Group Internal Root CAs Certificate Store

SSL Certificate Environment Variables for Corporate Networks

What Are SSL Certificate Environment Variables and Why Do You Need Them?

<< I don’t really want a lecture... I just want to solve my SSL issues >>

In that case, for MacOS, go to: Group Internal Root CAs Certificate Store | Method 1 Au

tomated

For Windows, I haven't scripted it so you'll need to read the below to understand the issue (and solve it)...

<< Do I have to built a cacert.pem myself ? >>
Well, not really... here's one byt it'll eventually become outdated! `cacert.pem`
Up-to-date as of: `3 Sept 2025`

If you're working on a corporate network, you've probably encountered SSL certificate errors when trying to use development tools, package managers, or cloud services. These errors typically look like:
This happens because your company uses custom Certificate Authorities (CAs) or proxy servers that intercept SSL traffic.
Most development tools don't automatically trust these corporate certificates, so you need to configure them manually using environment variables.

- "SSL certificate verify failed"
- "CERTIFICATE_VERIFY_FAILED"
- "unable to get local issuer certificate"

What You'll Need Before Starting

1. **Your company's certificate bundle file** - Usually a `.pem` file containing Base64 Root Signing CAs
2. **Administrative access** to modify **system** environment variables on your machine
   You can however change the **user** environment variable without needing administrative access.
3. **Knowledge of which development tools you use** - This determines which variables you need to set

The Complete List of SSL Certificate Environment Variables

Core Variables (Most Important)

These work with the most common development tools:

- `SSL_CERT_FILE` - Used by OpenSSL, Python urllib, and many other libraries
- `REQUESTS_CA_BUNDLE` - Python requests library
- `GIT_SSL_CAINFO` - Git version control system
- `NODE_EXTRA_CA_CERTS` - Node.js & npm package manager
- `AWS_CA_BUNDLE - AWS CLI and SDKs`
- `CURL_CA_BUNDLE` - cURL command-line tool

  I don't think this is needed on the SOE because cURL inherits what `Keychain Access` *or* `Windows Certificates Store` *trust and our provisioning team did a great work at trusted internal Signing Root CAs. However it doesn't hurt to set it anyway...*

<details>
<summary>The bigger list...</summary>

Language-Specific Variables

Python Development

- `PIP_CERT` - pip package manager

Node.js Development

- `NODE_EXTRA_CA_CERTS` - Node.js runtime
- `NPM_CONFIG_CAFILE` - npm package manager
- `YARN_CAFILE` - Yarn package manager

Java Development

- `JAVA_TOOL_OPTIONS` - All Java applications (set to `-Djavax.net.ssl.trustStore=/path/to/truststore`)
- `MAVEN_OPTS` - Maven build tool

Ruby Development

- `BUNDLE_SSL_CA_CERT` - Ruby Bundler

Javascript Development

- `DENO_CERT` - Deno JavaScript runtime
</details>

Version Control

- `GIT_SSL_CAINFO` - Git version control system

Cloud Services & APIs

- `AWS_CA_BUNDLE` - AWS CLI and SDKs
- `CLOUDSDK_AUTH_CORE_CUSTOM_CA_CERTS_FILE` - Google Cloud SDK

Container & DevOps Tools

- `DOCKER_CERT_PATH` - Docker Desktop
- `DOCKER_TLS_VERIFY` - Docker TLS verification
- `HELM_CA_FILE` - Helm Kubernetes package manager

