4.1.3.1 Core Scanning Architecture

### **4.1.3.1 Core Scanning Architecture**

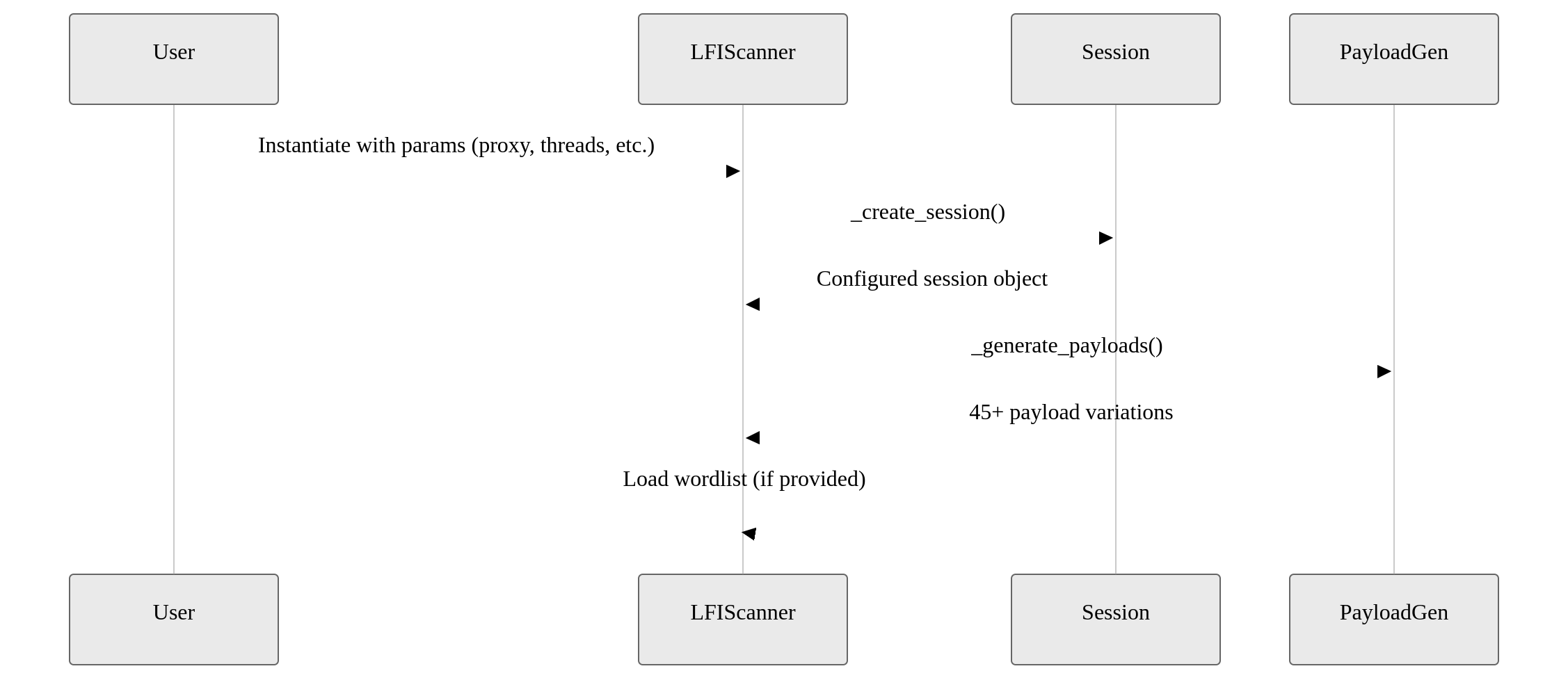
The Path Traversal Scanner employs a **multi-layered object-oriented architecture** designed for extensibility and efficient vulnerability discovery. Built around the LFIScanner class, it integrates five key subsystems:

4.1.3.1 Core Scanning Architecture-20250502214124612.webp

#### **1. Class Structure & Key Components**

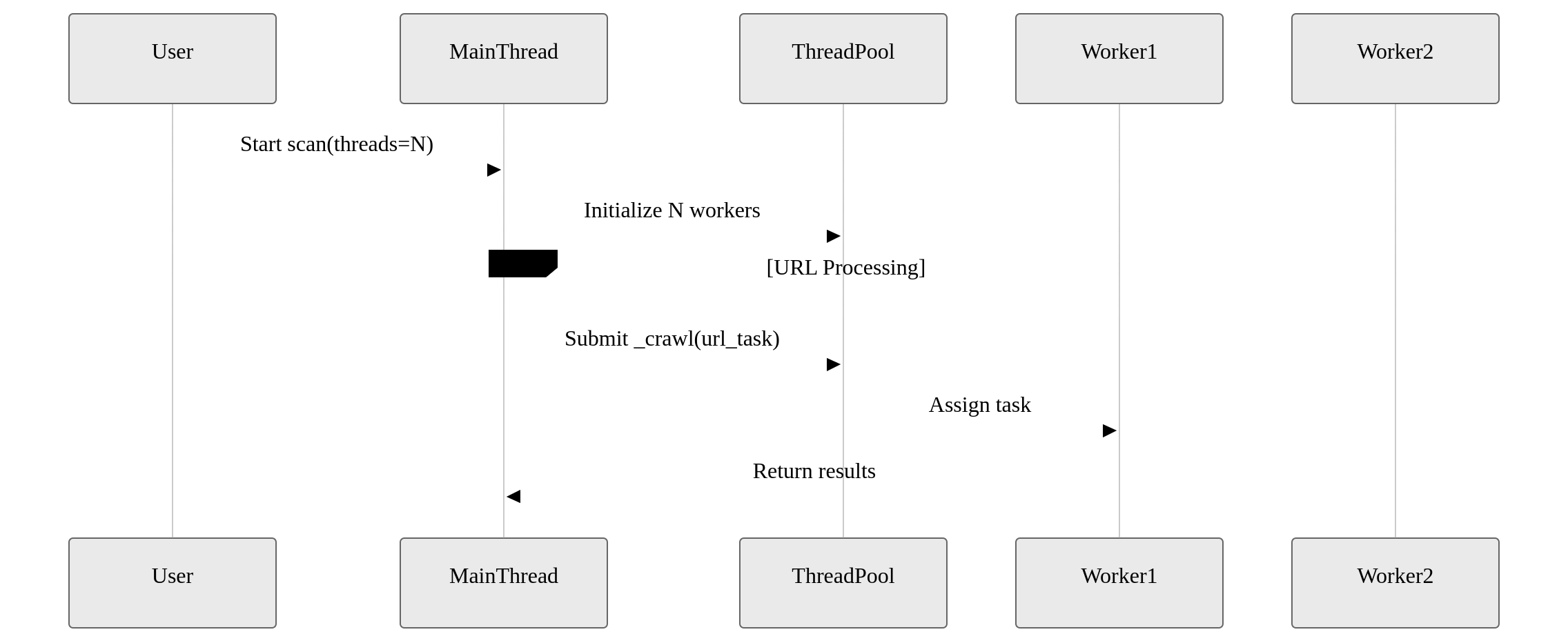
| Component | Type | Purpose | Implementation Details |
| --- | --- | --- | --- |
| session | Instance | Manages HTTP state | Configured with retry logic (3 attempts), browser-like headers, and proxy support |
| executor | ThreadPool | Concurrent execution | Fixed worker threads (default=10) with FIFO task queue |
| payloads | List | Attack vectors | Pre-generated during initialization via \_generate\_payloads() |
| vulnerabilities | List | Finding storage | Deduplicated via hash-based \_add\_vulnerability() method |

#### **2. Initialization Workflow**



#### **3. Critical Design Decisions** (Revised)

1. **Adaptive Concurrency Model**  
   Leverages Python's ThreadPoolExecutor with:

* - \*\*User-Defined Thread Scaling\*\*: Configurable via `--threads` parameter (default=10)  
     
  - \*\*Dynamic Task Queueing\*\*: Asynchronous I/O-bound task scheduling  
     
  - \*\*Optimal Resource Utilization\*\*: Automatic workload distribution across threads  
     
  - \*\*Thread-Safe Operations\*\*: Deduplication via synchronized sets
* 

#### **4. Architectural Advantages** (Revised)

| Design Feature | Benefit | Implementation Advantage |
| --- | --- | --- |
| Configurable Thread Pool | Adapts to hardware capabilities | Users balance speed/resource usage via --threads |
| Pre-generated Payloads | Immediate test availability | 45+ attack vectors loaded at initialization |
| Session Reuse | Consistent HTTP fingerprint | Maintains cookies/headers across requests |
| Dual-Phase Storage | Real-time analysis | In-memory findings allow instant processing |

#### **Enhanced Concurrency Details**

* **Thread Allocation Formula**:
  + 1 control thread orchestrates crawling
    - N-1 workers handle parameter testing
* **Memory Optimization**:
  + Shared vulnerability store with thread-safe locks
  + Batched result processing every 200ms

#### **5. Key Enhancements**

* **Live Thread Adjustment**: While not dynamically scaling mid-scan, users can:

# Start with conservative threads  
./scanner.py --url example.com --threads 10

# Subsequent scan with increased capacity   
./scanner.py --url example.com --threads 25

* **Context-Aware Execution**:
  + I/O-bound tasks never block main thread
  + CPU-intensive ops (payload generation) run pre-scan