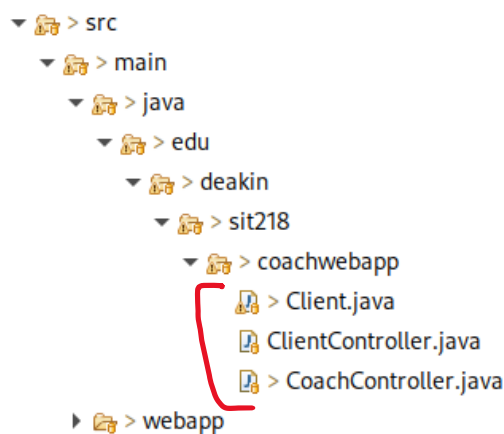
```
sudo ./eclipse
```

when prompted for password enter 'kali'

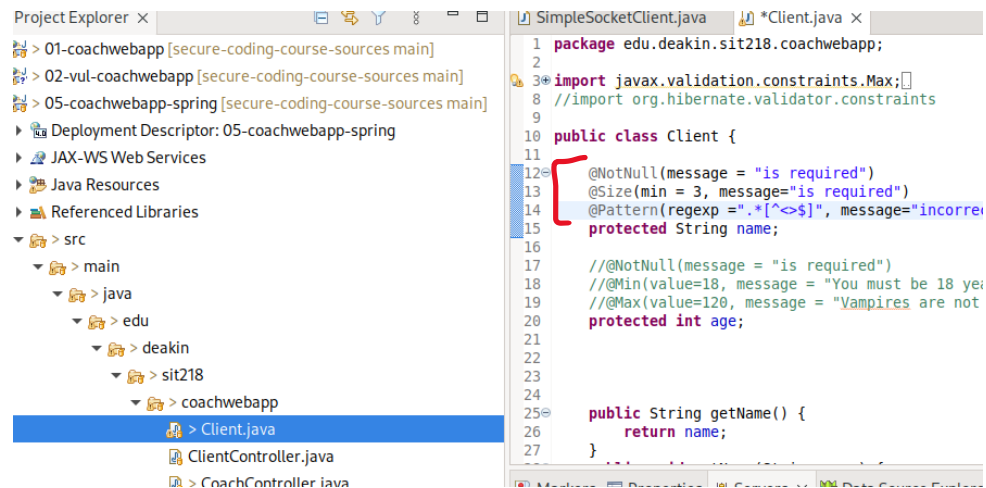
2. In the eclipse window expand the 05-coachwebapp-spring app from the project explorer



3. Now expand the src category until you see the .java files (client.java, ClientController.java and CoachController.java)



4. Double click on the Client.java and comment the lines (12,13,14) by adding a '//' before the line as highlighted below steps:

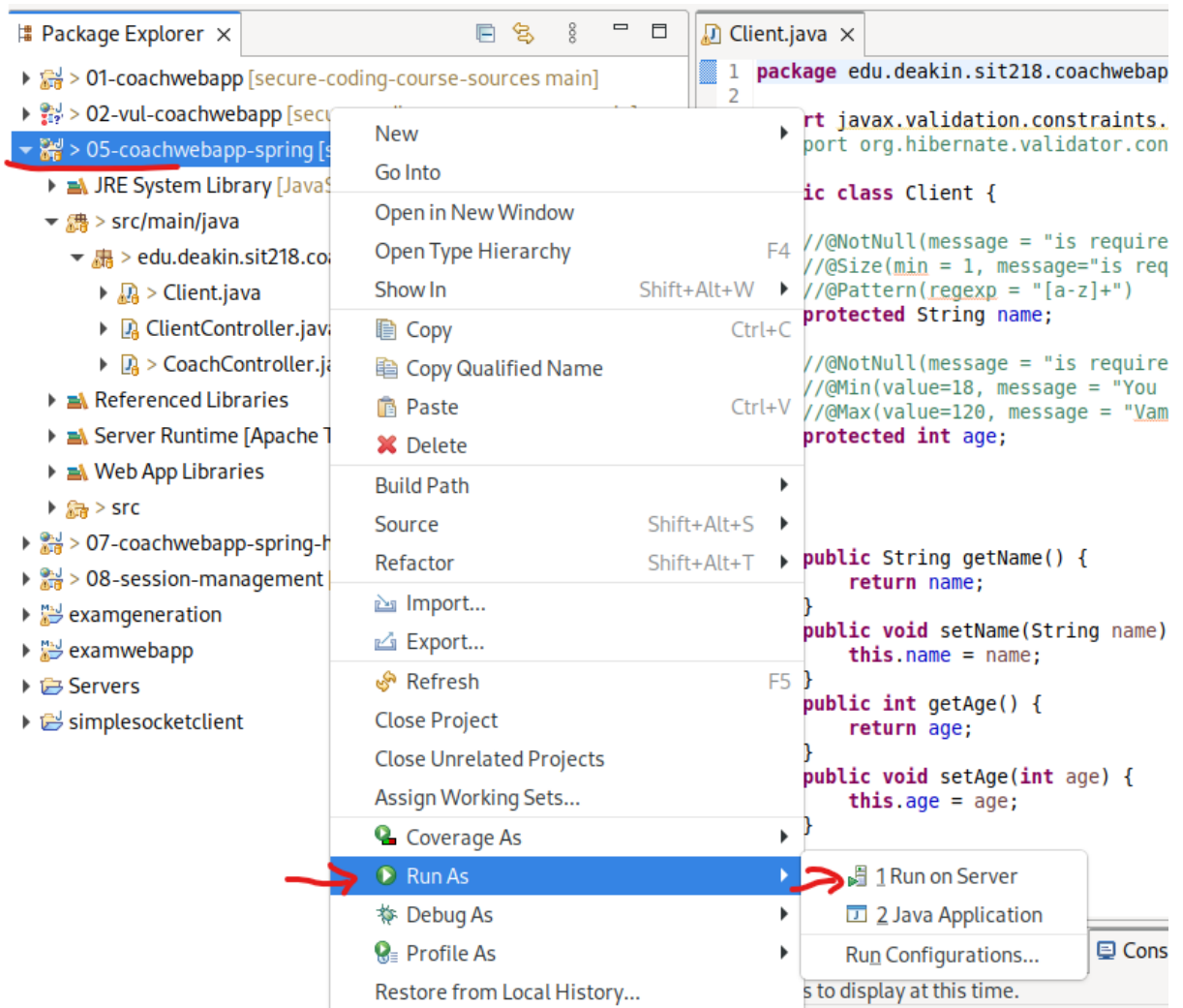


```

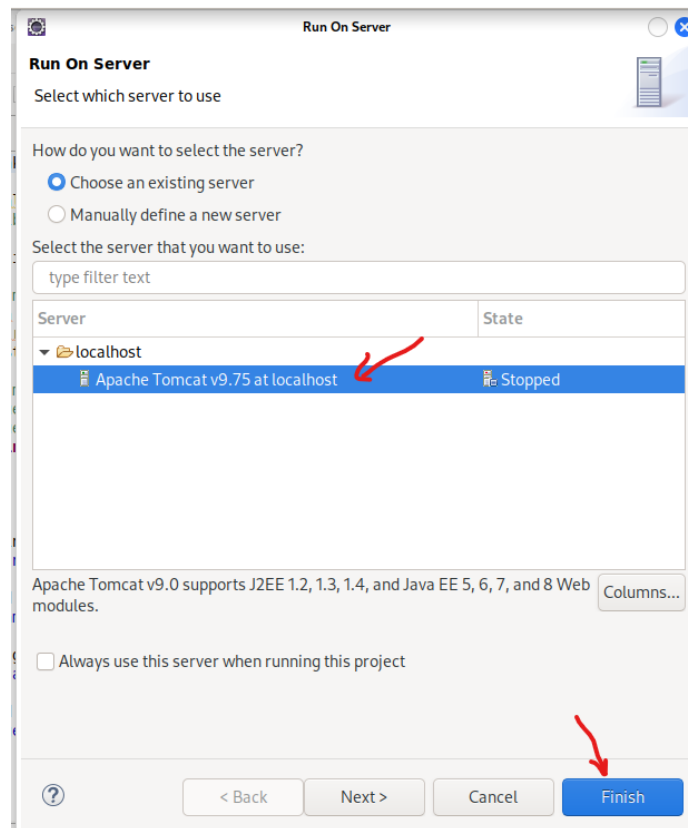
10 public class Client {
11
12 // @NotNull(message = "is required")
13 // @Size(min = 3, message="is required")
14 // @Pattern(regexp = ".*[<>]", message="incorrect format")
15 protected String name;
16
17 //@NotNull(message = "is required")
18 //@Min(value=18, message = "You must be 18 years old or older")
19 //@Max(value=120, message = "Vampires are not allowed")
20 protected int age;
21
22 }

