Design and Implementation of a Custom Network Proxy Server

1. Introduction

A network proxy server acts as an intermediary between clients and destination servers. Instead of clients communicating directly with external servers, all requests pass through the proxy, which can monitor, control, and modify traffic.

This project implements a **TCP-based HTTP forward proxy server** that receives client HTTP requests, applies configurable filtering rules, forwards allowed requests to destination servers, and relays responses back to clients. The proxy also logs all activity for monitoring and debugging purposes.

---

2. System Architecture

2.1 High-Level Architecture

The proxy server sits between the client and the destination server.

Architecture Flow:

Client → Proxy Server → Destination Server

Client ← Proxy Server ← Destination Server

2.2 Components

The system is divided into the following logical components:

1. **Listener Module**
    - Creates a TCP socket
    - Binds to a configurable address and port
    - Accepts incoming client connections
2. **Connection Handler**
    - Handles each client connection independently
    - Runs in a separate thread for concurrency
3. **HTTP Parser**
    - Reads incoming HTTP requests

- Parses request line and headers
- Extracts method, target, host, and port

4. **Filtering Module**
    - Loads blocked domains/IPs from a configuration file
    - Applies canonicalization (lowercase, trim)
    - Blocks requests matching filtering rules
5. **Forwarding Module**
    - Establishes a TCP connection to the destination server
    - Forwards client requests
    - Streams server responses back to the client
6. **Logging Module**
    - Records all proxy activity
    - Stores logs in a persistent file

---

3. Concurrency Model

3.1 Model Used

The proxy uses a **thread-per-connection** concurrency model.

3.2 Rationale

- Simple to implement and understand
- Suitable for moderate numbers of concurrent clients
- Each client connection is handled independently
- Matches the project's scope and requirements

3.3 Implementation

- When a client connects, a new thread is spawned
- The thread executes the client handling logic
- Threads are marked as daemon threads to ensure clean shutdown

---

4. Data Flow Description

4.1 Request Handling Flow

1. Client sends an HTTP request to the proxy
2. Proxy reads data until \r\n\r\n (end of headers)
3. HTTP request line and headers are parsed
4. Destination host and port are extracted
5. Filtering rules are applied
    - If blocked → return 403 Forbidden
6. If allowed → proxy connects to destination server
7. Proxy forwards the request
8. Proxy streams the response back to the client
9. Connection is closed after completion

4.2 Response Handling

- Responses are streamed in chunks
- The proxy does not buffer entire responses in memory
- This supports large responses efficiently

---

5. Filtering and Configuration

5.1 Filtering Rules

Filtering rules are defined in a text file:

config/blocked_domains.txt

Each line contains:

- A domain name or IP address
- One entry per line

5.2 Filtering Logic

- Hostnames are normalized (lowercase, trimmed)
- Exact and suffix matching is supported
- Example: blocking example.com also blocks sub.example.com

5.3 Blocked Request Handling

- Blocked requests return:

HTTP/1.1 403 Forbidden

- The event is logged with action = blocked

---

6. Logging Mechanism

6.1 Log File

Logs are written to:

logs/proxy.log

6.2 Logged Information

Each log entry includes:

- Timestamp (UTC)
- Client IP and port
- Destination host and port
- Action taken (allowed / blocked / error)
- HTTP response status
- Bytes transferred

6.3 Example Log Entry

2026-01-02T19:22:32Z client=127.0.0.1:53456 dest=example.com:80 action=blocked status=403

---

7. Error Handling

The proxy handles errors gracefully:

- **403 Forbidden** for blocked domains

- **502 Bad Gateway** for forwarding or connection failures
- **501 Not Implemented** for unsupported methods (e.g., CONNECT)

Sockets are closed properly to avoid resource leaks.

