Spotify - Listening to Music in the workplace...

Or an educational clickbait to deepdive into OAuth and DPoP (RFC 9449)...

RFC 9449: OAuth 2.0 Demonstrating Proof of Possession (DPoP)

Introduction

> Research shows that cranking up your favorite tunes at work isn't just about making the day more enjoyable, it can actually boost your performance and job satisfaction. The key? Use music intentionally to manage your mood, rather than just as background noise. A 2022 study of 244 workers found that those who actively used music for emotional regulation performed better than those who just had it playing in the background. So before you hit play on that office playlist, think about how you want the music to work for you.
>
> source: "Don't Stop the Music," Please: The Relationship between Music Use at Work, Satisfaction, and Performance

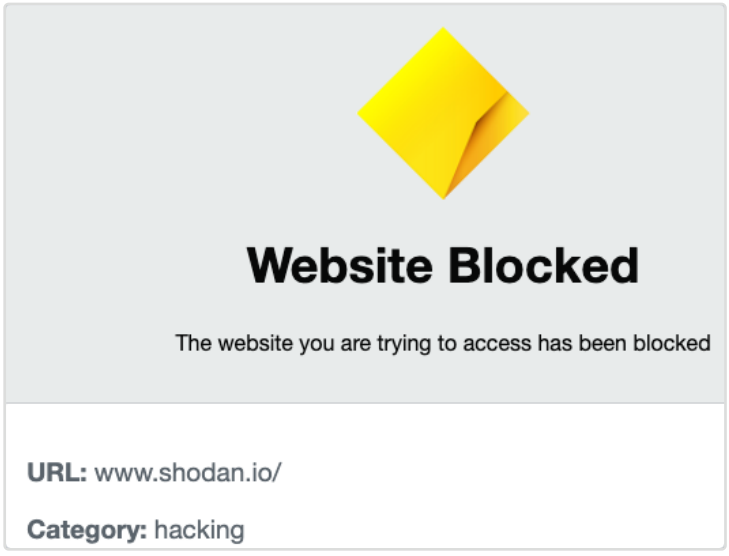
While you could plug in your headphones to your phone, having Spotify directly on your workstation eliminates the constant switching between devices, reduces distractions, and prevents productivity loss from managing multiple devices. It's a simple solution that keeps you in your workflow while enjoying the performance boosting benefits of music.

However, getting Spotify to work on SOE workstations is surprisingly challenging and users can encounter several roadblocks:

- File download appear to be complete but the filesize is is 0KB (corrupted)
- Stub installer failing to retrieve packages (network issues)
- Application control blocking the installation/executable
- Authentication failures after successful installation

Interestingly, it's unclear whether these barriers are intentional security measures or unintended technical issues. If this was a deliberate security decision (like preventing supply chain attacks through malicious updates) or a bandwidth conservation effort, we'd expect to see:

- A clear IT/security policy addressing streaming services
- Or at minimum, split-tunneling implementation to handle streaming traffic efficiently outside the VPN
- Explicit blocking messages from Prisma like this one on `*.spotify.com`:



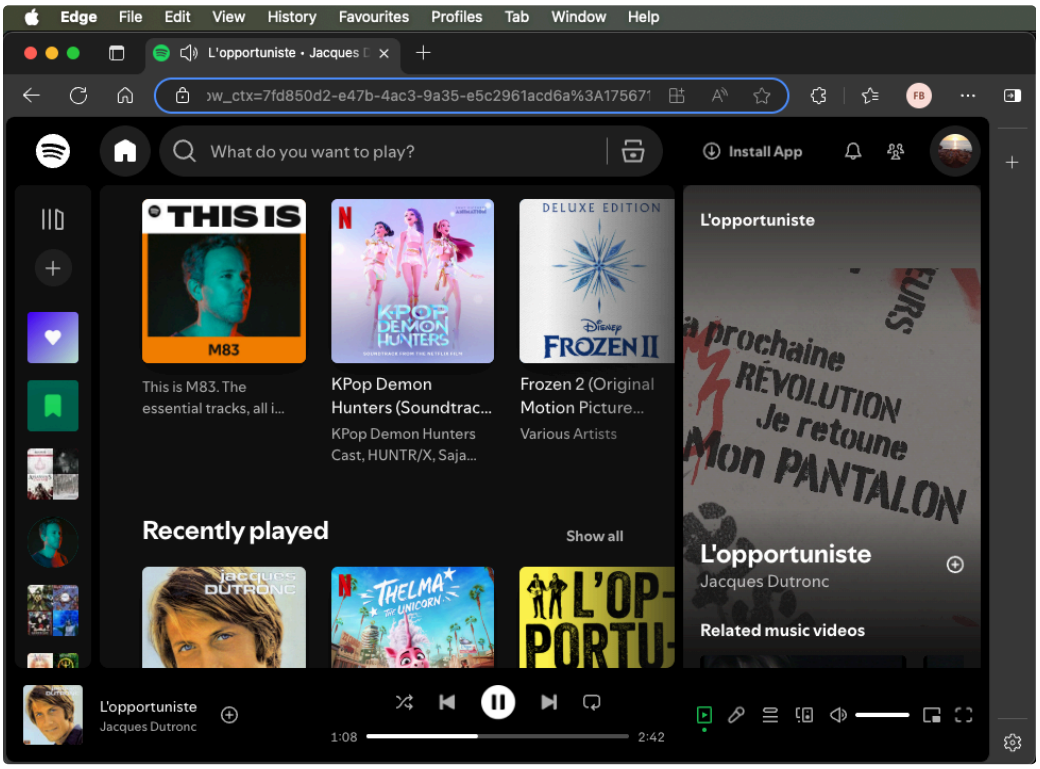
The absence of these standard practices suggests these might be technical hurdles we can overcome rather than intentional restrictions...