```

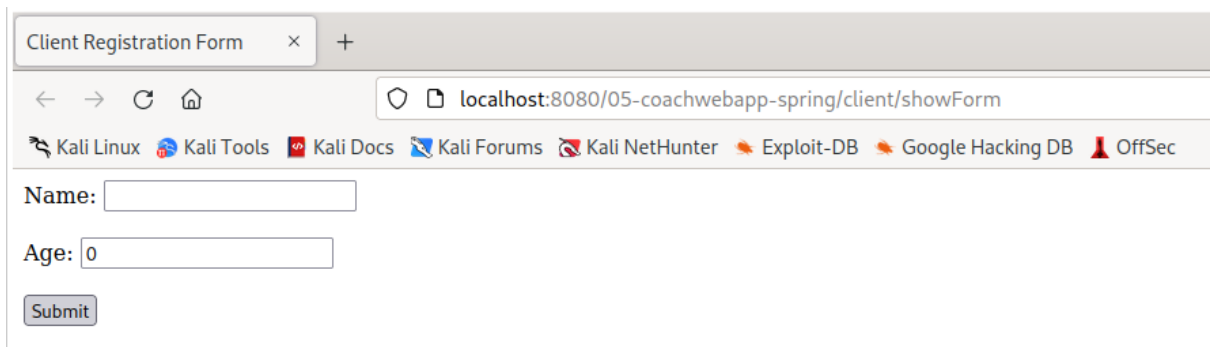
5. Save the file by pressing **ctrl+s** and right click on the **05-coachwebapp-spring** > select **Run As** and select > **Run on Server**



Select the Apache Tomcat server in the prompt that appears and click Finish.



This will launch the web app for this workshop.



Fuzz testing the input fields

Fuzz testing is a software testing technique that involves providing invalid, unexpected, or random data as inputs to a program in order to discover bugs, vulnerabilities, and other software defects. This will help us visualise the impact of applying our input validations in our webapp.

Activity-1:

Launch a separate terminal window.



Create a folder where we will save the random input values to the webapp.

mkdir wordlist

```
File Actions Edit View Help
(kali@kali)-[~]
$ mkdir wordlist
```

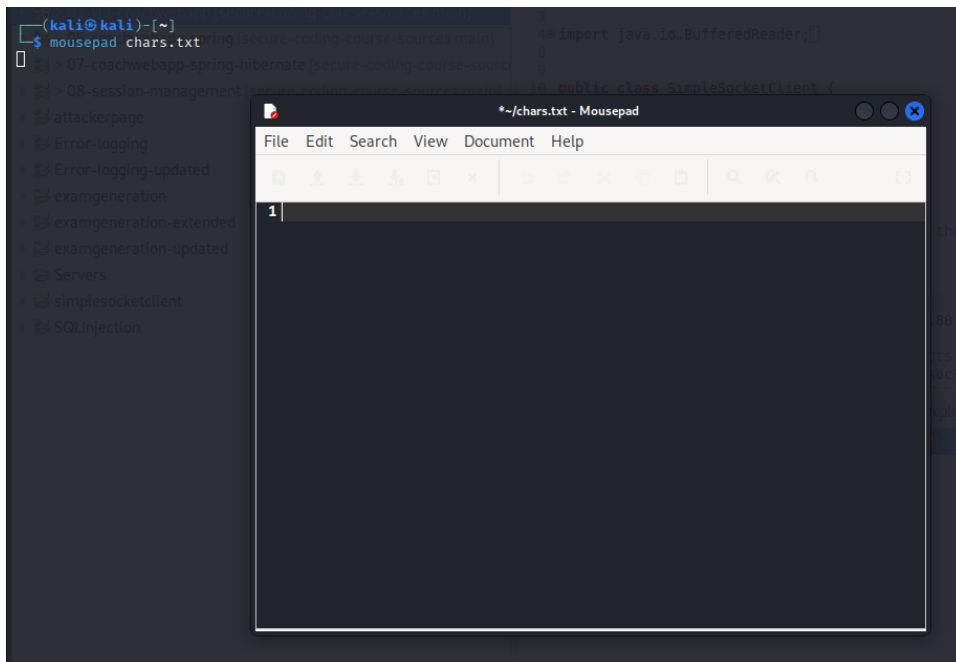
Type the following in the terminal window.