---

8. Testing Strategy

8.1 Functional Testing

- Verified request forwarding using curl
- Verified domain blocking behavior
- Verified correct HTTP status codes

8.2 Concurrency Testing

- Multiple requests issued sequentially and concurrently
- Each client handled independently without interference

8.3 Logging Verification

- Confirmed logs are written for every request
- Verified correct action and status entries

---

9. Limitations

- HTTPS CONNECT tunneling is not implemented
- Chunked transfer encoding is not explicitly parsed
- Persistent HTTP connections are not supported
- Designed for educational and moderate-load scenarios

---

10. Conclusion

This project successfully demonstrates the design and implementation of a custom HTTP forward proxy server using TCP sockets. The proxy correctly handles concurrent clients, parses and forwards HTTP requests, applies filtering rules, logs activity, and returns appropriate

responses. The modular design allows future extensions such as HTTPS tunneling, caching, and authentication.

SECTION 2

Demonstration and Testing

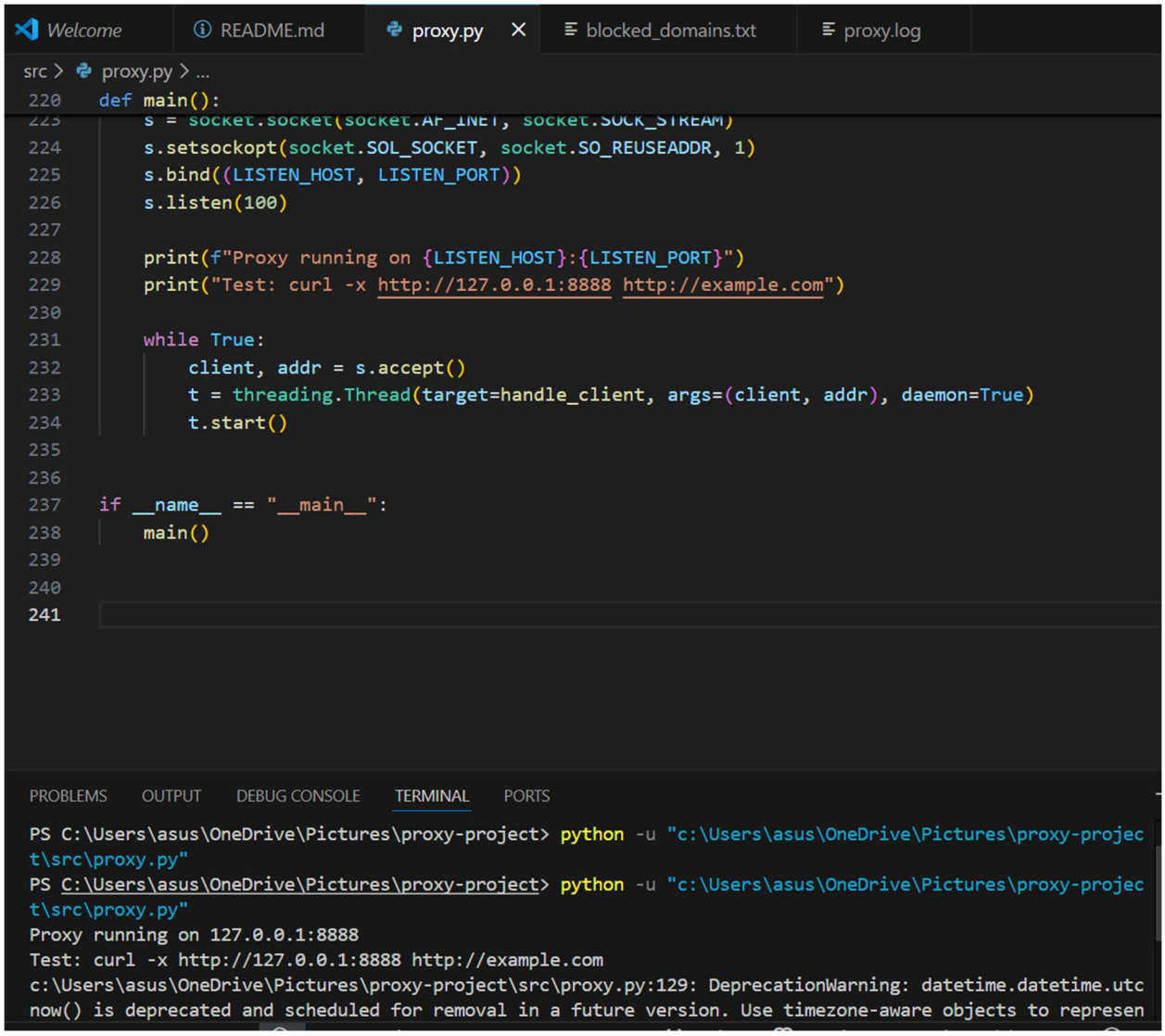

Figure 1: Proxy server successfully started and listening on port 8888.

