
YOU PROXIED
Intercepting Arbitrary TCP with alsanna

Architecture

Handlers

`handlers/*`

- `setup_client_facing()`
  Convert underlying socket to interact with client, probably `return ProtoHandler(sock)`
- `setup_server_facing()`
  Convert underlying socket to interact with server, probably `return ProtoHandler(sock)`
- `obj_to_printable()`
  Convert a message object to editable text
- `printable_to_obj()`
  Convert edited text back into a message object

Sockets

`handlers/*`

- `send()`
  Consume a message object and ensure it all gets sent using the underlying socket.
- `recv()`
  Read from underlying socket and produce a message object.
- `connect()`
  Just needs to connect the underlying socket to the right target, probably `self.sock.connect(*args)`
- `close()`
  Just needs to close the underlying socket, probably `self.sock.close()`

Connection Processes

`cnxn_proc.py`

- `manage_connections()`
  Set up `forward()` thread for client->server and server->client directions, register each with UI; setup client-facing socket handlers only
- `forward()`
  Shovel message objects in one direction; setup server-facing socket handlers when first message to server ready

  Final handler's message object converted to text, sent to UI for editing, result encoded back to message object

  For client->server: listener is client-facing, sender is server-facing; vice versa for server->client

Conventions

- Handlers can call other handlers, e.g. your protocol's `STARTTLS` can be outsourced to the `tls` handler:
  - Make calls to another handler's `setup_listener()` and `setup_sender()` in your own handler's if you know at setup time, or in your socket class if you need to mid-stream
  - Keep `obj_to_printable()` and `printable_to_obj()` stateless as much as possible
- The first handler always gets a socket that operates on `bytes` objects, but beyond that you need to know what the handler beneath is going to use
- The `ldap` handler prints messages as JSON sandwiched between metadata. `edit_utils.py` contains reusable code for this format
- Some protocols can't be implemented as handlers alone – for instance, if the server sends the first message, you must edit `cnxn_proc.py` which assumes the client sends the first post-handshake data