First Method: Web Player

Complexity: **Easy**

Via the Browser

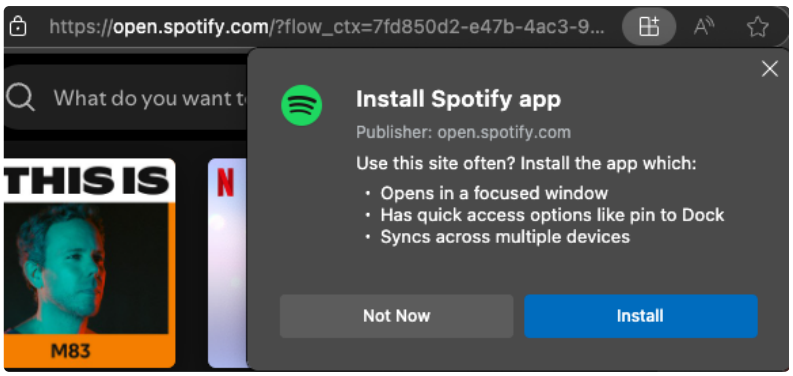
Open Spotify - Web Player: Music for everyone and authenticate

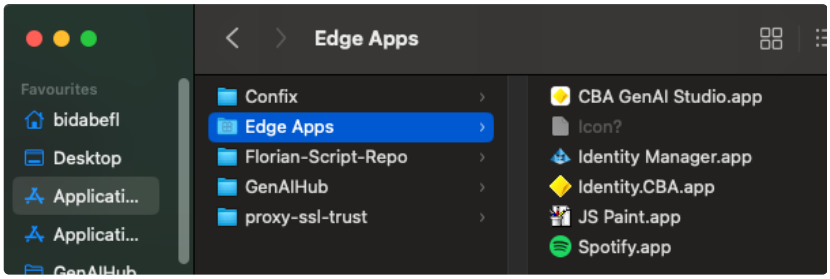


Guard that phone PIN better! The kids have been on a musical spree. My account is now a delightful mix of K-pop Daemon Hunters, Frozen, and Thelma the Unicorn. Parent life...

Standalone WebApp

Create a standalone application if you wish from Chromium based browsers...





It'll be stored in `~/Applications` (or `%userprofile%\AppData` I suppose...)

You can then pin the App to the Dock / Menu Bar and/or access it via Finder (cmd+space) or Explorer (Windows key) Search

Second Method: thick client

> Don't close this page yet, if you're after some practical learnings on Oauth and DPoP...

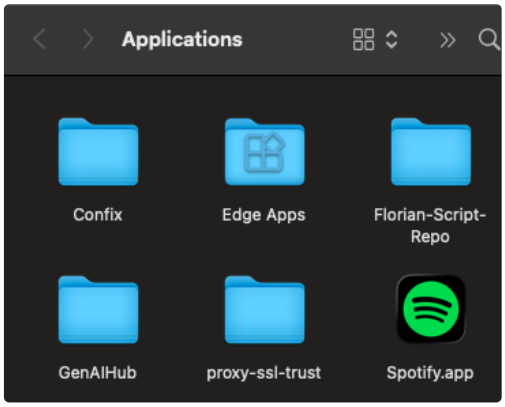
Complexity: **Medium/Hard**

Downloading Spotify

Nothing much to expect here, I believe the installer will download...
Hopefully it'll be more than 0KB