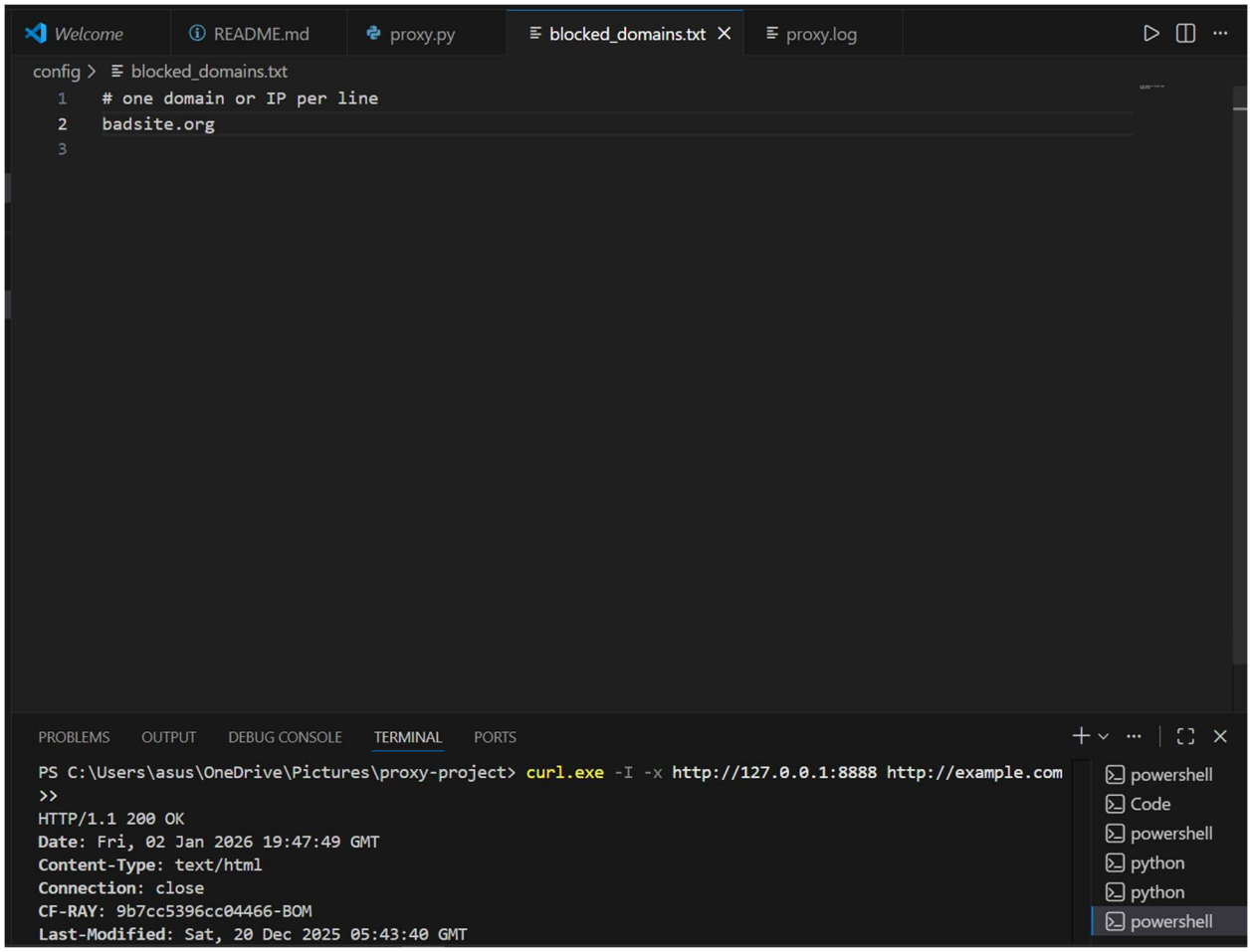

Figure 2: Successful forwarding of an HTTP request through the proxy server for an allowed domain, returning HTTP 200 OK.

