IOC Threat Intelligence Correlation Engine

Complete Code Documentation

Filename:	app.py
Total Lines:	398
Language:	Python 3.12+
Generated:	2025-08-11 16:08:22
Documentation Type:	Line-by-line Analysis

Table of Contents

1. Import Statements (Lines 1-9)

Essential libraries for HTTP requests, file handling, validation, and time management.

2. Configuration Section (Lines 11-19)

Centralized settings for API keys, rate limits, and output preferences.

3. IOCAnalyzer Class (Lines 21-24)

Main analysis engine with result storage and API call tracking.

4. IOC Validation (Lines 26-52)

Type detection and format validation for IPs, domains, and file hashes.

5. API Request Handler (Lines 54-83)

Generic HTTP client with rate limiting and error handling.

6. VirusTotal Integration (Lines 85-125)

File and domain analysis using VirusTotal API v3.

7. AbuseIPDB Integration (Lines 127-157)

IP reputation checking with abuse confidence scoring.

8. Shodan Integration (Lines 159-188)

Network intelligence gathering for open ports and vulnerabilities.

9. Risk Scoring System (Lines 190-226)

Composite risk calculation across multiple threat intelligence sources.

10. IOC Processing (Lines 228-255)

Main orchestration logic routing IOCs to appropriate APIs.

11. Report Generation (Lines 257-335)

Markdown and CSV output with risk-based sorting.

12. Configuration Validation (Lines 337-352)

API key validation with user-friendly warnings.

13. Main Application (Lines 354-398)

Command-line interface and workflow orchestration.

Key Code Implementations

IOC Validation Logic

```
def validate_ioc(self, ioc): ioc = ioc.strip() # IP Address Check try: ipaddress.ip_address(ioc) return ('ip', ioc) except ValueError: pass # Domain Check domain_regex = r'^([a-z0-9]+(-[a-z0-9]+)*\.)+[a-z]{2,}$' if re.match(domain_regex, ioc, re.IGNORECASE): return ('domain', ioc) # Hash Check hash_lengths = {32: 'md5', 40: 'sha1', 64: 'sha256'} clean_ioc = ioc.lower().replace('-', '') if len(clean_ioc) in hash_lengths and all(c in '0123456789abcdef' for c in clean_ioc): return (hash_lengths[len(clean_ioc)], clean_ioc) return (None, ioc)
```

Risk Scoring Algorithm

```
def calculate_risk_score(self, result): score = 0 if result['source'] == 'VirusTotal': malicious = result.get('malicious', 0) if malicious >= CONFIG['malicious_threshold']: score = 100 elif malicious > 0: score = 60 + (malicious * 5) elif result['source'] == 'AbuseIPDB': confidence = result.get('abuse_confidence', 0) if confidence >= CONFIG['high_abuse_score']: score = 90 + (reports * 0.1) return min(100, int(score))
```

API Request with Rate Limiting

```
def make_api_request(self, url, headers=None, params=None): if self.api_call_count > 0 and CONFIG['rate_limit_delay'] > 0: time.sleep(CONFIG['rate_limit_delay']) try: response = requests.get(url, headers=headers, params=params) self.api_call_count += 1 if response.status_code == 429: time.sleep(60) # Rate limit backoff response = requests.get(url, headers=headers, params=params) response.raise_for_status() return response.json() except requests.exceptions.RequestException as e: print(f"API request failed: {e}") return None
```

Technical Summary

- 398 lines of production-ready Python code
- Integration with 3 major threat intelligence APIs
- Comprehensive error handling and rate limiting
- Automatic IOC type detection and validation
- Risk scoring algorithm across multiple data sources
- Multiple output formats (Markdown and CSV)
- Command-line interface with configuration validation
- Modular design for easy extension and maintenance