Plugin History Analysis Tool - Complete User Guide

Plugin History Analysis Tool

Version 2.0 - December 2025

Plugin History Analysis Tool - Complete User Guide

Version 2.0 | A comprehensive vulnerability management and analysis system for Nessus/Tenable scan data.

Table of Contents

Overview

The Plugin History Analysis Tool transforms raw vulnerability scan data into actionable security intelligence. It processes Nessus/Tenable scan data to provide:

Core Capabilities

Category	Features
Lifecycle Tracking	Track findings from detection to resolution, reappearance detection, MTTR calculation
Host Analysis	Scan coverage, missing host detection, host churn analysis
Compliance	OPDIR/IAVM mapping, due date tracking, compliance status
Remediation	Package version impact, prioritization, effort estimation
Filtering	13 filter operators, environment classification, custom lists
Visualization	30+ chart types, interactive dashboards, export to PDF/PNG
AI Integration	OpenWebUI predictions, trend analysis, prioritization recommendations
Threat Intel	CISA KEV, EPSS scores, NVD integration, DISA IAVM feeds
Export	Excel, SQLite, JSON, CSV, PDF chart export

Installation & Getting Started

Prerequisites

```
# Required Python packages
pip install pandas numpy matplotlib openpyxl requests

# Optional for AI features
pip install openai tiktoken
```

Directory Structure

```
refactored_app/
├── analysis/          # Core analysis modules
├── core/              # Data parsing and processing
├── export/            # Export functionality
├── filters/           # Filtering engine
├── gui/               # GUI components
├── models/            # Data models
├── statistics/        # Aggregation functions
├── visualization/     # Charts and dashboards
└── ai/                # AI integration
```

Launching the Application

GUI Mode:

```
python -m refactored_app.main
```

Programmatic Use:

```
from refactored_app.gui import NessusHistoryTrackerApp
import tkinter as tk

root = tk.Tk()
app = NessusHistoryTrackerApp(root)
root.mainloop()
```

Loading Data

Supported Input Formats

Format	Description	Menu Path
`.nessus`	Native Nessus XML files	File → Load Nessus Files
`.zip`	Archives containing .nessus files	File → Load Nessus Archive
`.xlsx/.csv`	Tenable exports	File → Load Export File
`.db`	SQLite databases	File → Load Database
`plugins.xml`	Nessus plugin database	File → Load Plugin Database

Loading Nessus Archives

1. **File → Load Nessus Archive**
2. Select `.zip` file containing `.nessus` files
3. System automatically:
   - Extracts nested archives recursively
   - Parses all `.nessus` XML files
   - Extracts scan dates per host
   - Maps hostnames and IP addresses
   - Extracts CVE/IAVA references

Data Enrichment

Plugin Database:
- Load `plugins.xml` from Nessus installation
- Adds: solution text, CVE references, exploit availability, CVSS details

OPDIR Mapping:
- Load OPDIR Excel/CSV file
- Adds: OPDIR number, due dates, compliance status

IAVM Notices:
- Load IAVM summary file
- Adds: IAVM references, STIG severity, supersession chains

Finding Lifecycle Analysis

The lifecycle module tracks each vulnerability through its complete history.

Lifecycle Fields

Field	Description
`first_seen`	First scan date where finding appeared
`last_seen`	Most recent detection date
`days_open`	Duration from first detection to resolution/current
`status`	**Active** (still present) or **Resolved** (no longer detected)
`reappearances`	Times finding returned after resolution
`total_observations`	Total scan instances with this finding

Key Functions

```
from refactored_app.analysis import (
    analyze_finding_lifecycle,
    identify_reappearances,
    calculate_mttr,
    get_findings_by_age
)

# Full lifecycle analysis
lifecycle_df = analyze_finding_lifecycle(historical_df)

# Find reappeared findings (gap > 45 days)
reappeared = identify_reappearances(historical_df, gap_days=45)

# Calculate MTTR by severity
mttr = calculate_mttr(lifecycle_df, group_by='severity_text')

# Age buckets (0-30, 31-60, 61-90, 91-120, 121+)
age_buckets = get_findings_by_age(lifecycle_df)
```

Reappearance Detection

A finding is marked as "reappeared" when:
1. It was resolved (not seen for 45+ days)
2. It reappears in a later scan
3. The gap exceeds `REAPPEARANCE_GAP_DAYS` (configurable)

Host Presence Tracking

Track which hosts appear in which scans to identify coverage gaps.

Analysis Functions

```
from refactored_app.analysis import (
    create_host_presence_analysis,
    identify_missing_hosts,
    identify_unreliable_hosts,
    calculate_scan_coverage,
    identify_new_hosts
)

# Full host presence matrix
host_presence = create_host_presence_analysis(historical_df)

# Hosts missing from recent scans (last 2 scans)
missing = identify_missing_hosts(historical_df, threshold=2)

# Hosts with < 75% scan coverage
unreliable = identify_unreliable_hosts(historical_df, threshold=0.75)

# Overall coverage percentage
coverage = calculate_scan_coverage(historical_df)

# New hosts in latest scan
new_hosts = identify_new_hosts(historical_df)
```

Host Presence Matrix

The presence matrix shows:
- Rows: Individual hosts
- Columns: Scan dates
- Values: Present (1) or Missing (0)

Scan Changes Analysis

Analyze changes between consecutive scans.