cd wordlist

```
File Actions Edit View Help
(kali@kali)-[~]
$ cd wordlist
(kali@kali)-[~/wordlist]
$
```


We will add some alphanumeric characters to test with the application. Create a new file using the command :

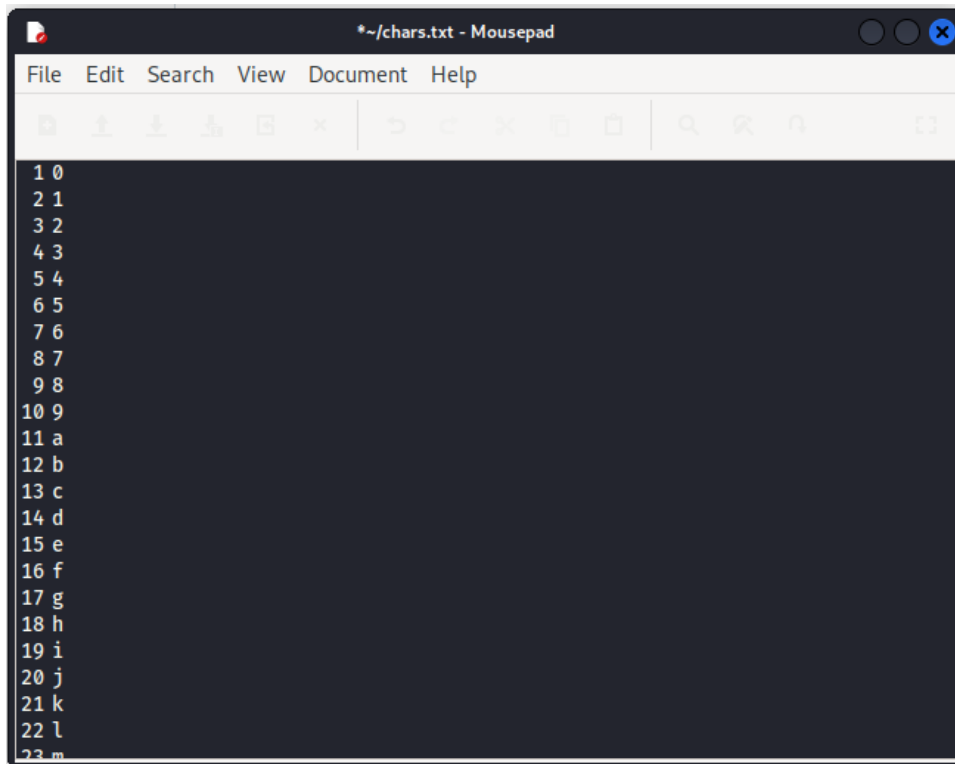
mousepad chars.txt



Copy the characters in the link

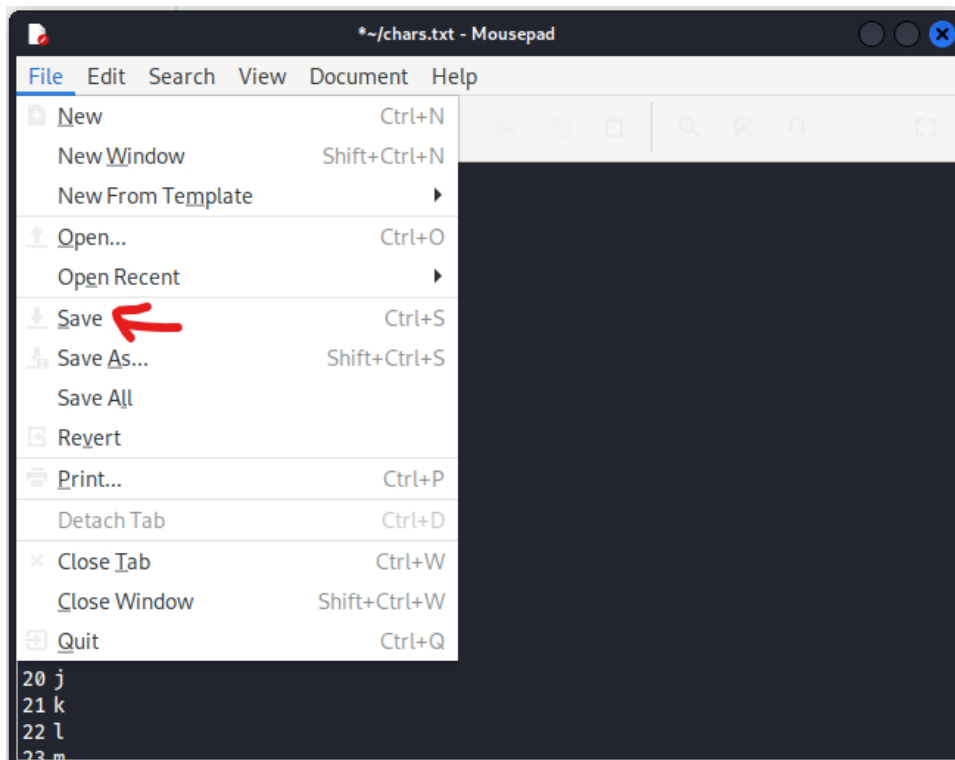
https://github.com/xmendez/wfuzz/blob/master/wordlist/stress/alphanum_case.txt

 xmendez Importing old wfuzz1.4c from google code	
1 contributor	
62 lines (62 sloc) 124 Bytes	
1	0
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	9
11	a
12	b
13	c
14	d
15	e
16	f
17	g
18	h
19	i
20	j



