Web Debugging

This guide is about Web Debugging for CLI (cURL, brew, az, aws, pip, npm, etc...) or any other software which supports specifying Proxy details (Firefox for instance).
Note, these steps can easily be followed on Windows or Linux/Unix.
Instead of Charles, Fiddler or Burp Suite can be used...
Instead of Alpaca, CNTLM could be used...

If you wish to understand how you could capture all traffic sent and received on the workstation, please head this way → Local MITM (capture all traffic) from the SOE UNDEFINED
Example: you want to capture the Traffic from Chromium (Edge, Chrome, Brave) but it uses the System Proxy...

Web Debugging proxies are tools used to intercept, inspect, and tamper HTTP and HTTPS traffic between a client (such as a web browser) and a server.
They are commonly used for various purposes, including:

1. **Debugging and Troubleshooting**:
   - **Inspecting Requests and Responses**: Developers can examine the details of HTTP requests and responses, including headers, payloads, and status codes.

- **Identifying Issues**: Helps in identifying and diagnosing issues in web applications, such as incorrect headers, malformed requests, or server errors.

2. **Security Testing**:
   - **Intercepting Traffic**: Intercept and analyse traffic to identify vulnerabilities such as SQL injection, cross-site scripting (XSS), and other security flaws.
   - **Modifying Requests**: Enables the modification of requests to test how the server responds to different inputs, which can help in finding security weaknesses.
3. **Performance Analysis**:
   - **Monitoring Performance**: Helps in analysing the performance of web applications by monitoring response times, identifying bottlenecks, and optimizing load times

     I'm writing this article because the current knowledge base only addresses capturing traffic from a mobile device using MacOS SOE.
     My particular scenario involves capturing the traffic from a corporate and managed workstation (SOE).

Requirements:

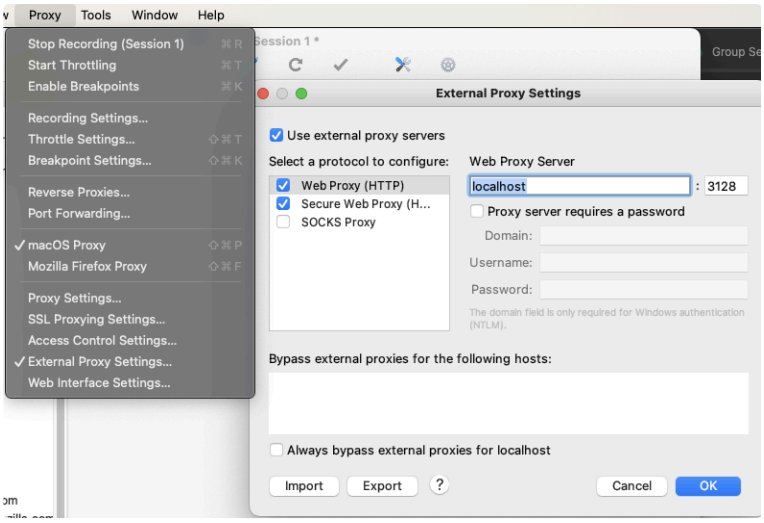
- MacOS Personal machine or SOE
- Local Admin
- Homebrew, Charles, and Firefox, and/or other clients