HashiCorp Tools

- `CONSUL_CACERT` - Consul
- `VAULT_CACERT` - Vault

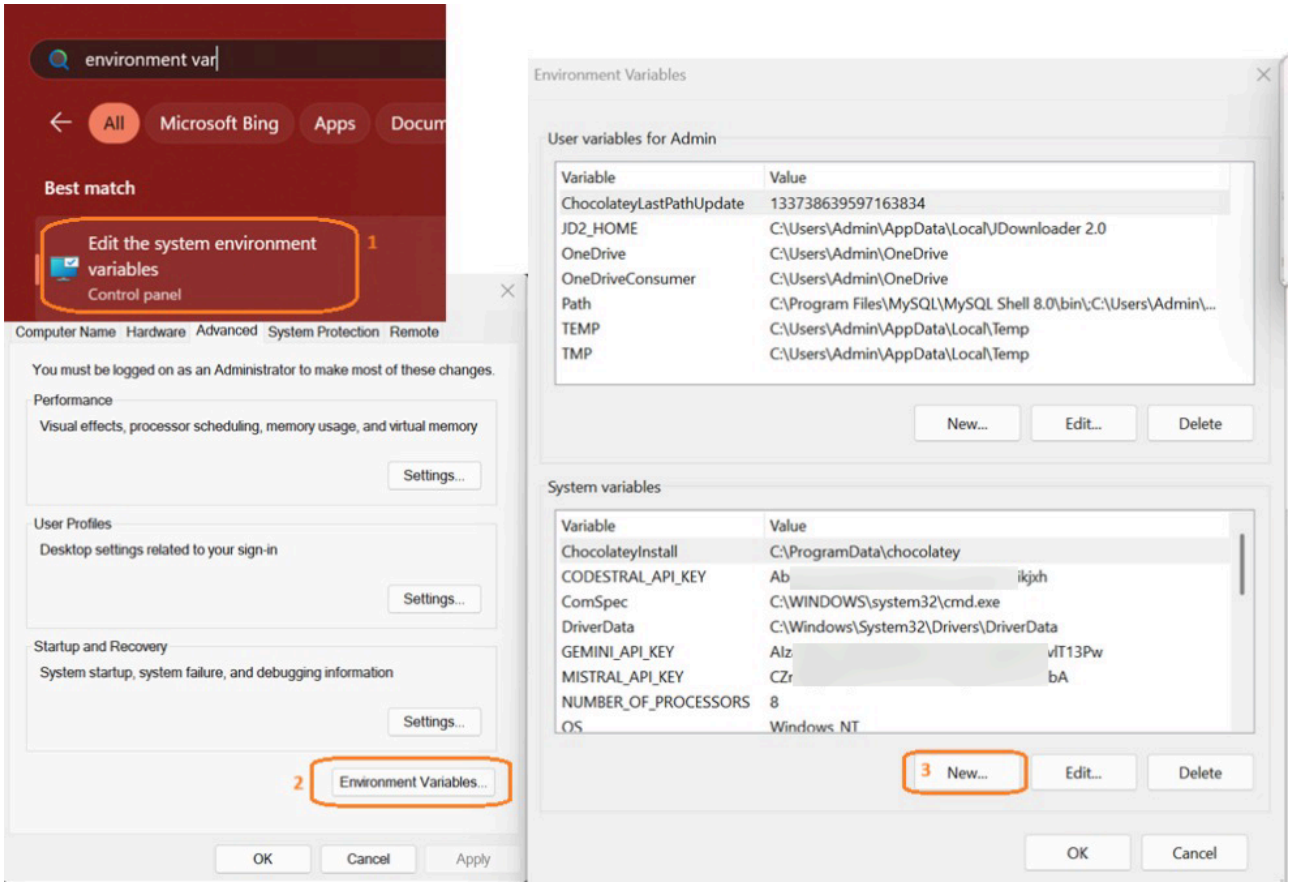
Specialized Tools

- `GRPC_DEFAULT_SSL_ROOTS_FILE_PATH` - gRPC libraries
- `TAPLO_EXTRA_CA_CERTS` - Taplo TOML toolkit

How to Set Up on Windows

Method 1: Using System Properties (Recommended)

1. **Press** `Win + R`, type `sysdm.cpl`, and press Enter
2. **Click the "Advanced" tab**
3. **Click "Environment Variables"**
4. **Under "User variables" (for your account only) or "System variables" (for all users), click "New"**
5. **Enter the variable name** (e.g., `SSL_CERT_FILE`)
6. **Enter the full path** to your certificate bundle (e.g., `C:\\certs\\company-bundle.pem`)
7. **Click OK** to save
8. **Repeat for each variable** you need



Method 2: Using Command Prompt (PowerShell)

```powershell
# Set for current user
[Environment]::SetEnvironmentVariable("SSL_CERT_FILE", "C:\certs\company-bundle.pem", "User")
[Environment]::SetEnvironmentVariable("REQUESTS_CA_BUNDLE", "C:\certs\company-bundle.pem", "User")
[Environment]::SetEnvironmentVariable("CURL_CA_BUNDLE", "C:\certs\company-bundle.pem", "User")

# For system-wide (requires admin privileges)
[Environment]::SetEnvironmentVariable("SSL_CERT_FILE", "C:\certs\company-bundle.pem", "Machine")
```

Method 3: Create a Batch Script

Create a file called `setup-certs.bat`:

```batch
@echo off
set CERT_PATH=C:\certs\company-bundle.pem

setx SSL_CERT_FILE "%CERT_PATH%"
setx REQUESTS_CA_BUNDLE "%CERT_PATH%"
setx CURL_CA_BUNDLE "%CERT_PATH%"
setx GIT_SSL_CAINFO "%CERT_PATH%"
setx NODE_EXTRA_CA_CERTS "%CERT_PATH%"
setx NPM_CONFIG_CAFILE "%CERT_PATH%"
setx PIP_CERT "%CERT_PATH%"
setx AWS_CA_BUNDLE "%CERT_PATH%"
```

```
12
13 echo Certificate environment variables have been set!
14 echo Please restart your command prompt or IDE for changes to take effect.
15 pause
```

Run this batch file as Administrator.