Change Types

Metric	Description
Hosts Added	New hosts appearing in scan
Hosts Removed	Hosts no longer in scan
Host Churn Rate	(Added + Removed) / Total Hosts
New Findings	Vulnerabilities appearing for first time
Resolved Findings	Vulnerabilities no longer detected
Severity Changes	Changes in severity distribution

Functions

```
from refactored_app.analysis import (
    analyze_scan_changes,
    analyze_finding_changes,
    calculate_host_churn,
    get_scan_summary
)

# Host changes between scans
host_changes = analyze_scan_changes(historical_df)

# Finding changes with severity breakdown
finding_changes = analyze_finding_changes(historical_df)

# Churn statistics
churn = calculate_host_churn(historical_df)

# Per-scan summary
summary = get_scan_summary(historical_df)
```

Advanced Metrics

Industry-standard vulnerability management metrics.

Available Metrics

Metric	Description	Good Value
MTTR	Mean Time to Remediation	Critical < 15 days
Reopen Rate	% of findings that reappear	< 5%
Coverage	% of assets scanned	> 95%
MTTD	Mean Time to Detect	< 30 days
Remediation Rate	Resolved / Discovered	> 80%
SLA Compliance	% within SLA	> 90%

Calculating All Metrics

```
from refactored_app.analysis import get_all_advanced_metrics

metrics = get_all_advanced_metrics(
    historical_df=historical_df,
    lifecycle_df=lifecycle_df,
    sla_targets={
        'Critical': 15,
        'High': 30,
        'Medium': 60,
        'Low': 90
    }
)

# Access individual metrics
print(f"Reopen Rate: {metrics['reopen_rate']['reopen_rate_pct']}%")
print(f"Coverage: {metrics['coverage']['coverage_pct']}%")
print(f"Remediation Rate: {metrics['remediation_rate']['remediation_rate_pct']}%")
```

SLA Breach Tracking

```
from refactored_app.analysis import calculate_sla_breach_tracking

sla_status = calculate_sla_breach_tracking(
    lifecycle_df,
    sla_targets={'Critical': 15, 'High': 30, 'Medium': 60, 'Low': 90}
)

# Results include:
# - breached: Count of SLA violations
# - at_risk: Within 25% of SLA deadline
# - on_track: Compliant
# - by_severity: Breakdown per severity
```

OPDIR/IAVM Compliance

Track compliance with OPDIR mandates and IAVM notices.

OPDIR Mapping

```
from refactored_app.analysis import (
    load_opdir_mapping,
    enrich_with_opdir,
    get_opdir_compliance_summary,
    get_overdue_findings
)

# Load OPDIR file
opdir_df = load_opdir_mapping('opdir_mapping.xlsx')

# Enrich findings with OPDIR data
enriched_df = enrich_with_opdir(
    lifecycle_df,
    opdir_df,
    iavx_column='iavx'
)

# Get compliance summary
summary = get_opdir_compliance_summary(enriched_df)

# Get overdue findings
overdue = get_overdue_findings(enriched_df)
```

OPDIR Status Types

Status	Meaning	Color
On Track	Due date > 7 days away	🟢 Green
Due Soon	Due date within 7 days	🟡 Yellow
Overdue	Past due date	🔴 Red
Unknown	No OPDIR mapping found	⚪ Gray

IAVM Integration

```
from refactored_app.analysis import (
    load_iavm_summaries,
    enrich_findings_with_iavm,
    merge_opdir_and_iavm
)

# Load IAVM notices
iavm_df = load_iavm_summaries('iavm_notices.xlsx')

# Enrich with IAVM data
enriched = enrich_findings_with_iavm(lifecycle_df, iavm_df)

# Merge OPDIR and IAVM for unified compliance view
unified = merge_opdir_and_iavm(opdir_df, iavm_df)
```

Package Version Impact Analysis

Identify highest-impact package upgrades for remediation prioritization.