Environment Setup

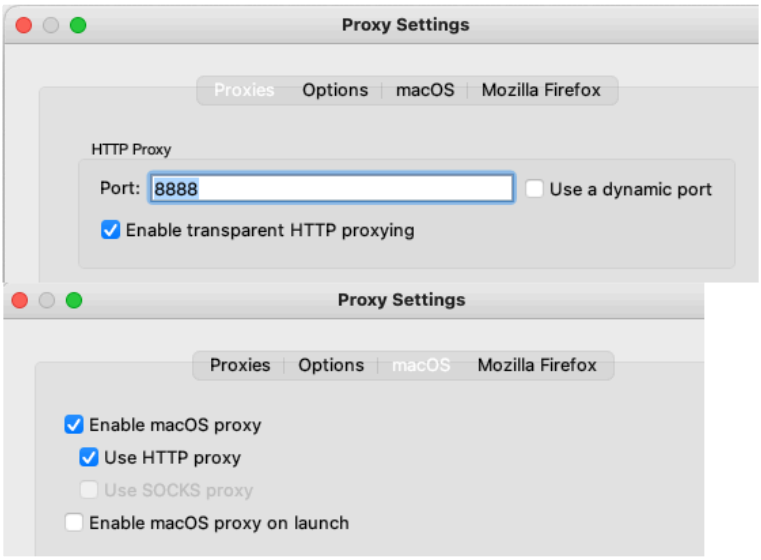
Charles Installation and Setup

Setting up the proxy

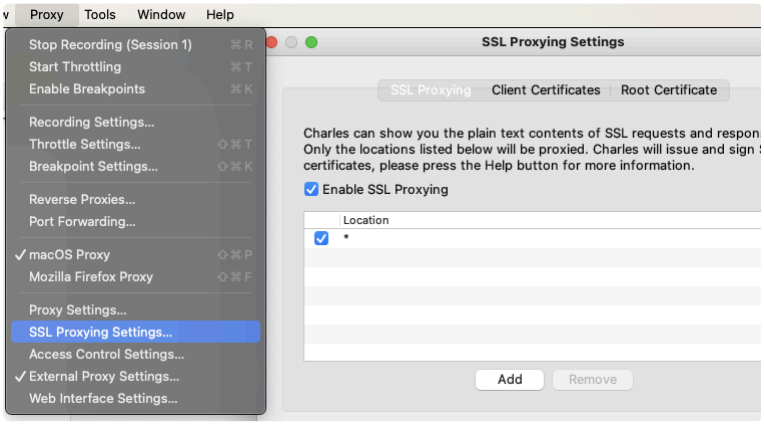
Check the MacOS Proxy is ticked and set up external proxy to localhost:3128 (Alpaca)
We also need to ensure it is listening on TCP888* and uses MacOS Proxy



Make sure Charles forwards the traffic to Alpaca...