The screenshot shows a window titled "*~/chars.txt - Mousepad". The menu bar includes File, Edit, Search, View, Document, and Help. Below the menu bar is a toolbar with icons for file operations and editing. The main text area contains a list of characters, each preceded by a line number from 1 to 23:

```
1 0
2 1
3 2
4 3
5 4
6 5
7 6
8 7
9 8
10 9
11 a
12 b
13 c
14 d
15 e
16 f
17 g
18 h
19 i
20 j
21 k
22 l
23 m
```



The screenshot shows the same Mousepad window, but with the "File" menu open. The menu items are listed on the left, and their corresponding keyboard shortcuts are on the right. A red arrow points to the "Save" option, which has the shortcut "Ctrl+S".

File	Edit	Search	View	Document	Help
New	Ctrl+N				
New Window	Shift+Ctrl+N				
New From Template					
Open...	Ctrl+O				
Open Recent					
Save	Ctrl+S				
Save As...	Shift+Ctrl+S				
Save All					
Revert					
Print...	Ctrl+P				
Detach Tab	Ctrl+D				
Close Tab	Ctrl+W				
Close Window	Shift+Ctrl+W				
Quit	Ctrl+Q				

The text area below the menu shows the following content:

```
20 j
21 k
22 l
23 m
```

Activity-2:

Now we will test the coach app's two input fields using WFUZZ web application fuzzing tool

wfuzz -c -w chars.txt <http://localhost:8080/05-coachwebapp-spring/workout/processForm?name=FUZZ>

Press Enter

```
*****
* Wfuzz 3.1.0 - The Web Fuzzer *
*****

Target: http://localhost:8080/05-coachwebapp-spring/workout/processForm?name=FUZZ
Total requests: 62
```

ID	Response	Lines	Word	Chars	Payload
000000003:	200	11 L	31 W	225 Ch	"2"
000000013:	200	11 L	31 W	225 Ch	"c"
000000010:	200	11 L	31 W	225 Ch	"9"
000000007:	200	11 L	31 W	225 Ch	"6"
000000012:	200	11 L	31 W	225 Ch	"b"
000000015:	200	11 L	31 W	225 Ch	"e"
000000014:	200	11 L	31 W	225 Ch	"d"
000000009:	200	11 L	31 W	225 Ch	"8"
000000001:	200	11 L	31 W	225 Ch	"0"
000000006:	200	11 L	31 W	225 Ch	"5"
000000037:	200	11 L	31 W	225 Ch	"A"
000000018:	200	11 L	31 W	225 Ch	"h"
000000004:	200	11 L	31 W	225 Ch	"3"
000000005:	200	11 L	31 W	225 Ch	"4"
000000022:	200	11 L	31 W	225 Ch	"l"
000000011:	200	11 L	31 W	225 Ch	"a"
000000016:	200	11 L	31 W	225 Ch	"f"
000000008:	200	11 L	31 W	225 Ch	"7"
000000030:	200	11 L	31 W	225 Ch	"t"
000000002:	200	11 L	31 W	225 Ch	"1"
000000033:	200	11 L	31 W	225 Ch	"w"
000000059:	200	11 L	31 W	225 Ch	"W"
000000039:	200	11 L	31 W	225 Ch	"C"
000000032:	200	11 L	31 W	225 Ch	"v"
000000035:	200	11 L	31 W	225 Ch	"y"
000000028:	200	11 L	31 W	225 Ch	"r"
000000029:	200	11 L	31 W	225 Ch	"s"
000000036:	200	11 L	31 W	225 Ch	"z"
000000031:	200	11 L	31 W	225 Ch	"u"
000000027:	200	11 L	31 W	225 Ch	"q"

You can see the all the characters in our list (numbers and letters) are accepted in the name field. Lets try the age field by changing the FUZZ keyword in our wfuz command.