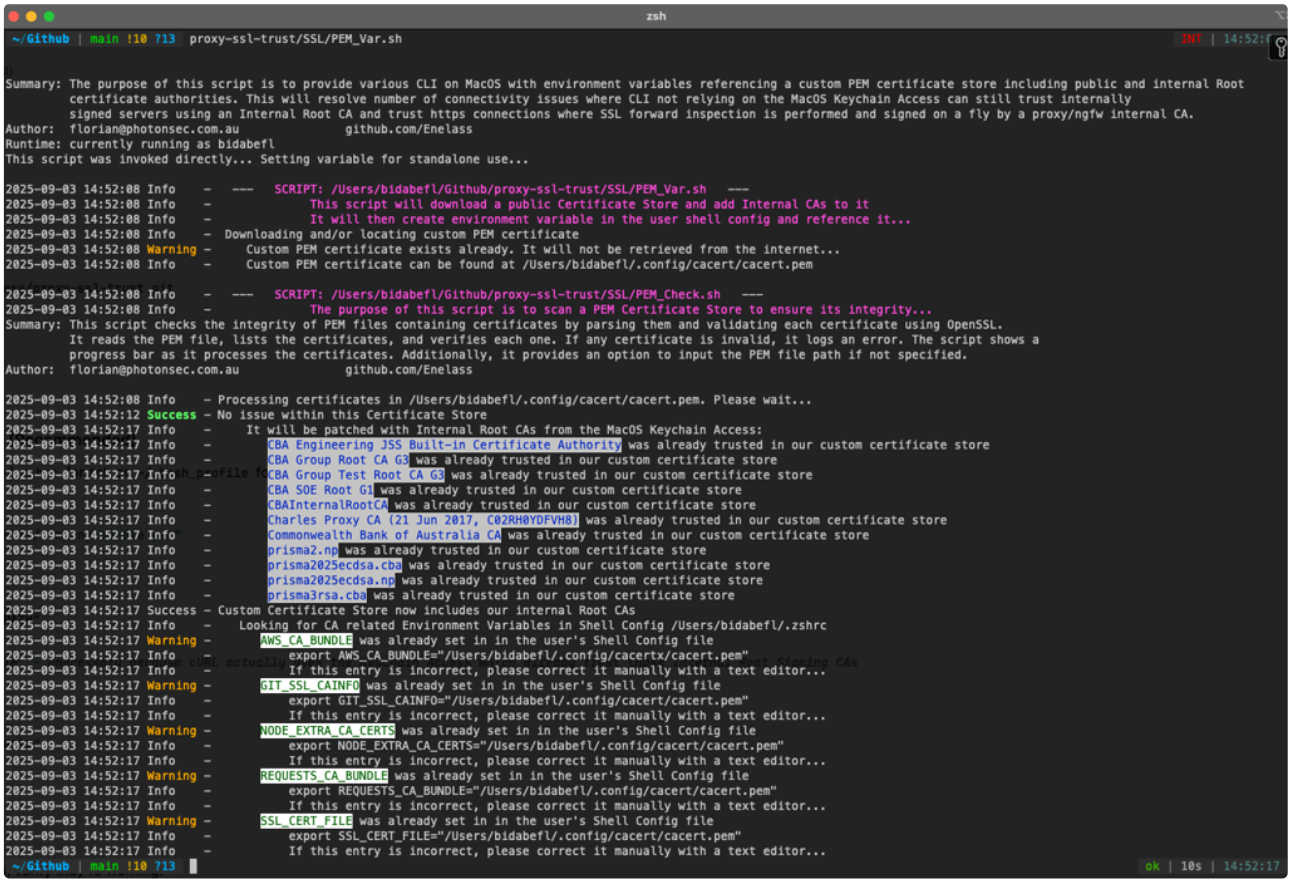
How to Set Up on macOS

Method 1 - Automated

```
1 git clone https://github.com/Enelass/proxy-ssl-trust.git
2 proxy-ssl-trust/SSL/PEM_Var.sh
```

What does it do?

1. It downloads a Public CA Store from the Mozilla foundation containing all Public Root CAs
2. It extract each Trusted Signing Root CAs from Keychain and add it to the store above. (Expired, untrusted, leaf and Intermediate CAs are excluded)
   a. Group Internal Root CAs are added so that internal *.cba are trusted
   b. SSL Forward Inspection CAs (Prisma) are trusted
   c. and others...
3. It checks the certificate store against corrupted entries...
4. It creates the environment variables in your default Shell config (whether that is `bash`, `zsh` or `fish`...)