Verify that your proxy is set to listen on TCP8888

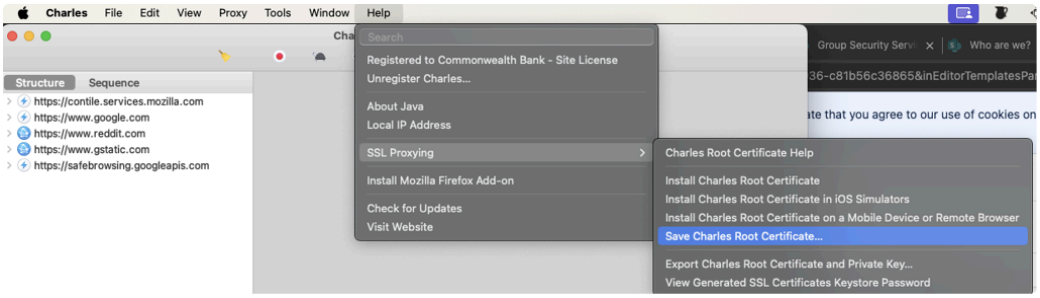


Make sure we are capture SSL traffic, so HTTPS and not just HTTP

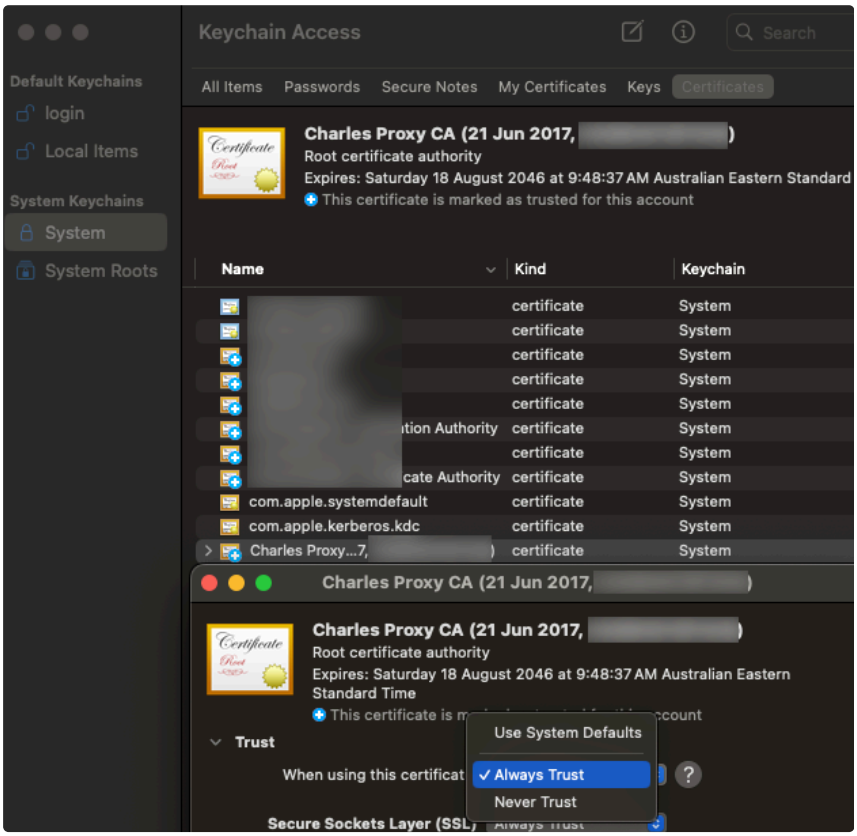
Trusting the Charles Root CA

Now Charles is listening but it uses its own Root Certificate Authority to signs certificates on-the-fly so we can inspect the traffic.

We need to trust that Root CA in our System certificate store (Keychain Access on Mac, MMC/Certificates in Windows) so the connections can be trusted, otherwise, we;ll see lots of Untrusted error messages as HTTP responses to our (intercepted) requests.



Charles: Export Charles Root CA



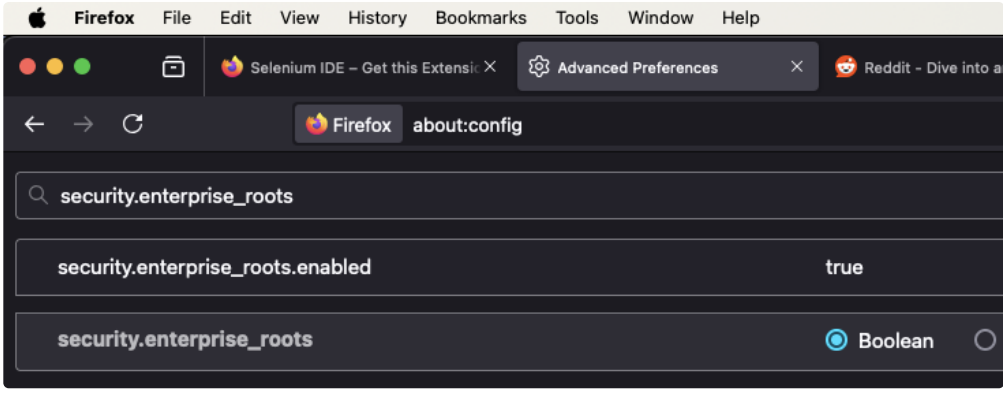
MacOS Keychain: Import it in Keychain Access and Trust it

Firefox setup

SSL Trust

By default Firefiox uses its own certificate store to assess whether to trust or not a Root CA. This is a cacert.pem (plain text file) containing Base64 certificates for each public Root CA. To trust Charles Root CA, we either need to append the Base64 cert to Firefox PEM Certificate

Store,
or we need to instruct Firefox, to use the System Certificate store instead of it's own... see screenshots below:



Firefox: about:config, make sure Firefox uses the MacOS Keychain as opposed to its own Base64 cacert.pem (if false)