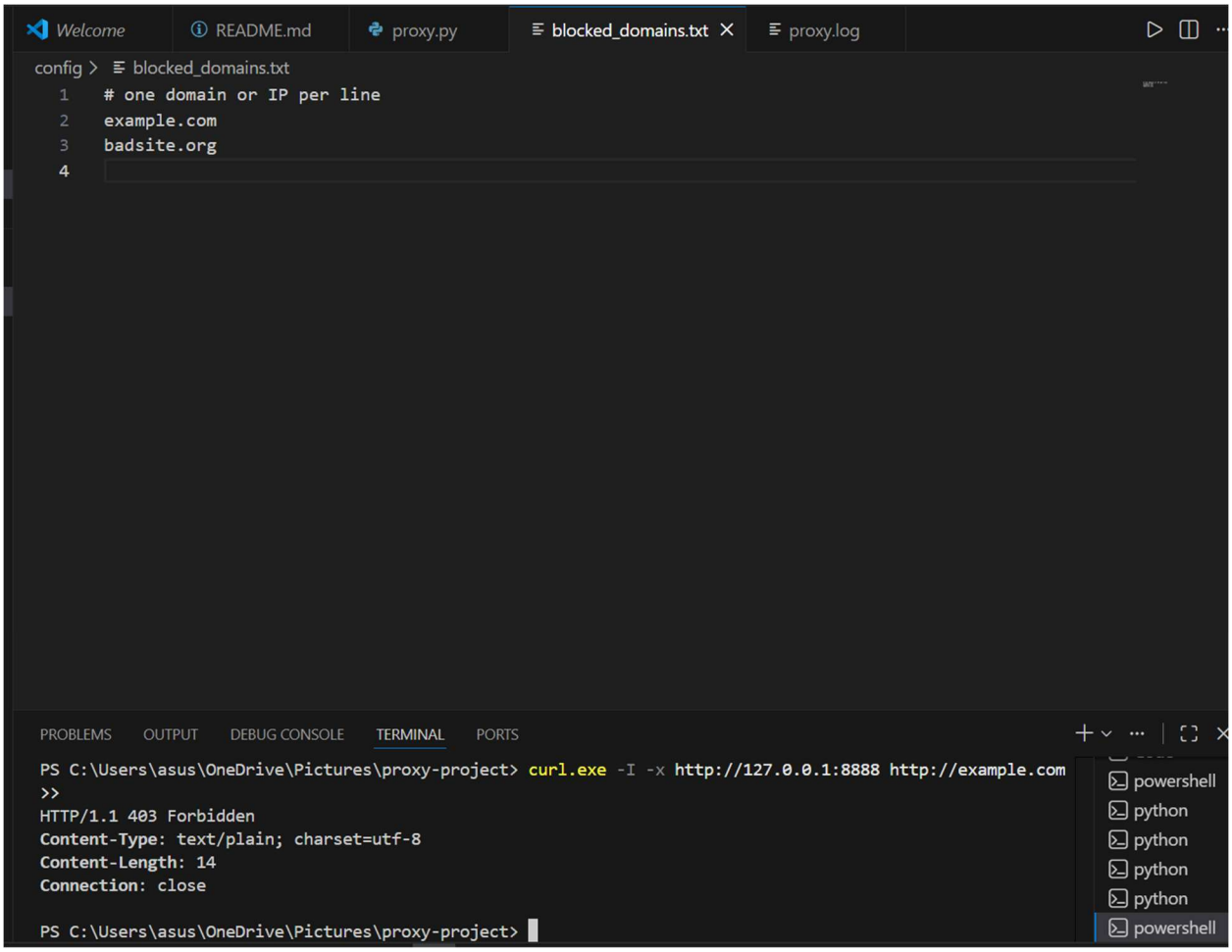

Figure 3: Proxy server blocking access to a restricted domain and returning HTTP 403 Forbidden.