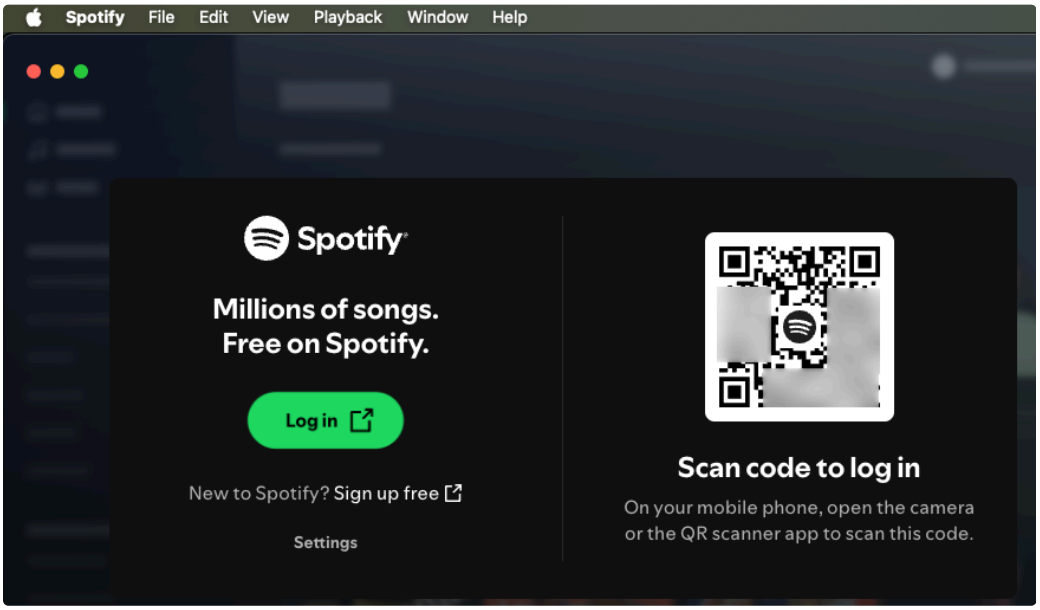
Installing it

This should be relatively straight forward On MacOS, but on Windows, you might need Application Control Exception. Note the installer is a stub, it also needs to download packages to install the app.
Note, the application is running in the User Context (not system wide)... and that's perfect like that because it means it does not require **nor have** administrative privileges.

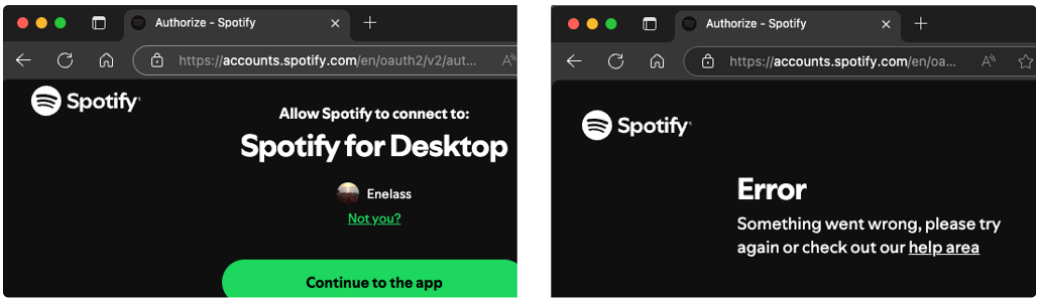


Sign in

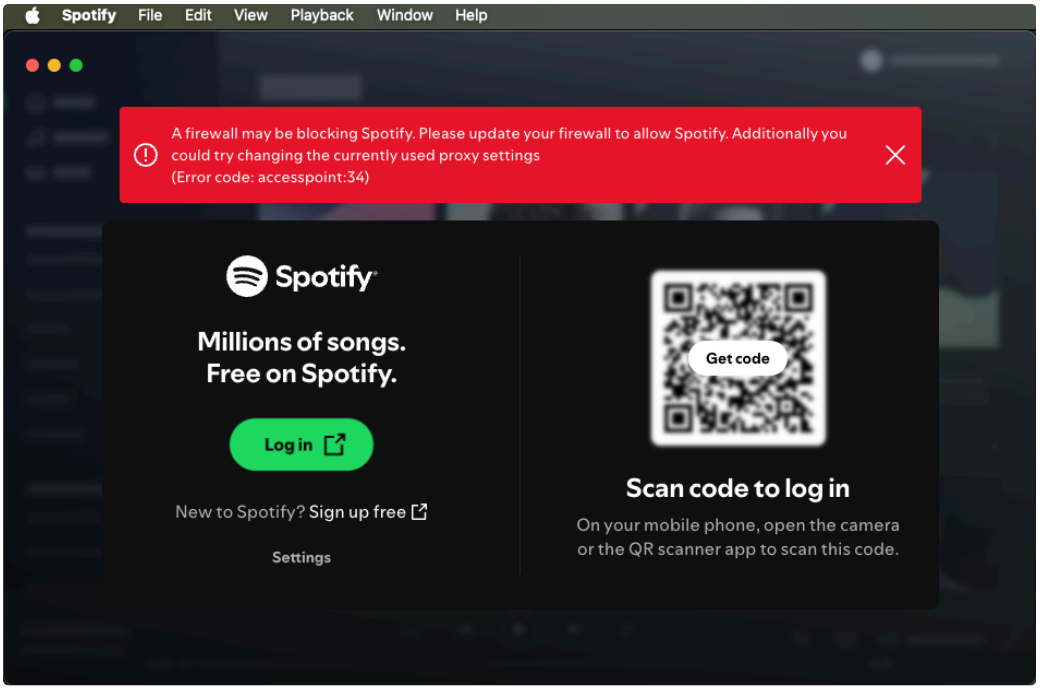
The problem (front-end)



Authenticate using whichever methods results in...



Oauth flow via the SOE ends with an error...



Oauth flow via the QR code (phone) times out and errors out...

The problem (behind the scenes...)

You'll notice the thick client will not let you authenticate...