wfuzz -c -w chars.txt <http://localhost:8080/05-coachwebapp-spring/workout/processForm?age=FUZZ>


```

*****
* Wfuzz 3.1.0 - The Web Fuzzer *
*****

Target: http://localhost:8080/05-coachwebapp-spring/workout/processForm?age=FUZZ
Total requests: 62

=====
ID           Response    Lines   Word     Chars     Payload
=====
0000000007:  200           11 L     31 W     228 Ch    "6"
0000000003:  200           11 L     31 W     228 Ch    "2"
0000000001:  200           11 L     31 W     228 Ch    "0"
0000000028:  500           27 L    117 W    3441 Ch    "r"
0000000024:  500           27 L    117 W    3441 Ch    "n"
0000000025:  500           27 L    117 W    3441 Ch    "o"
0000000026:  500           27 L    117 W    3441 Ch    "p"
0000000027:  500           27 L    117 W    3441 Ch    "q"
0000000015:  500           27 L    117 W    3441 Ch    "e"
0000000021:  500           27 L    117 W    3441 Ch    "k"
0000000032:  500           27 L    117 W    3441 Ch    "v"
0000000031:  500           27 L    117 W    3441 Ch    "u"
0000000030:  500           27 L    117 W    3441 Ch    "t"
0000000029:  500           27 L    117 W    3441 Ch    "s"
0000000023:  500           27 L    117 W    3441 Ch    "m"
0000000022:  500           27 L    117 W    3441 Ch    "l"
0000000018:  500           26 L    116 W    3326 Ch    "h"
0000000020:  500           26 L    116 W    3326 Ch    "j"
0000000012:  500           26 L    116 W    3326 Ch    "b"
0000000019:  500           26 L    116 W    3326 Ch    "i"
0000000016:  500           26 L    116 W    3326 Ch    "f"
0000000014:  500           26 L    116 W    3326 Ch    "d"
0000000011:  500           26 L    116 W    3326 Ch    "a"
0000000017:  500           26 L    116 W    3326 Ch    "g"
0000000013:  500           26 L    116 W    3326 Ch    "c"
0000000009:  200           11 L     31 W     228 Ch    "8"
0000000033:  500           26 L    116 W    3326 Ch    "w"
0000000002:  200           11 L     31 W     228 Ch    "1"
0000000005:  200           11 L     31 W     228 Ch    "4"

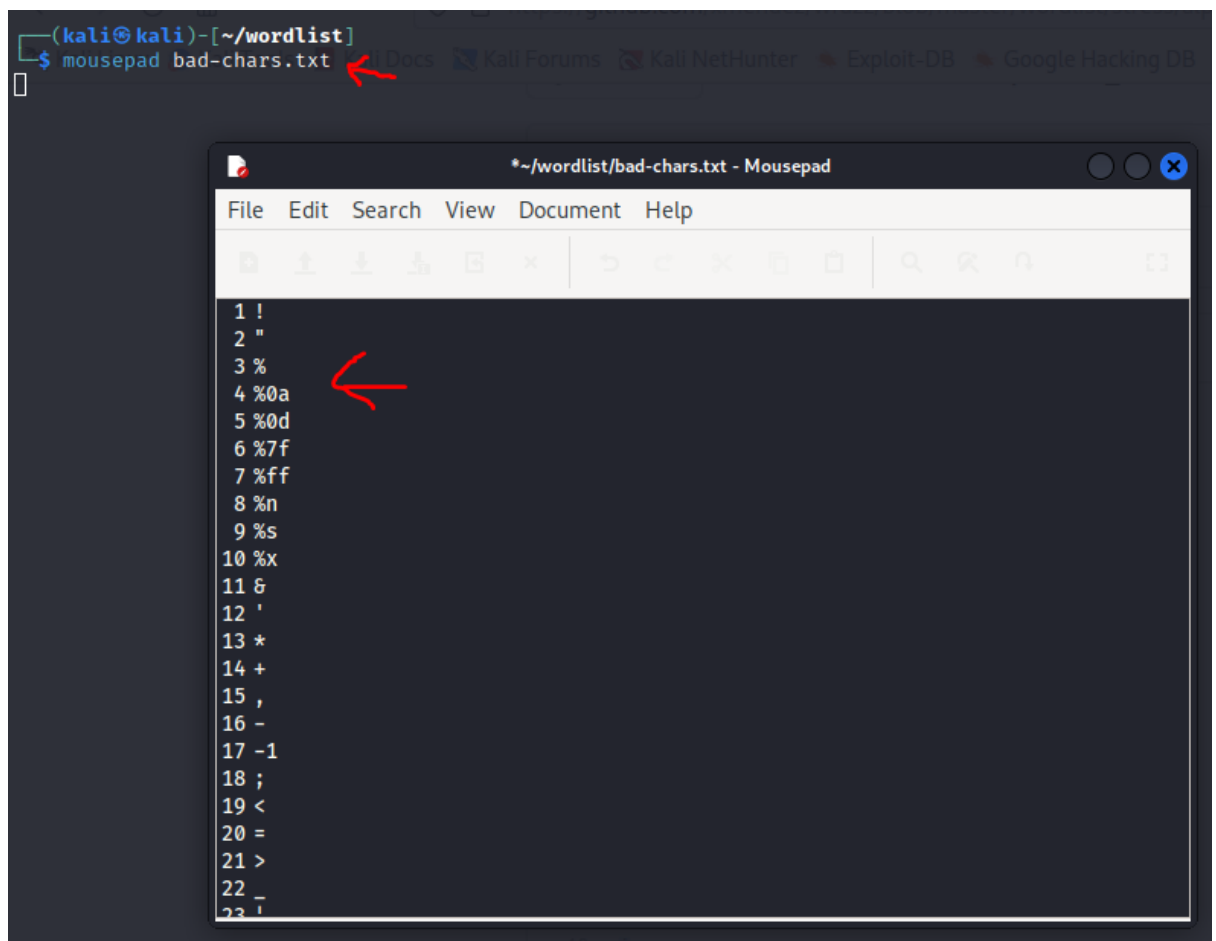
```

You can see the response received from the server is 500 which means the server responded with internal server error which indicates it doesn't accept letters in the "age" field.

Activity-3:

Create another file named bad-chars.txt using the mousepad command used previously and add the following characters to that file

```
!  
"  
%  
%0a  
%0d  
%7f  
%ff  
%n  
%s  
%x  
&  
'  
*  
+  
,  
-  
-1  
;  
<  
=  
>  
_  
!
```



The screenshot shows a Kali Linux terminal window with the command `mousepad bad-chars.txt` executed. A red arrow points to the command. Below the terminal, a Mousepad window titled `*~/wordlist/bad-chars.txt - Mousepad` is open, displaying the contents of the file. The file contains a list of characters and escape sequences, numbered 1 through 23. A red arrow points to the 4th line, which is `4 %0a`.

```
(kali㉿kali)-[~/wordlist]  
$ mousepad bad-chars.txt  
  
1 !  
2 "  
3 %  
4 %0a  
5 %0d  
6 %7f  
7 %ff  
8 %n  
9 %s  
10 %x  
11 &  
12 '  
13 *  
14 +  
15 ,  
16 -  
17 -1  
18 ;  
19 <  
20 =  
21 >  
22 _  
23 !
```

Repeat fuzz process by changing the wordlist to “bad-chars.txt” and report your response from the server.