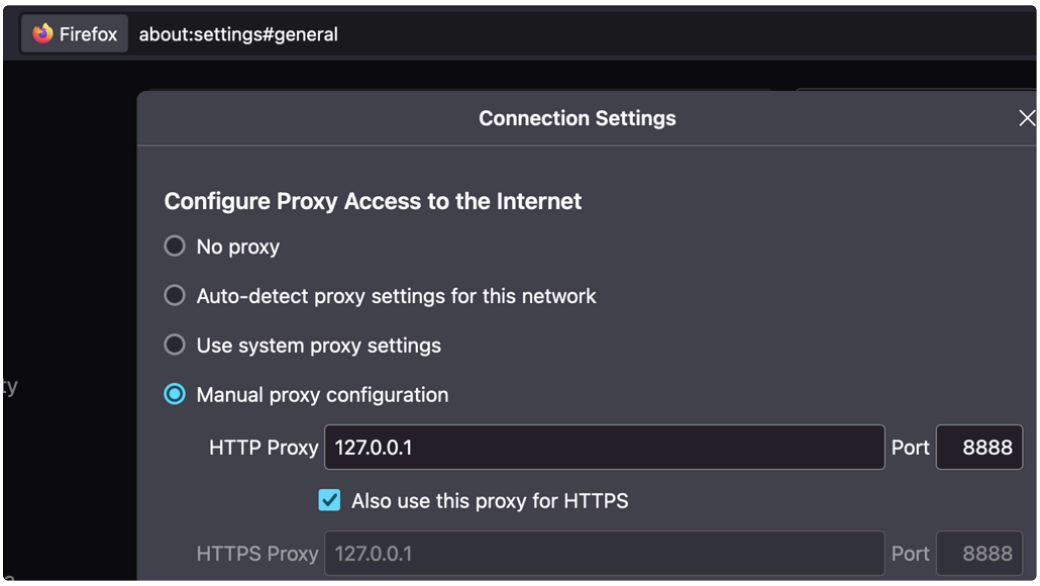
Proxy Settings

Unlike other browsers, we can explicity set the proxy of our choice in Firefox `about:settings`

This is very useful since other Browser might use the system proxy which might be locked by a MDM profile.

If you really need to see the traffic from such software, see this → Charles: Debug it all...

UNDEFINED



Firefox, about:settings, network settings or search for Proxy and point Firefox to Charles

What's happening now?

We're almost there

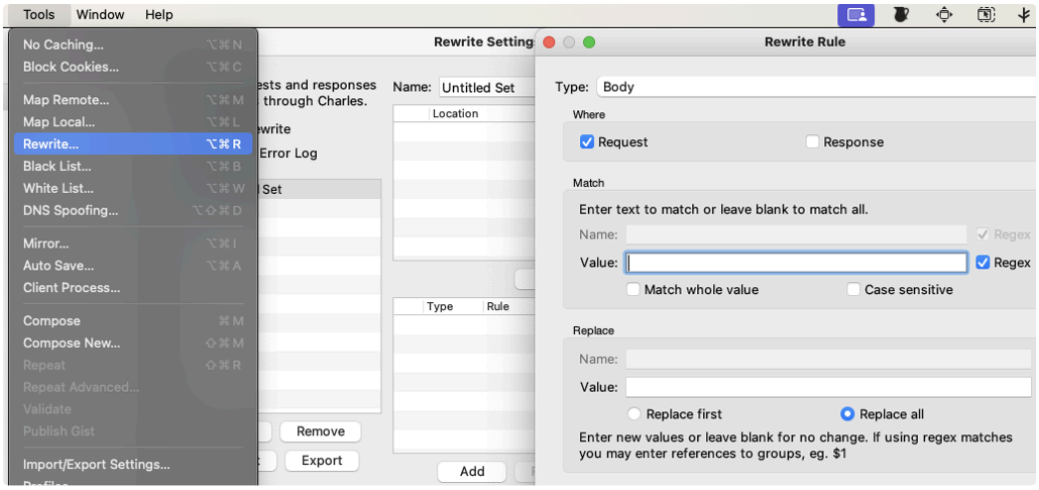
Is the proxy listening?