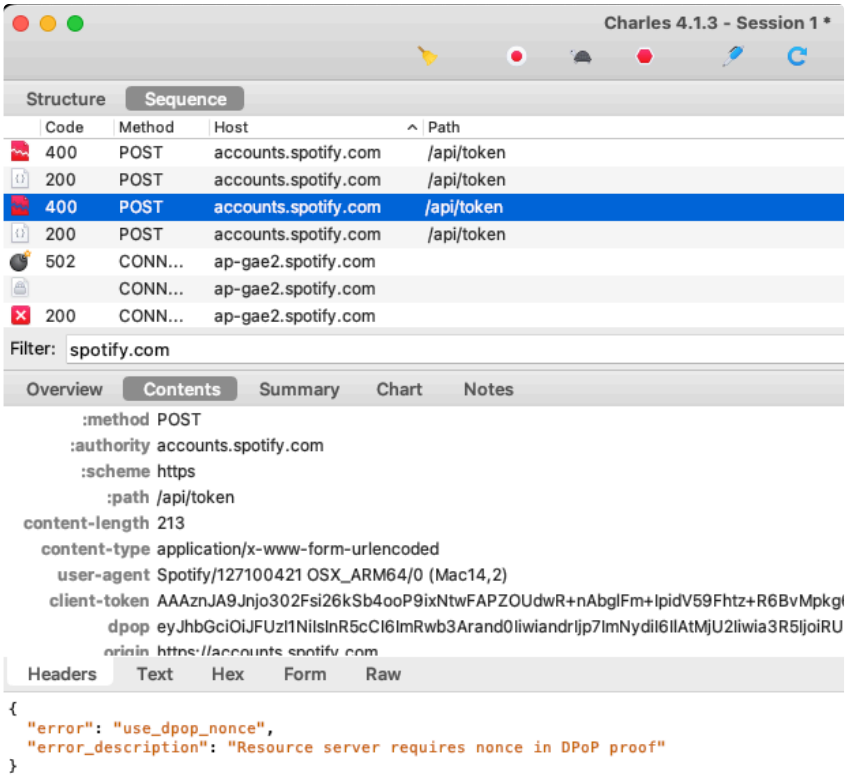
The app support the setting of proxy, so pointing it to Charles (Web Debugging) for further troubleshooting to see what happend behind the scenes reveal a bunch of HTTP400, 502 and 503 Errors... whether using `proxy.au.eds.com:8080` or `cba.proxy.prismaaccess.com:8080`.

If of interest, see this for more information on proxies...

CLI - Why you should not set CNTLM nor Prisma

& Proxy and SSL Certificate issues

& Setting up Alpaca on Windows / Setting up Alpaca on MacOS



A DPoP (Demonstrating Proof of Possession) error indicating the server requires a nonce (one-time number) in the authentication process. This is a security measure to prevent replay attacks by ensuring each request is unique and hasn't been copied from a previous session. To resolve, you'll need to first obtain a nonce from the server and include it in your subsequent requests.

Problem Hypothesis...

SSL/TLS inspection (also known as HTTPS interception or SSL forward proxy) can break DPoP authentication because:

1. DPoP relies on cryptographic proofs tied to the original TLS connection
2. When SSL inspection is in place, it creates two separate TLS connections (client-to-proxy and proxy-to-server)
3. This breaks the chain of trust DPoP tries to establish

In essence, the proxy may inadvertently change the cryptographic properties of the connection that DPoP is verifying. And just like Certificate Pinning... DPoP does not support SSL inspection!

While SSL Forward decryption exception cannot be made at the SOE level (but on PanOS or Prisma Cloud Control plane...), we can delegate the authenticatioin to a device where SSL forward inspection is not at plat: your smartphone...

Attempt Resolution #1

Since the Auth won't succeed on the SOE, let's try to delegate it to another device where the connection to server is not intercepted or tampered with... my mobile phone!

Generate a OAuth GET request

A GET request is generated after clicking Login on Spotify, it opens the browser...

Intercept that GET Request and close the tab before it loads...

Request parameters

```
Base URL: https://accounts.spotify.com/oauth2/v2/auth

Parameters:
- client_id=65b708073f**********************87bd
- response_type=code (Authorisation Code Flow)
- redirect_uri=http://127.0.0.1:4381/login
- scope=(multiple Spotify API permissions)
- code_challenge_method=S256 (PKCE)
- code_challenge=UOQs01wJ95************************7nN2I
- flow_ctx=e4a7ab71-****-****-****-d03ff4032fa5:1756722825
- dpop_jkt=WlIrLvT************************CoObMYdo
- utm_source=spotify
- utm_medium=desktop-macos
- utm_campaign=organic
- creation_flow=desktop
- creation_point=https://login.app.spotify.com/
```

