OSINT Tools Configuration and Usage Guide

Overview

This guide explains how each OSINT tool works and how to configure them for optimal parallel execution.

Tool Descriptions and Configurations

1. Sherlock (Username Search)

- **Purpose**: Searches for usernames across 300+ social media sites
- **Best for**: Finding social media profiles when you only have a name
- **Configuration**:

  python

  ```python
  # In config file:
  enable_sherlock: true
  sherlock_timeout: 20  # seconds per username
  sherlock_sites_limit: 50  # limit sites to check for speed
  ```

- **Optimization Tips**:
  - Limit username variations to 3-4 to prevent timeouts
  - Use `--timeout 5` flag to skip slow sites
  - Consider using `--site-list` with priority sites

2. Photon (Web Crawler)

- **Purpose**: Fast web crawler that extracts URLs, emails, social media accounts, files, secret keys, and subdomains
- **Best for**: Deep crawling of discovered websites
- **Configuration**:

  python

  ```python
  enable_photon: true
  photon_level: 2  # crawl depth
  photon_threads: 4  # parallel threads
  photon_timeout: 3  # timeout per request
  ```

- **Optimization Tips**:
  - Keep crawl level low (1-2) for speed
  - Use `--dns` flag to enumerate subdomains

  - Enable `--keys` to find API keys

3. theHarvester (Email/Domain Gathering)

- **Purpose**: Gathers emails, subdomains, hosts, employee names from public sources
- **Best for**: Corporate/organizational intelligence when you have a domain
- **Configuration**:

  ```python
  enable_harvester: true
  harvester_sources: ['google', 'bing', 'linkedin', 'twitter']
  harvester_limit: 100  # results limit
  ```

- **Optimization Tips**:
  - Focus on specific sources instead of 'all' for speed
  - Best when you have email domain information

4. DaProfiler (Social Media Profiling)

- **Purpose**: Automated profile search across major social platforms
- **Best for**: Quick social media profile discovery
- **Configuration**:

  ```python
  enable_daprofiler: true
  daprofiler_platforms: ['facebook', 'twitter', 'instagram', 'linkedin']
  daprofiler_use_selenium: false  # set true for deeper search
  ```

- **Requirements**:
  - Needs geckodriver.exe (already in the folder)
  - May require Firefox installed for Selenium mode

5. Proton (Advanced Search Aggregator)

- **Purpose**: Multi-engine search aggregator with result ranking
- **Best for**: Comprehensive web search with result prioritization
- **Configuration**:

  ```python
  enable_proton: true
  proton_engines: ['google', 'bing', 'duckduckgo', 'yandex']
  proton_max_results: 20
  ```

6. snscrape (Social Media Scraper)

- **Purpose**: Scrapes social media platforms without API limits
- **Best for**: Twitter, Facebook, Instagram content extraction
- **Configuration**:

  python

  ```
  enable_snscrape: true
  snscrape_platforms: ['twitter-search', 'facebook-user', 'instagram-user']
  snscrape_max_results: 50
  ```

- **Optimization Tips**:
  - Twitter search is fastest and most reliable
  - Use date filters to limit results

7. Twint (Twitter Intelligence)

- **Purpose**: Advanced Twitter scraping without API
- **Best for**: Deep Twitter analysis, historical tweets
- **Configuration**:

  python

  ```
  enable_twint: true
  twint_limit: 20  # tweets per search
  twint_since: "2023-01-01"  # optional date filter
  ```

- **Note**: Twint may have issues with recent Twitter changes

8. Tookie (Multi-purpose OSINT)

- **Purpose**: All-in-one OSINT framework
- **Best for**: Comprehensive searches across multiple data types
- **Configuration**:

  python

  ```
  enable_tookie: true
  tookie_modules: ['whois', 'dns', 'social', 'email']
  tookie_output_format: 'json'
  ```

Parallel Execution Strategy

1. Thread Pool Configuration

python

```python
# In ScraperOrchestrator.__init__
self.executor = concurrent.futures.ThreadPoolExecutor(
    max_workers=10  # Adjust based on system resources
)
```

