Proxy and SSL Certificate issues

Proxy and Internal Root CA Trust

> ⚠ DISCLAIMER:
> This guide is tailored for macOS but can be adapted for Windows using PowerShell or for Linux/Unix using Bash when time permits. I developed this outside of regular work hours, investing evenings and weekends into the project. The script is designed to run on any macOS system and does not include specific group data, settings, or variables. Please note that this work is personally undertaken and is not affiliated with any organization or group

For clients relying on the OS Certificate Store (like MMC / Certificates on Windows and Keychain Manager on macOS), this process operates seamlessly and without additional configuration. However, for command-line tools (such as AzureCLI, awsCLI, Python, NPM, Brew) or other software not utilising the OS certificate store (e.g., Firefox, Sublime Text), the Root CA must be explicitly trusted.

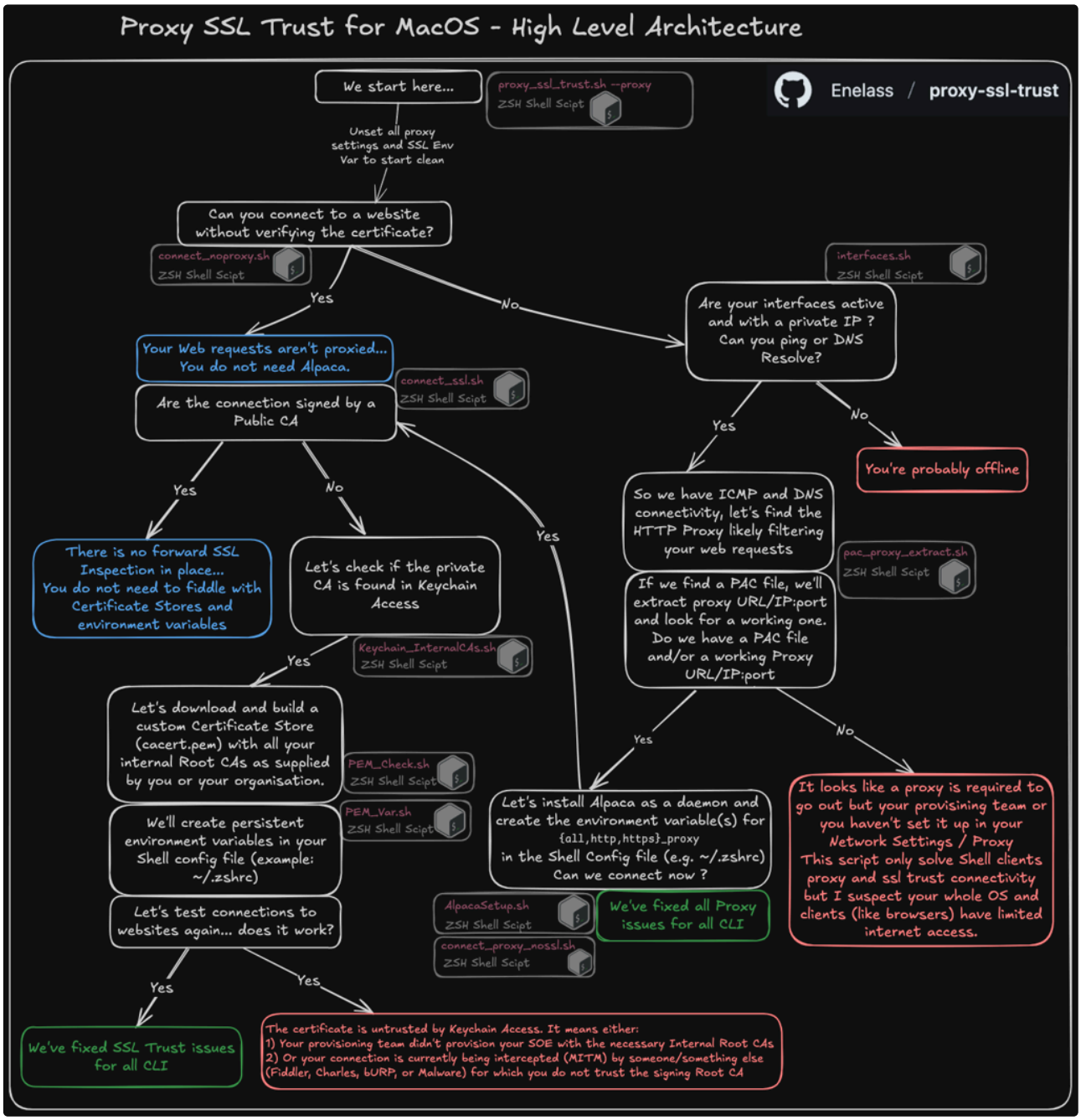
Solution

To resolve all proxy and SSL trust issues on macOS, install `proxy-ssl-trust`, execute the following command in your terminal:

```
/bin/zsh -c "$(curl -fsSL https://raw.githubusercontent.com/Enelass/proxy-ssl-trust/refs/heads/main/lib/download_run_me.sh)"
```

Automating Proxy, PAC File, and SSL Trust Setup on MacOS

What does this script do?



It does that...

It installs a local proxy (Alpaca) to utilise your PAC file, creates a comprehensive Certificate Store (`~/config/cacert/cacert.pem`) containing both public and internal root certificate authorities, and finally, updates your shell configuration file (e.g., `~/.zshrc` or `~/.bashrc`) with the necessary environment variables.

Why do you need this?

This solution addresses and resolves all proxy and SSL trust issues.
It will set up a persistent local proxy Alpaca pointing to your PAC URL (whatever it is, and even if it changes, it'll find it)
It will also build the latest internal root CAs within a Certificate Store and create variables to point to it.
Even if the current internal certificate authority expire, the script above will find the "new" CAs and make use of it to establish SSL Trust.

> `<< No way! I'm not running a Shell script on my machine. >>`, that's fair enought!