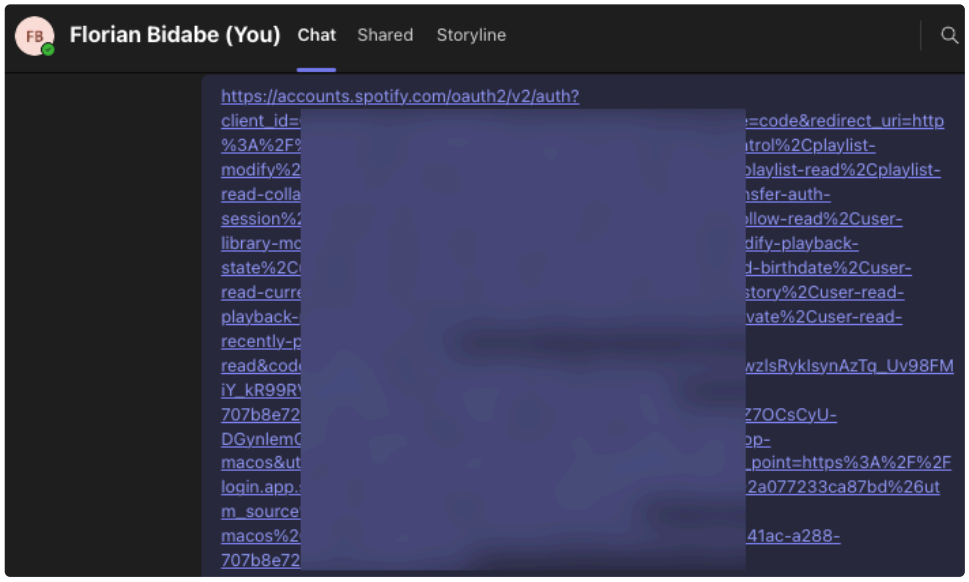
Request Analysis

Analysis of each secret/sensitive value in the authorisation request

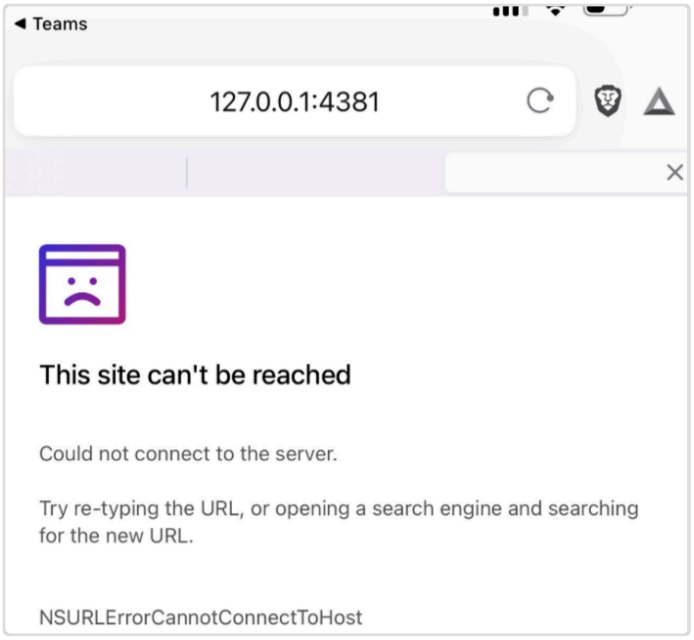
```
1. client_id=65b708073f**********************87bd
   - Length: 32 characters
   - Format: Hexadecimal string
   - Purpose: Spotify application identifier
   - Sensitivity: Public but unique to app registration

2. code_challenge=UOQs01wJ95************************7nN2I
   - Length: 43 characters
   - Format: Base64URL-encoded SHA256 hash
   - Purpose: PKCE challenge derived from code_verifier
   - Generation: BASE64URL-ENCODE(SHA256(code_verifier))
   - Sensitivity: Public but cryptographically bound to private code_verifier

3. flow_ctx=e4a7ab71-****-****-****-d03ff4032fa5:1756722825
   - Format: UUID v4 + timestamp
   - Structure: {UUID}:{timestamp}
   - Purpose: Session tracking and flow state
   - Sensitivity: Session-specific identifier

4. dpop_jkt=WlIrLvTz************************CoObMYdo
   - Length: 43 characters
   - Format: Base64URL-encoded JWK Thumbprint
   - Purpose: DPoP public key binding
   - Generation: BASE64URL(SHA256(JWK))
   - Sensitivity: Public key thumbprint, cryptographically bound to private key
```

Copy that GET request from the SOE browser after clicking Login, and...

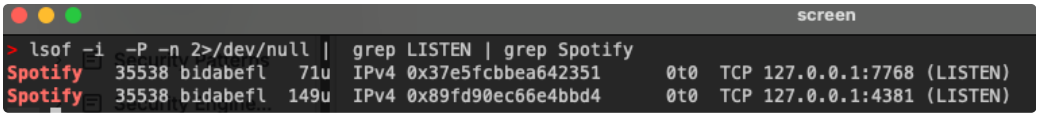
Pass it to your phone



I send the Oauth GET request to myself via MS Teams, and open it on my phone...



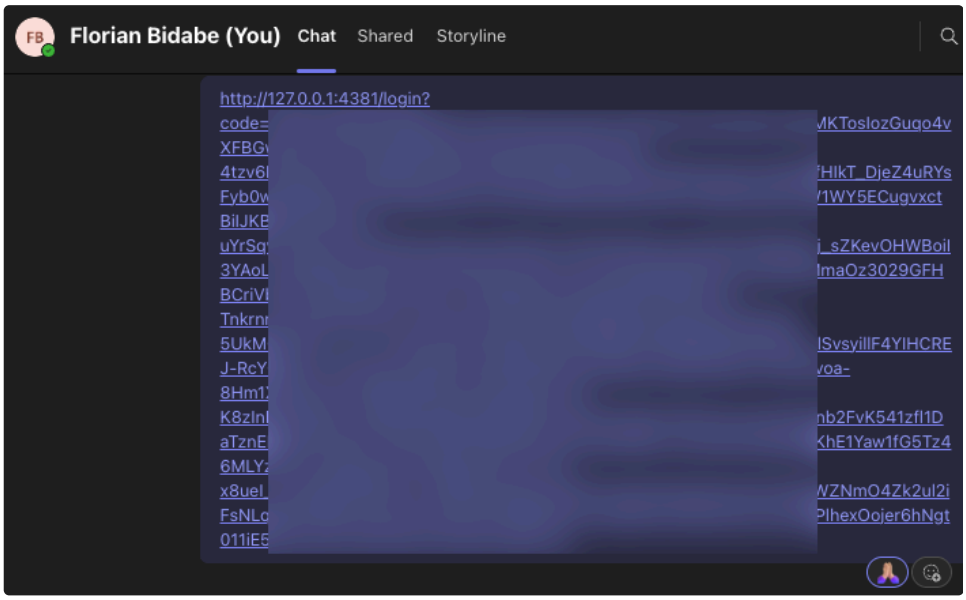
I am redirected to `http://127.0.0.1:4381`
This tells me that the server response is an other GET Request to talk to the App via localhost



