Fiddler MCP Server – Overview

This guide shows how the Fiddler MCP server - bridge works. This is High-level data flow and key elements of the solution. Designed to be used with Gemini 2.5 Pro or any other Gemini model. However, in my testing it worked with GPT4.1 mini and nano just fine.

Happy to answer any questions!

TLDR – Traffic to Insight

```
┌──────────────┐    HTTP POST    ┌────────────────────┐     REST     ┌────────────────────┐     MCP     ┌──────────────────────────┐
│ Fiddler      │  ────────────▶  │ enhanced-bridge.py │  ────────▶   │ 5ire-bridge.py     │  ───────▶   │ MCP-aware LLM client     │
│ (sessions)   │                 │ (HTTP buffer + API)│              │ (FastMCP adapter)  │             │ (Gemini, 5ire, Claude)   │
└──────────────┘                 └────────────────────┘              └────────────────────┘             └──────────────────────────┘
```

1. **Fiddler Classic** captures live web traffic and posts JSON snapshots to enhanced-bridge.py which is a flask HTTP server. It makes the sessions available for MCP to retrieve by sending standard GET requests.
2. **enhanced-bridge.py** normalises and caches the sessions, exposing REST endpoints for search, inspection, and statistics.
3. **5ire-bridge.py** translates those REST endpoints into MCP tools and speaks the Model Context Protocol over stdin/stdout. 5ire-bridge names comes from the fact I wrote it initially for 5ire client available on GitHub it can be used with any Client which supports the open MCP protocol
4. **Gemini or any other MCP client** calls the tools in natural language: find sessions, pull headers/bodies, get stats, and so on..

Key Endpoints:

1. `fiddler_mcp__live_sessions` : Get a list of recent sessions with metadata.
2. `fiddler_mcp__sessions_search` : Search for specific sessions using filters.
3. `fiddler_mcp__session_headers` : Fetch only the HTTP headers for a captured session.
4. `fiddler_mcp__session_body` : Retrieve the actual content/payload from request and response bodies.
5. `fiddler_mcp__live_stats` : Summarise bridge health: buffer depth, capture rates, uptime, and utilisation.
6. `fiddler_mcp__sessions_timeline` : Visualise activity bursts by time, host, status, or MIME type.
7. `fiddler_mcp__sessions_clear` : Clear the rolling buffers once evidence has been exported.

Component roles at a glance

Role	Primary Responsibility	Runs Where
Fiddler Classic	Captures HTTP/S sessions and forwards them in real time.	Analyst workstation / Windows VM
enhanced-bridge.py	Buffers sessions, provides REST queries, exposes health metrics.	Python runtime (local host)
5ire-bridge.py	Implements the MCP server, maps REST calls to FastMCP tools.	Python runtime (local host)
MCP Client (Gemini, 5ire, Claude)	Presents tools to the user, asks questions, formats answers.	Analyst UI

Data flow: from session to prompt

1. **Capture** – Fiddler intercepts requests/responses. CustomRules script serialises them into JSON with metadata, headers, and optional bodies.
2. **Ingress** – The JSON payload is POSTed to http://127.0.0.1:8081/live-session (default). enhanced-bridge ingests it into ring buffers (live + suspicious) and records timestamps, risk flags, and derived fields.
3. **REST Query** – MCP bridge requests data via REST: /api/sessions, `/api/session/<id>` , `/api/stats` , etc. Only raw facts are returned—risk scoring lives in the HTTP bridge.
4. **Tool Exposure** – 5ire-bridge publishes a compact toolset (list, search, headers, bodies, stats, timeline, clear). It enforces the new MCP handshake: `initialize` → `notifications/initialized` → `tools/list` .
5. **Natural-Language Loop** – Gemini (or the MCP client) chains tools to answer questions. Example: “What data did session 240 included?” triggers `sessions_search` → `session_body` .

Components

Fiddler & Custom Rules

- Enables “Post to MCP in real-time”.
- Serialize sessions with truncated bodies (default 2 MB cap) and MIME filtering.
- Keeps user browser behaviour unchanged; everything downstream is passive analysis.

enhanced-bridge.py

- HTTP server on `127.0.0.1:8081`.
- Buffers the latest ~2000 sessions, plus a separate suspicious queue.
- Provides REST endpoints for listing, searching, fetching headers/bodies, timelines, and stats.
- Adds contextual metadata (risk flags, derived host, response size, etc.).
- Logging shows ingestion (“POST /live-session”) and HTTP clients hitting the API.

5ire-bridge.py

- Uses FastMCP to expose tools over stdin/stdout (or SSE if configured).
- Handles retries/backoff on REST calls, proxy bypass on localhost, JSON normalisation.
- Enforces handshake: respond to initialize, expect notifications/initialized, then serve requests.
- Tool list is minimal by design; each tool returns pure data so the LLM performs reasoning. I tested this with a large list of tools and the model was getting confused which tool to use. Not just Gemini 2.5 Pro but other models had the same issue.

Gemini Fiddler Client

- Launches 5ire-bridge.py as a subprocess, sends the handshake, and keeps stderr in mcp_server.err.log.
- Builds prompts that describe the toolset so the model chooses correctly (session_body for content, session_headers for metadata, etc.).
- Currently captures every MCP request/response for quick debugging in the same log file.

MCP Conversation Flow (FastMCP)

```
Client                          5ire-bridge.py                  enhanced-bridge.py
------                          ---------------                 ------------------
initialize ───────────────▶ (ack handshake) ─────────────▶     N/A
notifications/initialized ─▶ (ready state)                      N/A
tools/list ───────────────▶ (return schema)                     Optional REST lookups
tools/call(session_search) ─▶ REST GET /api/sessions/search ──▶ returns metadata
tools/call(session_body)  ──▶ REST GET /api/sessions/body/<id> ─▶ returns bodies
```

- **Timeout handling**: 5ire-bridge retries slow REST endpoints and maps failures to success: false payloads, so the LLM can explain errors.
- **Tool envelopes**: Responses conform to the FastMCP response.data structure; the Gemini client flattens them before handing data to the LLM.

Getting Started Checklist

1. **Start enhanced bridge**: python enhanced-bridge.py
2. **Confirm health**: curl http://127.0.0.1:8081/api/stats
3. **Start Gemini client**: python gemini-fiddler-client.py (first run through prompts to setup Gemini API key/model)
4. **Verify handshake**: look for "Found 7 tools" output in console after start
5. **Generate traffic**: browse through Fiddler to populate buffers
6. **Ask questions in CMD client**: “Explain the response body for session 42”

Thanks for reading,
Lucas