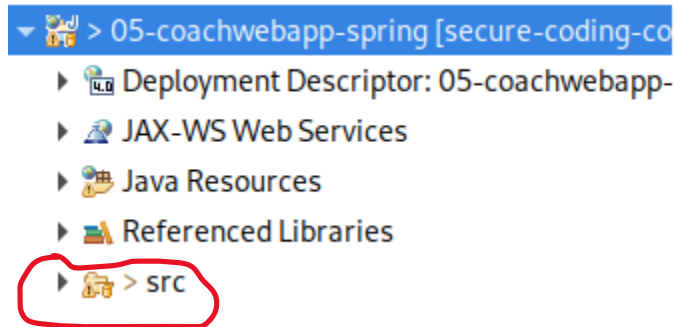
wfuzz -c -w bad-chars.txt <http://localhost:8080/05-coachwebapp-spring/workout/processForm?name=FUZZ>

wfuzz -c -w bad-chars.txt <http://localhost:8080/05-coachwebapp-spring/workout/processForm?age=FUZZ>

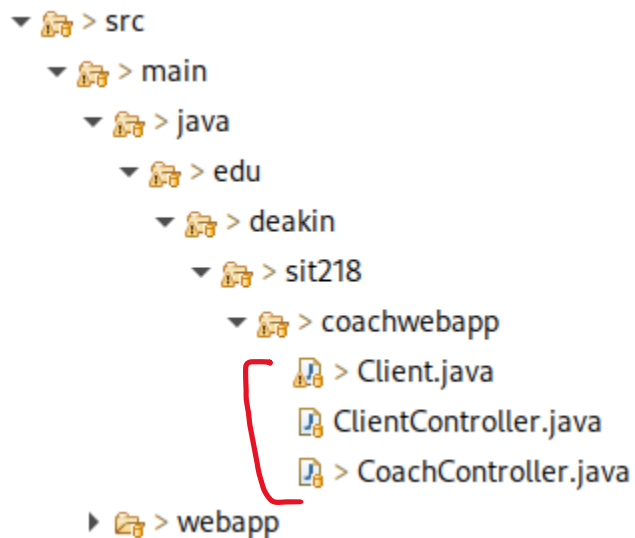
Adding Input validations

Now let's enable few validations in our web app fields

In the eclipse window expand the 05-coachwebapp-spring app from the project explorer



Now expand the src category until you see the .java files (client.java, ClientController.java and CoachController.java)



Now double click on the Client.java which is the bean which will be associated with the name and age fields on the webpage.

```

SimpleSocketClient.java Client.java x
1 package edu.deakin.sit218.coachwebapp;
2
3 import javax.validation.constraints.Max;
4 //import org.hibernate.validator.constraints
5
6 public class Client {
7
8 // @NotNull(message = "is required")
9 // @Size(min = 3, message="is required")
10 // @Pattern(regexp = ".*[<>]", message="incorrect format")
11 protected String name;
12
13 //@NotNull(message = "is required")
14 //@Min(value=18, message = "You must be 18 years old or older")
15 //@Max(value=120, message = "Vampires are not allowed")
16 protected int age;
17
18
19
20
21
22

```

For the name field we need to enable some validations that will enforce the web app to accept only valid content and if validations fails, error will displayed to the user. Delete the “//” in the @NotNull and @Size lines of the code.

@NotNull(message = "is required") - this means this field cannot be left blank.

@Size(min = 3, message="is required") - this field means the size of this field must be minimum three characters long.

After editing you should see these lines enabled as show below:

```

4
5
6
7
8 import javax.validation.constraints.Max;
9 //import org.hibernate.validator.constraints
10
11 public class Client {
12
13 @NotNull(message = "is required")
14 @Size(min = 3, message="is required")
15 // @Pattern(regexp = ".*[<>]", message="incorrect format")
16 protected String name;
17
18
19
20
21
22

```

Press ctrl+s and save the program. Wait for a couple of seconds before the updated app is published to the tomcat server by eclipse. (**Note:** if your tomcat server is not already running, then right click on the webapp in the project explorer and run in on the server). You will see the following messages in the eclipse console.

```

Markers Properties Servers Data Source Explorer Snippets Terminal Console x
Apache Tomcat v9.7.5 at localhost [Apache Tomcat] /opt/eclipse/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17
Initializing Servlet 'dispatcher'
, 2023 2:23:02 AM org.hibernate.validator.internal.util.Version <clinit>
HV000001: Hibernate Validator 6.2.0.Final
, 2023 2:23:02 AM org.springframework.web.servlet.FrameworkServlet initServletBean
Completed initialization in 373 ms
, 2023 2:23:02 AM org.apache.catalina.core.StandardContext reload
Reloading Context with name [/05-coachwebapp-spring] is completed

```

Now navigate back to the terminal where we were using the wfuzz tool

Now create a new text file "common.txt" in the /wordlist folder using the mousepad command and add the following characters in it

```
@
00
01
02
03
1
10
100
1000
123
2
20
200
2000
2001
2002
2003
2004
2005
3
a
aa
aaa
certificates
certs
cfdocs
cfg
cgi
cgibin
cgi-bin
cgi-win
chan
change
changepw
channel
chart
chat
class
classes
classic
classified
classifieds
client
clients
cluster
cm
cmd
```

Now test the webapp with the common.txt wordlist and notice the response code.

```
wfuzz -c -w common.txt http://localhost:8080/05-coachwebapp-
spring/workout/processForm?name=FUZZ
```

```

--$ wfuzz -c -w common.txt http://localhost:8080/05-coachwebapp-spring/workout/processForm?name=FUZZ
/usr/lib/python3/dist-packages/wfuzz/__init__.py:34: UserWarning:Pycurl is not compiled against Openssl. Wfuzz might not work correctly when fuzzing SSL sites. Check Wfuzz's documentation for more information.
*****
* Wfuzz 3.1.0 - The Web Fuzzer
*****

Target: http://localhost:8080/05-coachwebapp-spring/workout/processForm?name=FUZZ
Total requests: 47

ID      Response  Lines  Word  Chars  Payload
-----
000000001: 500        26 L   116 W   3326 Ch  "@"
000000015: 200        11 L    31 W    228 Ch  "2001"
000000029: 200        11 L    31 W    230 Ch  "cgibin"
000000028: 200        11 L    31 W    227 Ch  "cgi"
000000027: 200        11 L    31 W    227 Ch  "cifs"
000000026: 200        11 L    31 W    230 Ch  "cfdocs"
000000025: 200        11 L    31 W    229 Ch  "certs"
000000023: 200        11 L    31 W    227 Ch  "aaa"
000000019: 200        11 L    31 W    228 Ch  "2005"
000000018: 200        11 L    31 W    228 Ch  "2004"
000000003: 500        26 L   116 W   3326 Ch  "01"
000000007: 500        26 L   116 W   3326 Ch  "10"
000000024: 200        11 L    31 W    236 Ch  "certificates"
000000022: 500        26 L   116 W   3326 Ch  "aa"

```

Now you should see 500 error response for input values that are less than 3 characters.