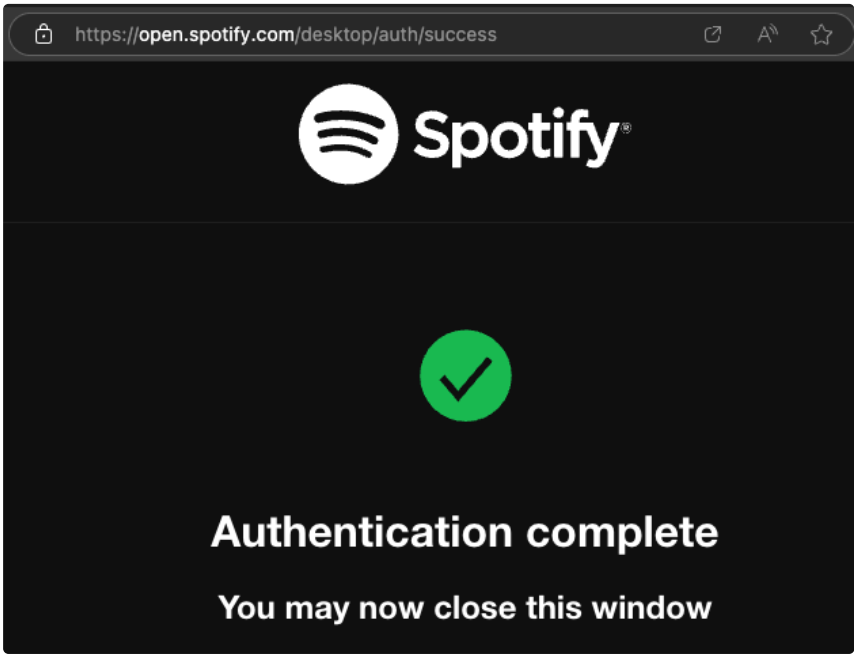
Expectedly my phone isn't listening on TCP4381, but my SOE is...

Pass the Oauth response back to your SOE

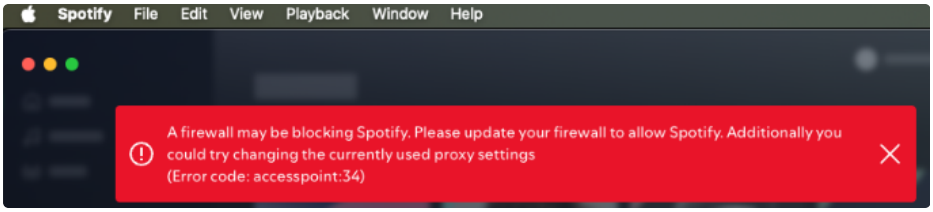
```
http://127.0.0.1:4381/login?code=AQDTjQbFxbDTNpzL... (authorisation code)
```



And copy / paste this cryptographic challenge to my SOE, and...



It's looking good, but...



Oh bummer... It fails again!
By now, we know the cryptographic challenge was valid and expected...
127.0.0.1 received it otherwise it wouldn't say `Authentication complete`
The subsequent query however must have failed...

Attempt Resolution #2

Authentication is often time-sensitive, and Spotify is no longer listening on TCP 4381 😢 ...
We'll need to start again from generating a new OAuth GET request, but we're not giving up yet...

This time, let's instruct Spotify to go from Charles (Web Debugging Proxy), to Alpaca (PAC aware Auth proxy), to Prisma. Fact is, this is local traffic, it does not need to go anywhere but we need more clarity on what's happening and we need to ensure 127.0.0.1 receive the auth code, and can communicate to/from Spotify App...

Use a local proxy

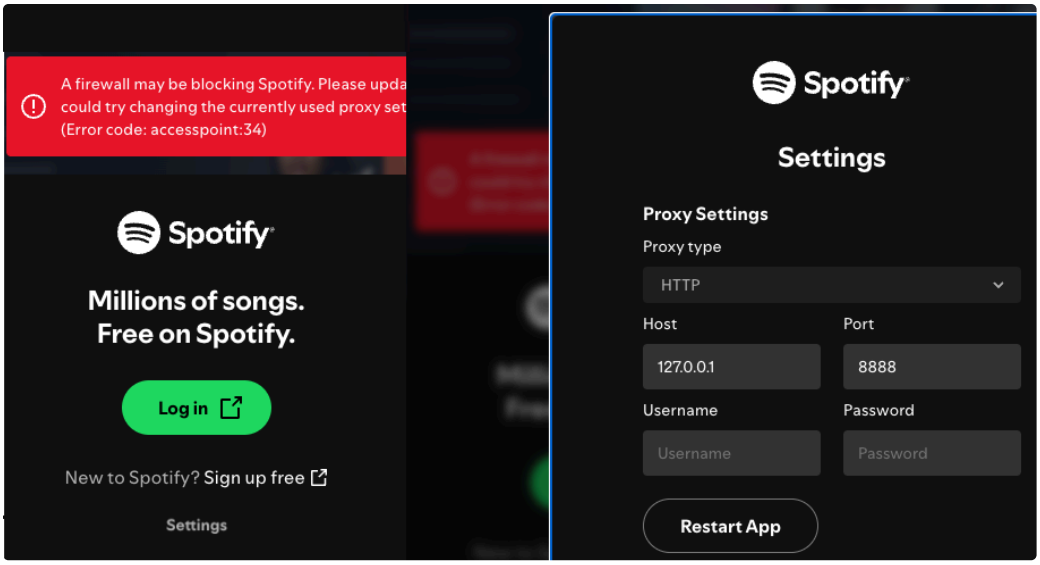
So let's instruct Spotify to go through Charles first