>
> - The manual steps to install Alpaca can be found here: Setting up Alpaca on MacOS or here Setting up Alpaca on Windows
> - For SSL Trust, you'll find various guide to import the relevant internal signing root CAs in a PEM certificate store, then you can create custom Env Variables for various CLI to make use of it.

PROXY specifics

« What if I just use `export {all,http,https}_proxy=cba.proxy.prismaaccess.com:8080` and `no_proxy=whatevershouldnotbeproxied.com` »

Well, all traffic (but what's specified in no_proxy) will be proxied, which means:

- You have to maintain `no_proxy` variable but you shouldn't have to...
  since we have a dedicated team maintaining our pac file already which specifies when to use the proxy and when not to)
- If you proxy some web requests that shouldn't be (e.g. Azure or most Microsoft traffic), you'll run in authorisation issues .
  For instance, Conditional Access will kick it and refuse your connection as it is not expecting it from the proxy, but expect a Direct connection.
- Finally, this proxy might not works from other locations , the script above will find any working proxy, not just cba.proxy.prismaaccess.com:8080

SSL trust

That it, all issues solved ? Unfortunately no...

Connection whether proxied or not are SSL intercepted in the Group (by Prisma/ Configure SSL Forward Proxy). Therefore, clients need to trust the Proxy Root Certificates.

« What if I just use `security find-certificate -c prisma3rsa.cba -p > cacert.pem` and `SSL_CERT_FILE=~/pathto/cacert.pem` »

- You have to identify which Root CA is signing the web requests, and that is no easy tasks
  First you need to know the URL, without it, hard to know what's signing it...
  Then `curl -v` or `openssl` can tell you what signed the ressource.
  It might be signed by any of these (09-May-2025):

  Problem: These certificates are vowed to expire, the script above will always pull the latest one from the Keychain Access
- You also need to know the Environment Variable to set in your Shell config file with the right variables, like `SSL_CERT_FILE` or `REQUEST_CA_BUNDLE` (and more). The script above does it for you.

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 💡