Test script injection

Activity-1:

Now let's check if we can perform an injection attack as done in previous workshop.

Launch the webapp using the URL <http://localhost:8080/05-coachwebapp-spring/client/showForm> and enter the following value into the name field and click submit.

Client Registration Form

localhost:8080/05-coachwebapp-spring/client/showForm

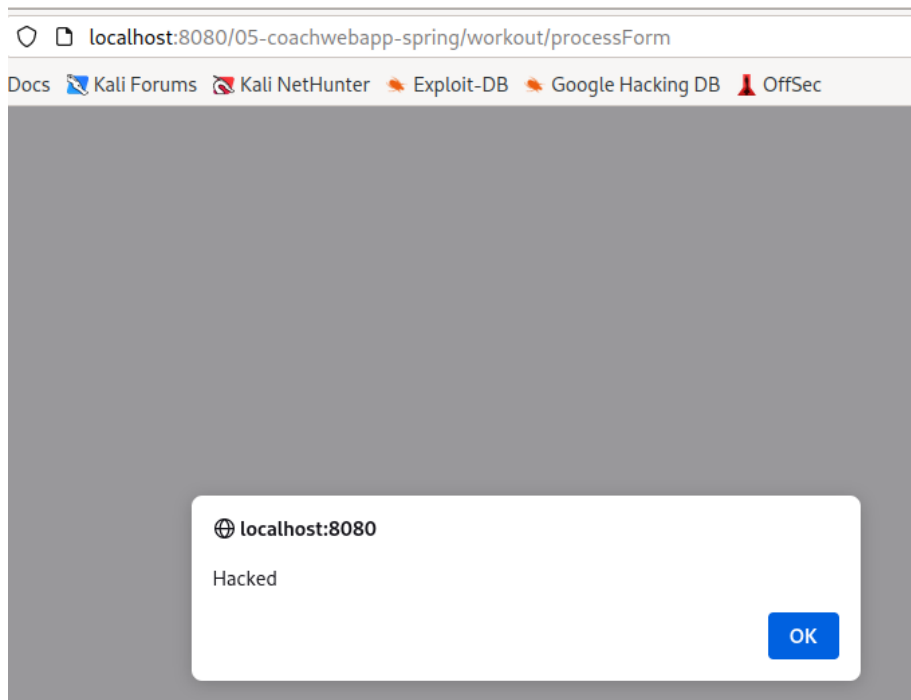
Kali Linux Kali Tools Kali Docs Kali Forums Kali NetHunter Exploit-DB Google Hacking

Name:

Age:

Submit

As you can see you can see you will still be able to pass scripts to the web app.



Activity-2:

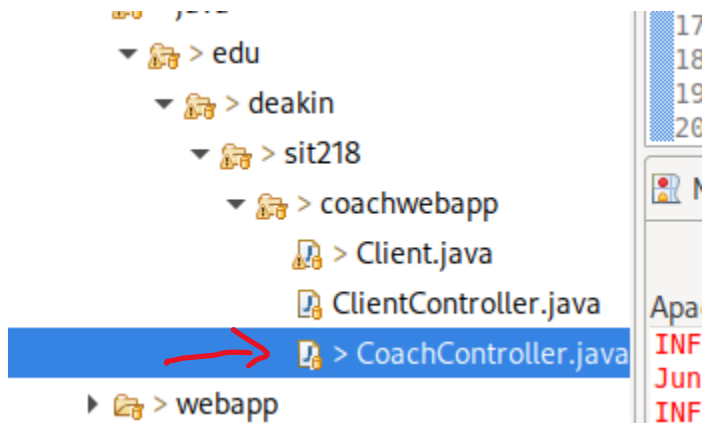
Now to prevent any input that contains these characters '<>' used in the script tags, we will use regular expressions to block them. Now go back to the Client.java code and enable the following line above the name field by removing the '//'.

@Pattern(regexp = ".*[^<>]", message="incorrect format")

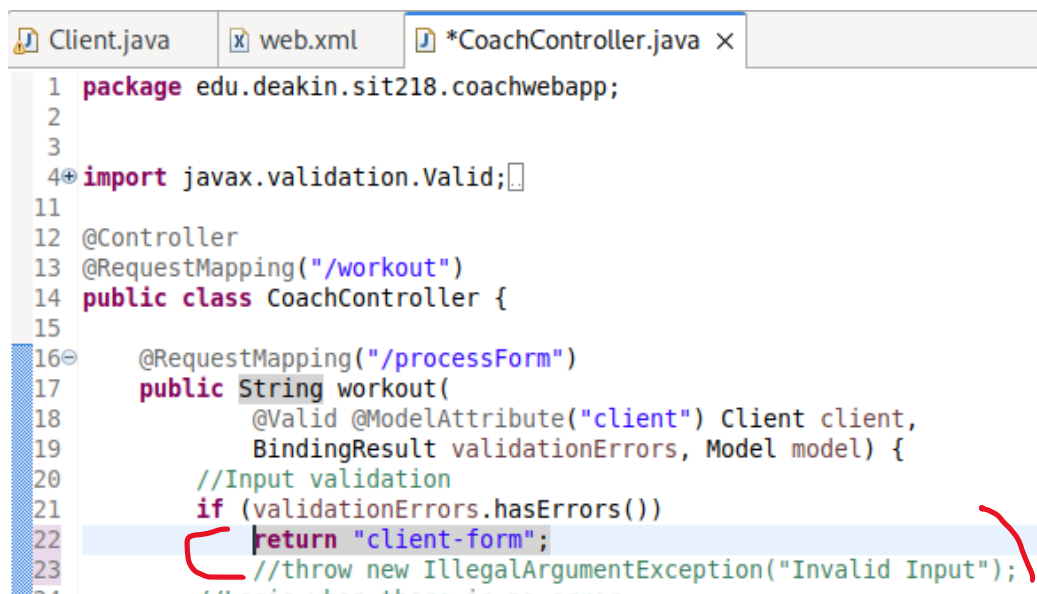
```
public class Client {  
  
    @NotNull(message = "is required")  
    @Size(min = 3, message="is required")  
    → @Pattern(regexp = ".*[^<>]", message="incorrect format")  
    protected String name;  
  
    //@NotNull(message = "is required")  
    //@Min(value=18, message = "You must be 18 years old or older")  
    //@Max(value=120, message = "Vampires are not allowed")  
    protected int age;
```