2. Resource Management

- **CPU-bound tools**: Sherlock, DaProfiler, Tookie
- **I/O-bound tools**: Web scrapers, Photon, snscrape
- **Memory-intensive**: Twint, theHarvester with large domains

3. Timeout Strategy

python

```python
scraper_timeouts = {
    'google_search': 10,
    'sherlock': 30,
    'photon': 30,
    'theharvester': 30,
    'daprofiler': 30,
    'proton': 20,
    'snscrape': 20,
    'twint': 25,
    'tookie': 30
}
```

Installation Requirements

1. Python Dependencies

bash

```bash
# Install all dependencies
pip install -r requirements.txt

# Tool-specific installations
pip install sherlock
pip install snscrape
pip install twint==2.1.21  # Specific version for stability
```

2. System Requirements

- Python 3.8+

- Firefox (for DaProfiler Selenium mode)
- 4GB+ RAM for parallel execution
- Stable internet connection

3. Environment Setup

```bash
# Add tools to PATH
export PATH=$PATH:/path/to/search_methods_2/sherlock
export PATH=$PATH:/path/to/search_methods_2/DaProfiler

# For Windows
set PATH=%PATH%;C:\path\to\search_methods_2\sherlock
```

Usage Example

```python
from scraper_orchestrator import ScraperOrchestrator, SearchQuery
from config_module import Config

# Initialize config
config = Config()
config.scraper.enable_sherlock = True
config.scraper.enable_photon = True
config.scraper.enable_harvester = True
config.scraper.enable_daprofiler = True
config.scraper.enable_proton = True
config.scraper.enable_snscrape = True
config.scraper.enable_twint = True
config.scraper.enable_tookie = True

# Create orchestrator
orchestrator = ScraperOrchestrator(config)

# Create search query
query = SearchQuery(
    first_name="John",
    last_name="Doe",
    activity="software engineer",
    location="San Francisco",
    additional_info={
        'email': 'john.doe@example.com',
        'domain': 'example.com'
    }
)

# Run async search
import asyncio
results = asyncio.run(orchestrator.search(query, timeout=60))

# Get summary
summary = orchestrator.get_summary()
print(f"Found {summary['total_urls_found']} URLs across {summary['successful_scrapers']} scrapers")
```

Performance Optimization Tips

1. **Parallel Execution**:
    - Use asyncio for I/O-bound operations
    - ThreadPoolExecutor for CPU-bound tools
    - Process pool for truly independent tools

2. **Rate Limiting**:
    - Implement delays between requests to avoid IP bans
    - Use rotating user agents
    - Consider proxy rotation for large-scale operations
3. **Result Caching**:
    - Cache search results to avoid duplicate queries
    - Store successful username lookups
    - Save discovered domains for theHarvester
4. **Error Handling**:
    - Implement retry logic with exponential backoff
    - Log failed searches for manual review
    - Continue execution even if individual tools fail
5. **Resource Management**:
    - Monitor memory usage, especially with Twint
    - Limit concurrent executions based on system resources
    - Use streaming for large result sets

Troubleshooting

Common Issues:

1. **Sherlock timeout**: Reduce sites checked or increase timeout
2. **Photon memory issues**: Reduce crawl depth or thread count
3. **theHarvester blocked**: Use API keys for search engines
4. **DaProfiler geckodriver**: Ensure geckodriver.exe is in PATH
5. **Twint errors**: Twitter changes frequently break Twint
6. **snscrape rate limits**: Add delays between searches

Debug Mode:

```python
import logging
logging.basicConfig(level=logging.DEBUG)
```

Security Considerations

1. **Legal Compliance**:
    - Respect robots.txt

   - Follow platform ToS
   - Use for legitimate OSINT only
2. **Operational Security**:
   - Use VPN for sensitive searches
   - Rotate IP addresses
   - Don't search for yourself first
3. **Data Handling**:
   - Encrypt stored results
   - Limit PII collection
   - Implement data retention policies