If you need to setup Charles on your SOE, expand this...

Charles can be installed on MacOS via SelfService
Web Debugging

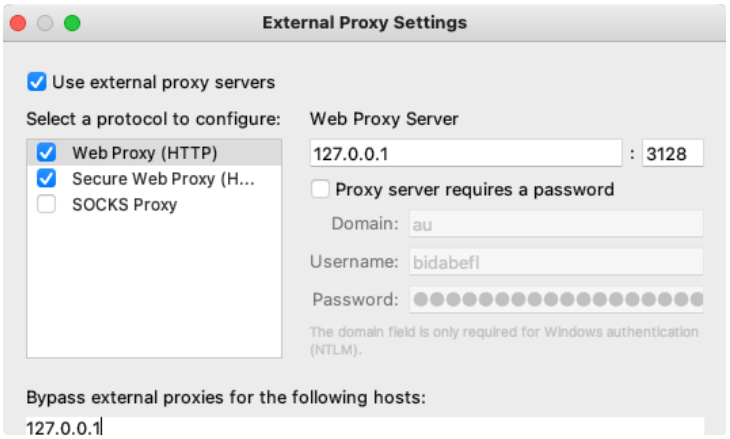
On Windows, it would need to be installed manually and require:

- Application Control exception
- Admin Privilege (To install the App, but also to Trust Charles CA)
- and the Group licence key from a MacOS machine: It's in the config file attached to this page
  Guide: How to install Charles Proxy



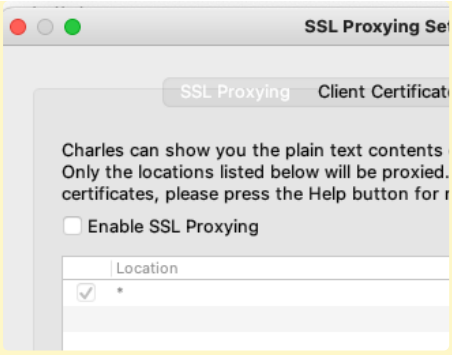
as opposed to `auto-detect` (which goes to prisma)...

We set Charles to forward the traffic to Alpaca and in turn to Prisma as instructed by the Group PAC



Proxy / External Proxy Settings
We also tell Charles, to exclude/bypass `127.0.0.1` since there is no point forwarding local traffic to an external proxy

⚠️ Also, by now, we strongly suspect that the auth process is (very) susceptible to SSL Forward Inspection... so let's disable SSL Forward Inspection on Charles.



Proxy / SSL Proxy Settings

Now we need a browser who talks to Charles... Edge and Chrome (Managed) by default will not. They'll talk to Prisma `http://cba.proxy.prismaaccess.com:8080` as a first hop bypassing Charles and Alpaca...

<< What if we use Firefox (Managed), or Brave (Unmanaged)? >> Yes, that's an option.
On Firefox, we can set proxy settings, we're not bound to the System proxy settings!
On Brave, we can as well, but only with `FoxyProxy` extension since, by default, Brave is a Chromium based browser which simply inherit proxy settings from the OS (Whether Windows or MacOS)

For more information on Browsers & Proxy fiddling, see below...

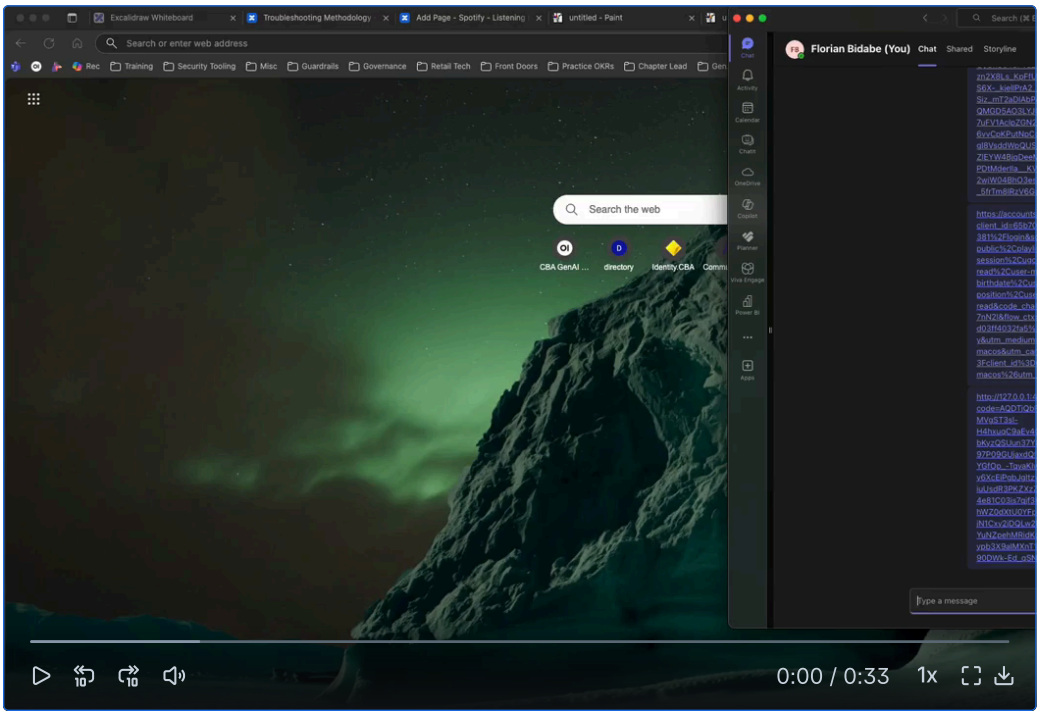
- Block pesky HTML elements on a browser
- Troubleshooting Methodology