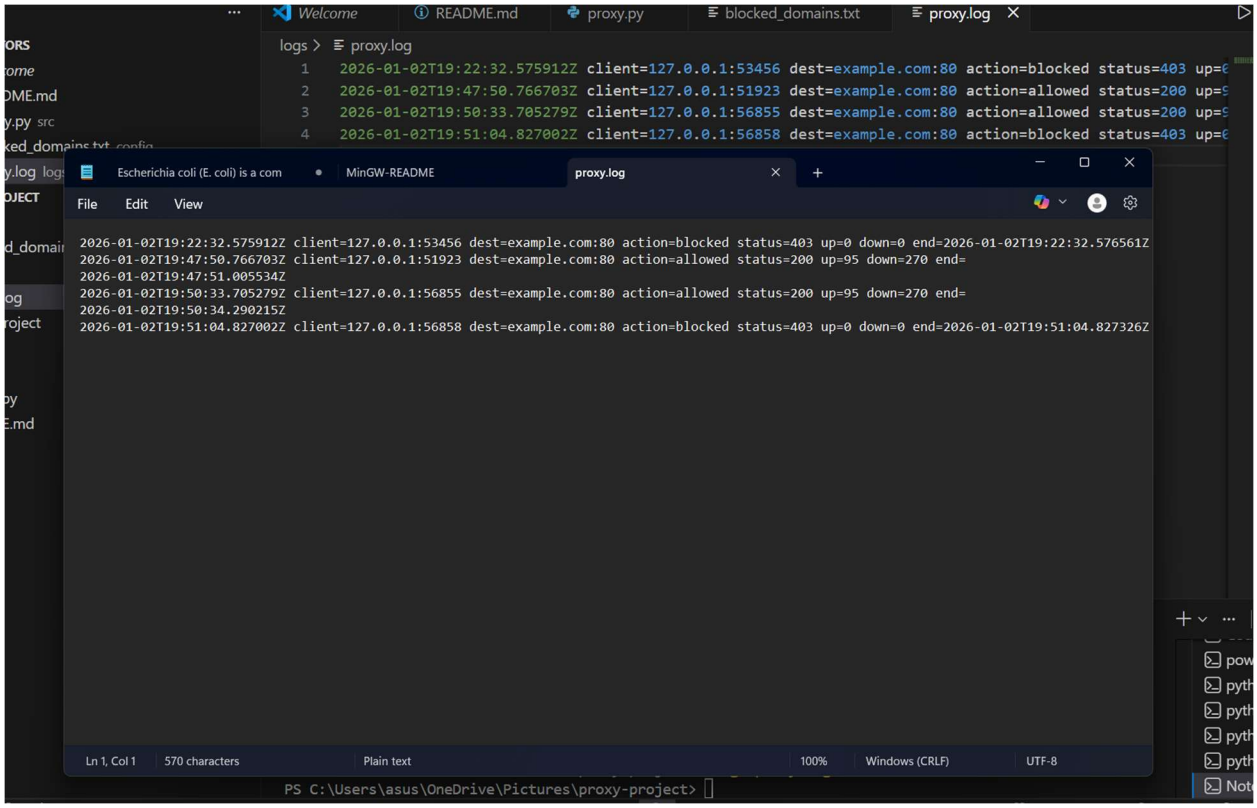

Figure 4: Proxy server log file showing recorded HTTP requests, including blocked requests with action marked as blocked and HTTP 403 status.