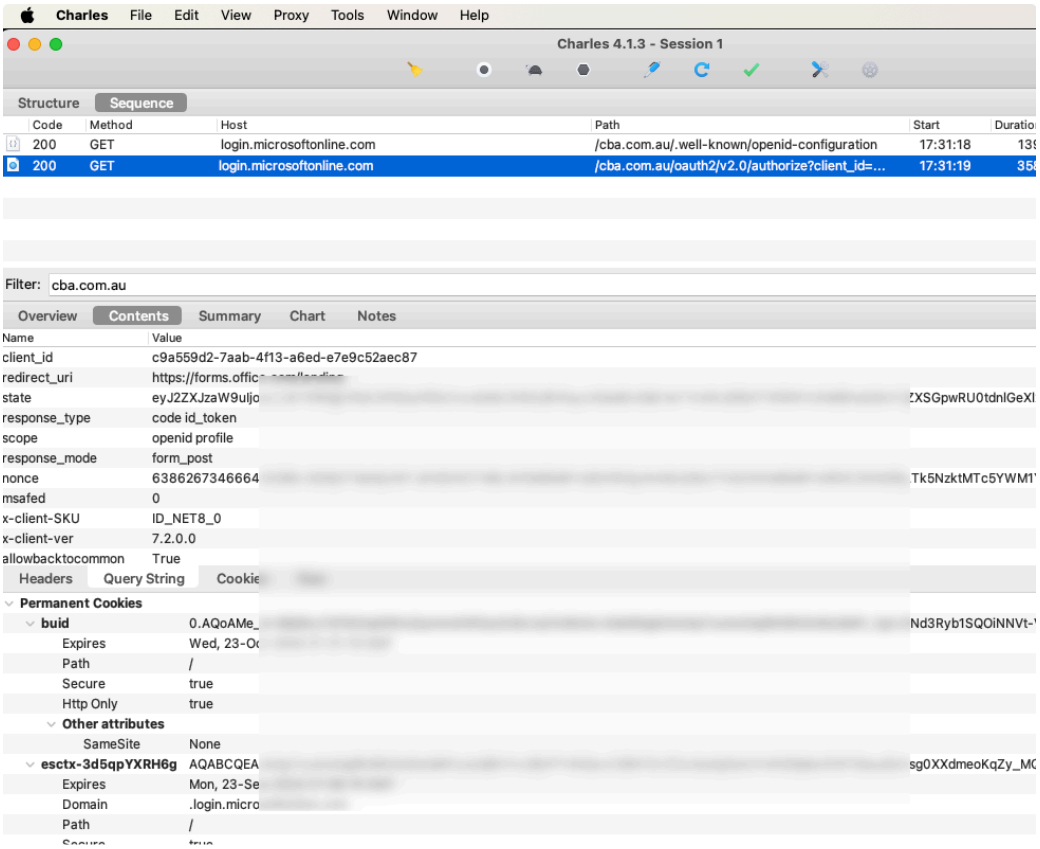
Start Charles and make sure it's listening using lsof: `sudo lsof -i :8888 -P -n | grep LISTEN`

Charles should listen on TCP 8888 (by default), Alpaca should listen on TCP 3128

If these ports are not Listening, then the processes are either not running or are failing to start...

Well, that's it, you can now inspect the traffic, record it, intercept it and tamper the request or response it using Regex, have fun!





Alpaca, Charles, Prisma... Why so many proxies?

Personal use

Well, on a personal machine, you'd be either be using no proxy, or only one of your choice

Corporate use

Now in a corporate environement, you'd be using one or more proxies

Web-debugging

You could use other debugging proxies as a substitute for `Charles`:

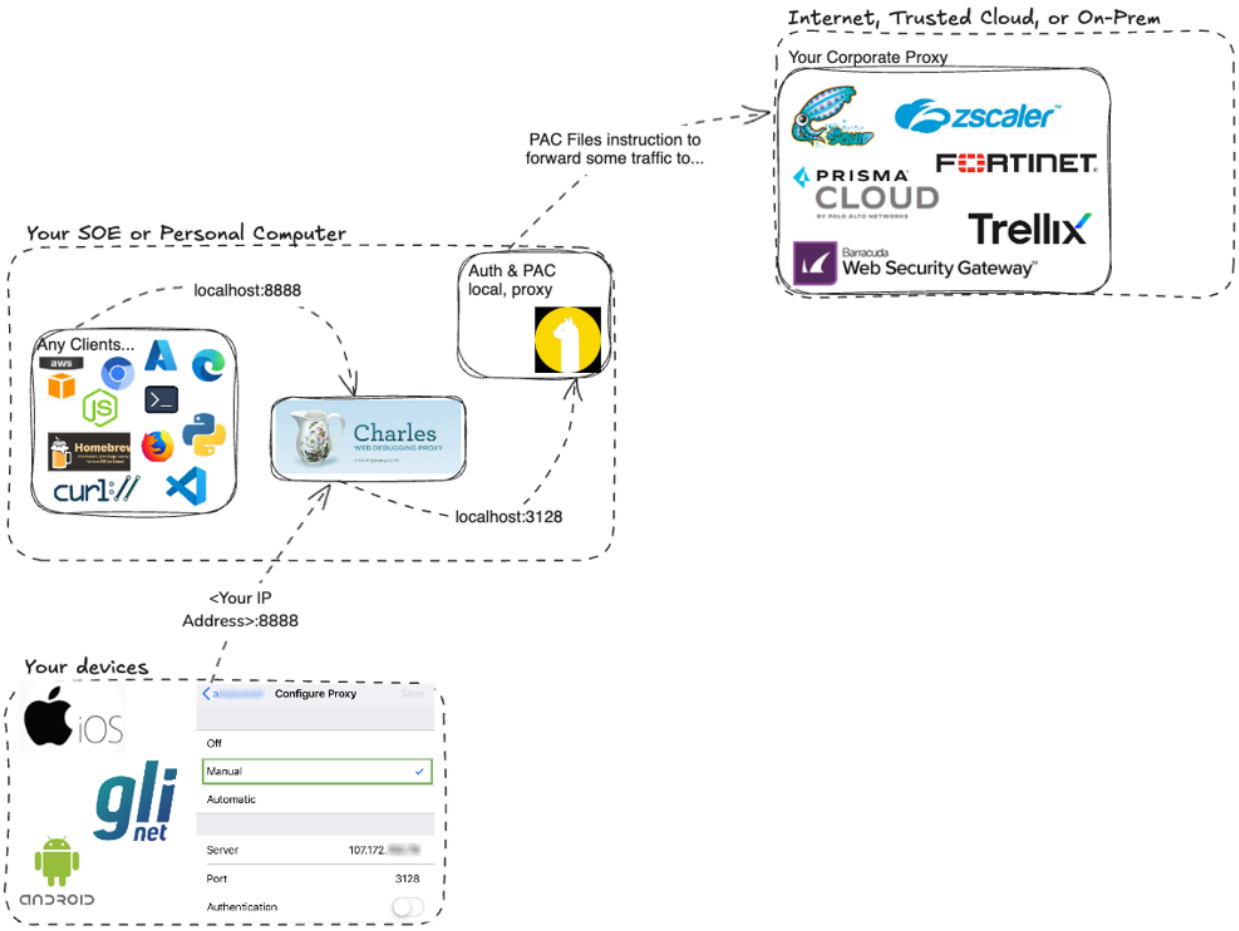
- `proxyman` Proxyman · Debug, intercept & mock HTTP with Proxyman
- `Fiddler` Web Debugging Proxy and Troubleshooting Tools | Fiddler
- technically you could even use `burp suite` but it isn't quite designed for debugging... Burp Suite Professional - PortSwigger Download Burp Suite Community Edition - PortSwigger

NTLM Proxies

- Alpaca GitHub - samuong/alpaca: A local HTTP proxy for command-line tools. Supports PAC scripts and NTLM authentication. is an HTTP proxy for command-line tools. It **supports PAC files** and NTLM authentication
- `CNTLM` Cntlm Authentication Proxy it supports NTLM Authentication **but not PAC files**

SSL forward decryption proxy

- Finally, in a work environment, the traffic may traverse a corporate proxy (packed with security goodies: like URL Whitelisting, Malware inspection, Application Fingerprinting, SSL interception, etc...)



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 💡