Once again save the code and now open the CoachController.java file by double clicking it in the project explorer.

This regex allows normal text input but prevents injected script that ends with a '>' character. Understand the behaviour and its explanation of this regex expression at <https://regex101.com/> with Java8 as the language.



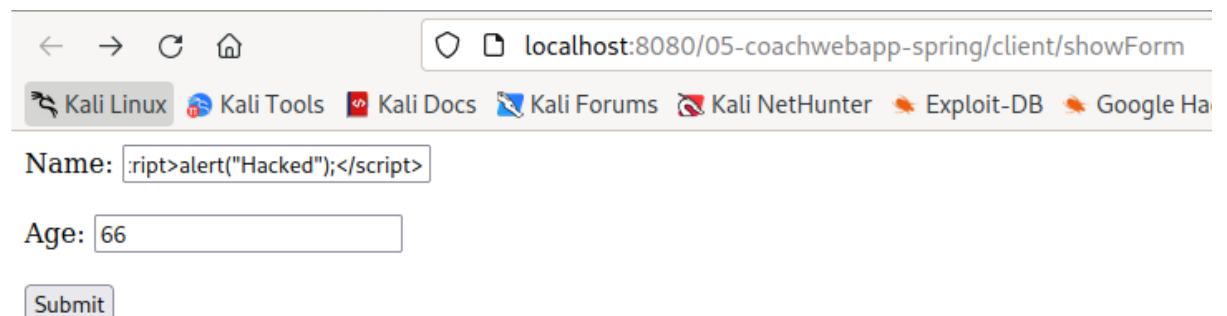
Comment the lines which says “throw new IllegalArgumentException(“Invalid Input”);” and un-comment the line which says “return “client-form”” as shown below:



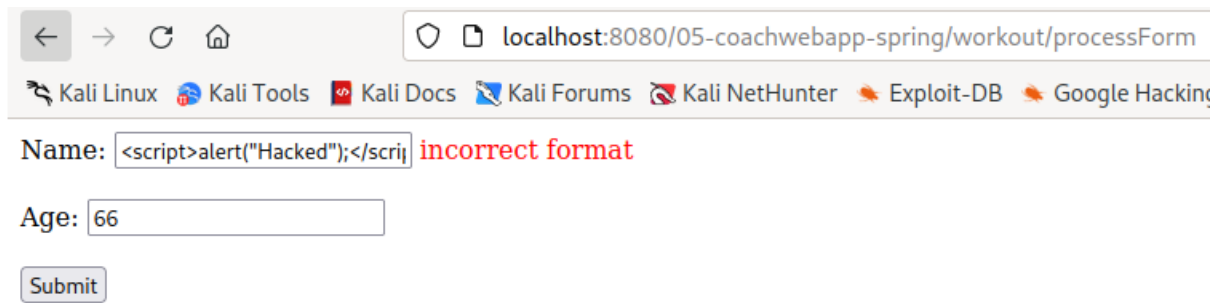
Save the code and wait for few seconds before the app is published to tomcat.

Now launch the app using the url: <http://localhost:8080/05-coachwebapp-spring/client/showForm>

Now re-enter the script we inject before: (<script>alert("Hacked");</script>) and click submit.



Now you should see the below error:



The screenshot shows a web browser window with the address bar displaying `localhost:8080/05-coachwebapp-spring/workout/processForm`. The browser's tab bar includes links to Kali Linux, Kali Tools, Kali Docs, Kali Forums, Kali NetHunter, Exploit-DB, and Google Hacking. The form has two input fields: 'Name:' and 'Age:'. The 'Name:' field contains the text `<script>alert("Hacked");</script>` and has a red error message 'incorrect format' next to it. The 'Age:' field contains the number '66'. Below the fields is a 'Submit' button.

you have now blocked the user from entering the script tags. Try entering normal text input you should see it going through.

XSS bypass

Even though you have added some validations, sometimes these filters can be bypassed using more clever inputs that do not get caught in the filters.

Activity-1: Try to enter the below payloads in the name field of your web app **Test script injection (Activity-2)** above.

- 1) name "<script>alert("XSS"); </script>"
- 2) <a aa aaa aaaa aaaaa aaaaaa aaaaaaa aaaaaaaaa aaaaaaaaaa aaaaaaaaaa href=javascript:alert(1)>ClickMe
- 3) <script x> alert(1) </script 1=2

Show your outputs in your task submission and explain why these are getting bypassed.

Whitelisting vs Blacklisting

Read the discussion about whitelisting and blacklisting in the Lecture notes as well as <https://owasp-top-10-proactive-controls-2018.readthedocs.io/en/latest/c5-validate-all-inputs.html>

Reflect on the validations we just created are they whitelisting or blacklisting?

Server-side vs client-side validations

The validations we did so far are server side and validations are performed after the request is received by the server. There are techniques that allow performing DOM-based XSS which bypasses the server-side validations. See some examples of how server-side validations can be evaded from <http://www.webappsec.org/projects/articles/071105.shtml>

This necessitates the application of client-side validations. HTML provides methods to implement client-side validations as explained here https://developer.mozilla.org/en-US/docs/Learn/Forms/Form_validation