"BENIGN" or "SUSPICIOUS")

**1. Update Logging Setup**

* **Configure a file handler** for detailed logs.
* **Set the stream handler** (for terminal) to only output the request status.

**Example Changes**

**Logging Setup Section**

# ------------------ LOGGING SETUP ------------------  
  
logger = logging.getLogger("pywaf")  
logger.setLevel(logging.INFO)  
  
# File handler for all logs  
fh = logging.FileHandler('pywaf.log')  
fh.setLevel(logging.INFO)  
file\_formatter = logging.Formatter('%(asctime)s %(levelname)s %(message)s')  
fh.setFormatter(file\_formatter)  
logger.addHandler(fh)  
  
# Stream handler for minimal terminal output  
ch = logging.StreamHandler()  
ch.setLevel(logging.INFO)  
# Only log the status line to terminal, so use a custom filter  
class StatusOnlyFilter(logging.Filter):  
 def filter(self, record):  
 # Only allow status logs to the terminal  
 return getattr(record, 'status\_only', False)  
ch.addFilter(StatusOnlyFilter())  
ch.setFormatter(logging.Formatter('%(message)s'))  
logger.addHandler(ch)

**2. Adjust Logging Calls**

* **Add a status\_only attribute** to status log records for terminal output.
* **Send all details to the file logger** as before.

**Example Usage in data\_received**

Replace the terminal printout and logging section with:

now = datetime.now().strftime('%Y-%m-%d %H:%M:%S')  
src\_ip = self.peername[^1\_0] if self.peername else 'unknown'  
status = "SUSPICIOUS" if attack\_detected else "BENIGN"  
status\_line = f"[{now}] {src\_ip} {status} - {req\_obj.method} {req\_obj.url}"  
  
# Log status to terminal  
logger.info(status\_line, extra={'status\_only': True})  
  
# Log full details to file  
logger.info(f"[{now}] {src\_ip} {status} Request: {req\_obj.method} {req\_obj.url}")  
logger.info(f"Headers: {headers}")  
if attack\_detected:  
 logger.warning(f"Attack Detected: {[r.name for r in detected\_rules]}")

**3. Remove Direct print() Calls**

* Eliminate or comment out any print() statements related to request status, as logging now handles terminal output.

**4. Result**

* **All logs** (detailed) go to pywaf.log.
* **Terminal** displays only the request status (one line per request).

⁂

**Logging to Both File and Terminal in Python**

**Example Code**

import logging  
  
# Create a logger  
logger = logging.getLogger("waf\_logger")  
logger.setLevel(logging.INFO)  
  
# File handler for logging to a file  
file\_handler = logging.FileHandler('waf\_smuggling.log')  
file\_handler.setLevel(logging.INFO)  
file\_formatter = logging.Formatter('%(asctime)s %(message)s')  
file\_handler.setFormatter(file\_formatter)  
logger.addHandler(file\_handler)  
  
# Stream handler for printing to terminal  
console\_handler = logging.StreamHandler()  
console\_handler.setLevel(logging.INFO)  
console\_formatter = logging.Formatter('%(asctime)s %(message)s')  
console\_handler.setFormatter(console\_formatter)  
logger.addHandler(console\_handler)  
  
# Example log entry  
reason = "CL-TE header smuggling"  
self\_path = "/vulnerable"  
self\_headers = {"Content-Length": "10", "Transfer-Encoding": "chunked"}  
log\_entry = f"[WAF] HTTP Request Smuggling detected: {reason} | Path: {self\_path} | Headers: {self\_headers}"  
  
logger.info(log\_entry)

With this setup, every time you call logger.info(log\_entry), the message will be written to waf\_smuggling.log and also printed to the terminal, both with timestamps and your custom message format[[4]](#fn4)[[5]](#fn5)[[6]](#fn6).

**Key Points**

* **File Logging:** All log entries are saved to waf\_smuggling.log for future analysis[[5]](#fn5)[[6]](#fn6).
* **Terminal Output:** The same log entries are printed to the terminal in real time for immediate visibility[[4]](#fn4)[[5]](#fn5).
* **No Need for print():** Using the logger with both handlers replaces the need for manual print() statements for these logs