Now, let's try again from copying the Oauth request and response via a mobile phone.
The response will be pasted into a Browser that passes the traffic to Charles, and to localhost.
If this all works, the following request should resemble the token exchange example below, but we cannot confirm it as we are no longer SSL Proxying with Charles...

```
POST https://accounts.spotify.com/api/token
Headers:
  Content-Type: application/x-www-form-urlencoded
  Authorization: Basic {base64(client_id:client_secret)}
Body:
  grant_type=authorization_code
  code=AQDTjQbFxbDTNpzL...      // The received code
  redirect_uri=http://127.0.0.1:4381/login
  code_verifier=xxxxx           // Original PKCE verifier
  client_id=65b708073f...

2. DPoP Proof Addition:
```

```
Header:
  DPoP: eyJ0eXAiOiJkcG9wK2p3dCIs...  // DPoP proof JWT
```

And... Success, at last!



Conclusion

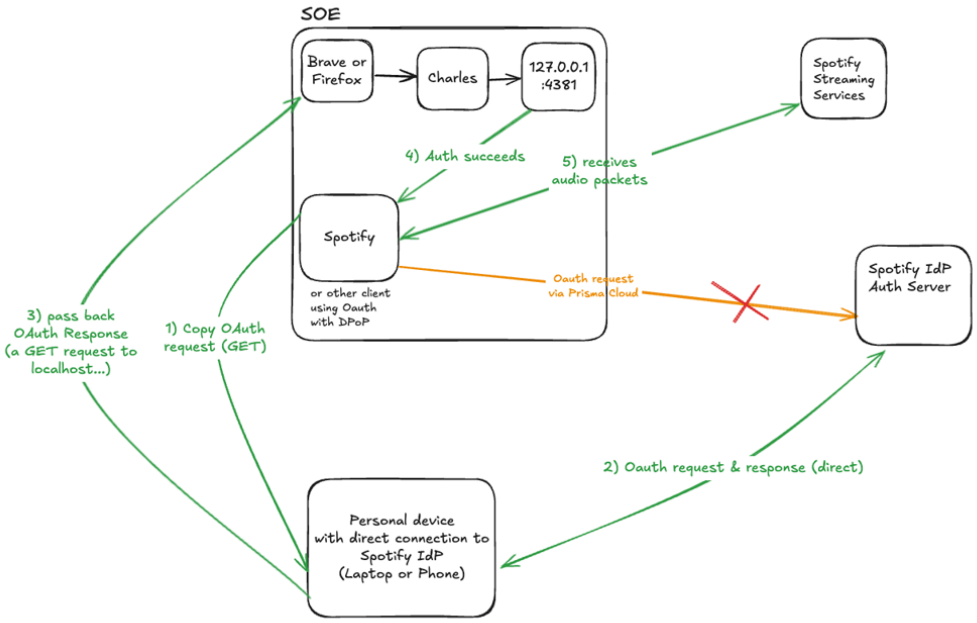
> ### Success Path
>
> ```
> [Client] → [Charles (NO SSL INSPECT)] →
> [Prisma] → [Spotify]
>                    |
>          Maintains original DPoP
>          and PKCE integrity
> ```

Failing Path

```
[Client] → [Prisma SSL Inspection] → [Spotify]
                ↓
         Breaks DPoP binding
```

Possible Explanation

```
Scenario (Succeed):
[Client] → [Charles (NO SSL INSPECT)] → [Prisma
(NO SSL INSPECT)] → [Spotify]
                      |
|
              Passes CONNECT                Passes
CONNECT
              transparently
transparently

Scenario (Failing):
[Client] → [Charles (WITH SSL INSPECT)] →
[Prisma (WITH SSL INSPECT)] → [Spotify]
                      |
              SSL inspection breaks
              DPoP validation
```



The authentication flows

By now, you should have a better understanding of the OAuth authentication process using DPoP. Also you can remove Spotify Proxy settings to Charles, and flick it back to Auto-detect (Prisma) since only the Auth was failing, not the actual music streaming...

💡 *Helpful? Drop me a thanks on Achievers ! And if you've got knowledge to share, don't hold back - we all grow when we learn from each other.*