Output

Method 2 - Semi-Manual - Security Engineering Handbook

> ⚠️ The command below will work, but it will trust all certificates in the Keychain, including Leaf and Intermediate CAs, which is unnecessary. Worse, it will also add untrusted and expired certificates.
>
> Also you will still have to declare the environement variables via `export` in your Shell config file (`~/.zshrc` , `~/.bashrc` or `~/.bash_profile`)
>
> ```
> security find-certificate -a -p
> /Library/Keychains/System.keychain
> /System/Library/Keychains/SystemRootCertificates.keychai
> n ~/Library/Keychains/login.keychain-db >
> ~/ca_bundle.pem
> ```

source: https://engineering-handbook.pages.commbank.io/knowledge-base/mac-guides/CBA-root-certificate#instructions-terminal

Method 3 - Manual - Using Terminal (Recommended)

Add these lines to your shell profile file (`~/.zshrc` for zsh or `~/.bash_profile` for bash):

```
# SSL Certificate Configuration
export CERT_PATH="/Users/Shared/certs/company-bundle.pem"

# Core variables
export SSL_CERT_FILE="$CERT_PATH"
export REQUESTS_CA_BUNDLE="$CERT_PATH"
# export ... etc
# export CURL_CA_BUNDLE="$CERT_PATH" - Unecessary because cURL actually uses the Keychain Access which already trust those Internal Root Signing CAs
```

After editing the file, run:

```
source ~/.zshrc  # or source ~/.bashrc
```

Verification and Testing

After setting up the environment variables, verify they're working:

Test Common Tools

```
# Test cURL
curl -vI https://google.com

# Test Python requests
python -c "import requests; print(requests.get('https://google.com').status_code)"

# Test npm
npm config get cafile

# Test Git
git config --get http.sslCAInfo
```

Check Environment Variables

Windows (Command Prompt):

```
echo %SSL_CERT_FILE%
set | findstr CERT
```

Windows (PowerShell):

```
$env:SSL_CERT_FILE
Get-ChildItem Env: | Where-Object Name -like "*CERT*"
```

macOS/Linux:

```
echo $SSL_CERT_FILE
env | grep CERT
```

Troubleshooting Common Issues

Problem: Variables not taking effect

Solution: Restart your terminal, command prompt, or IDE completely. Environment variables are only loaded when a new process starts.

Problem: Connection Errors (not related to SSL)

- See CLI - Why you should not set CNTLM nor Prisma

Problem: Certificate file not found

Solution:

- Verify the certificate file exists at the specified path
- Check file permissions (must be readable)
- Use forward slashes (/) in paths, even on Windows when possible

Problem: Still Getting SSL Errors

Solutions:

1. **Verify certificate format** - Should be PEM format with proper headers
2. **Check if you need additional certificates** - Some tools may require the full certificate chain
3. **Tool-specific configuration** - Some tools have their own certificate settings that override environment variables
4. **Possibly Strict-Validation Issue**, like with Python 3.13: 💔SSL Trust - Missing "Authority Key Identifier"

Problem: Java Applications Still Fail

Solution: Java uses a different certificate store format. You may need to:

1. Import certificates into Java's truststore using `keytool`
2. Set `JAVA_TOOL_OPTIONS` to point to a custom truststore file

Best Practices

1. **Store certificates in a standard location:**
   - Windows: `C:\\certs\\` or `%PROGRAMDATA%\\certs\\`
   - macOS: `/Users/Shared/certs/` or `~/.config/cacert/`
2. **Use a single certificate bundle file** when possible to avoid maintaining multiple copies
3. **Keep certificates updated** - Check what certificates are trusted in the OS Certificate Store (this is maintained by our provisioning teams SCCM & JAMF)

Security Considerations

- **File permissions:** Ensure certificate files are readable but not writable by unauthorised users
- **Path security:** Store certificates in secure locations that regular users cannot modify
- **Regular updates:** Keep certificates current to maintain security
- **Note on sensitivity:** These certificate bundles contain only public CA certificates for establishing trust; they do not include private keys and are not used for decryption.)

Conclusion

Setting up SSL certificate environment variables properly will save you countless hours of debugging SSL errors in corporate environments.
While it may seem overwhelming at first, most developers only need to set up the variables for the tools they actually use.

Start with the core variables (`SSL_CERT_FILE`, `REQUESTS_CA_BUNDLE`, `CURL_CA_BUNDLE`) and add language-specific ones as needed.
Remember to restart your development environment after making changes,
With this setup complete, your development tools should work seamlessly within our corporate network environment.