Loading Version Data

The system accepts output from the Tenable Version Extractor or compatible CSV/Excel:

Required Column	Description
`Component_Name`	Package name
`Installed_Version`	Current version
`Should_Be_Version` or `Fixed_Version`	Target version
`Hostname`	Affected host

Recommended Column	Description
`Plugin_ID`	Nessus plugin ID
`Severity`	Finding severity
`CVEs`	Associated CVE IDs

Impact Analysis

```
from refactored_app.analysis import (
    analyze_package_version_impact,
    create_remediation_summary_df,
    calculate_cumulative_impact,
    estimate_remediation_effort,
    export_remediation_plan
)

# Analyze version data
plan = analyze_package_version_impact(version_df, findings_df)

# Create summary DataFrame
summary_df = create_remediation_summary_df(plan)

# Calculate cumulative impact (80/20 analysis)
cumulative = calculate_cumulative_impact(plan)

# Estimate effort
effort = estimate_remediation_effort(plan)
print(f"Effort Level: {effort['effort_level']}")
print(f"Recommendations: {effort['recommendations']}")

# Export to Excel
export_remediation_plan(plan, 'remediation_plan.xlsx', format='xlsx')
```

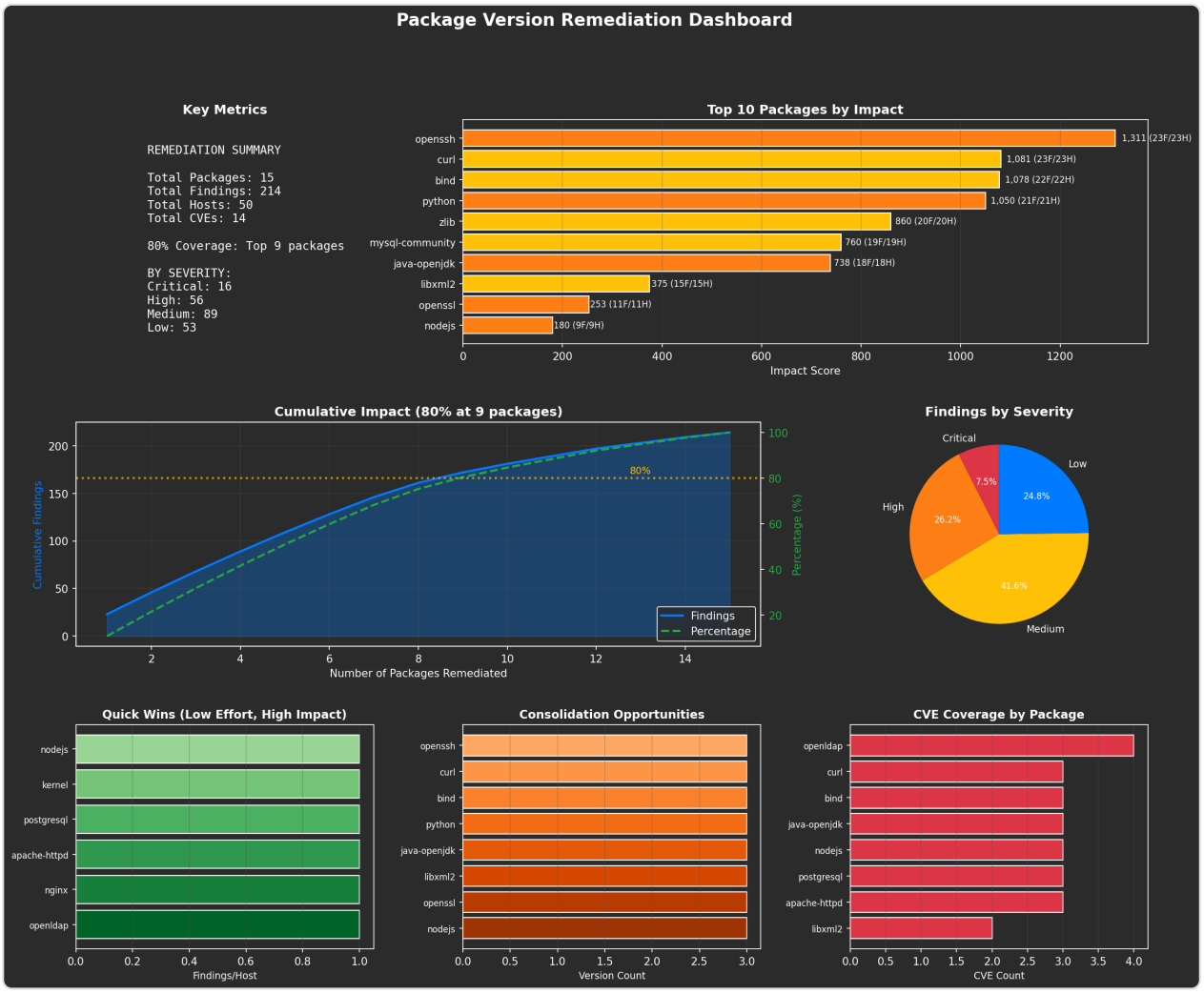
Impact Calculation Formula

```
Impact Score = (Critical×4 + High×3 + Medium×2 + Low×1) × Unique Hosts
```

Example: OpenSSL upgrade affecting 25 hosts with 15 Critical, 20 High, 10 Medium findings:

```
Impact = (15×4 + 20×3 + 10×2) × 25 = (60 + 60 + 20) × 25 = 3,500
```

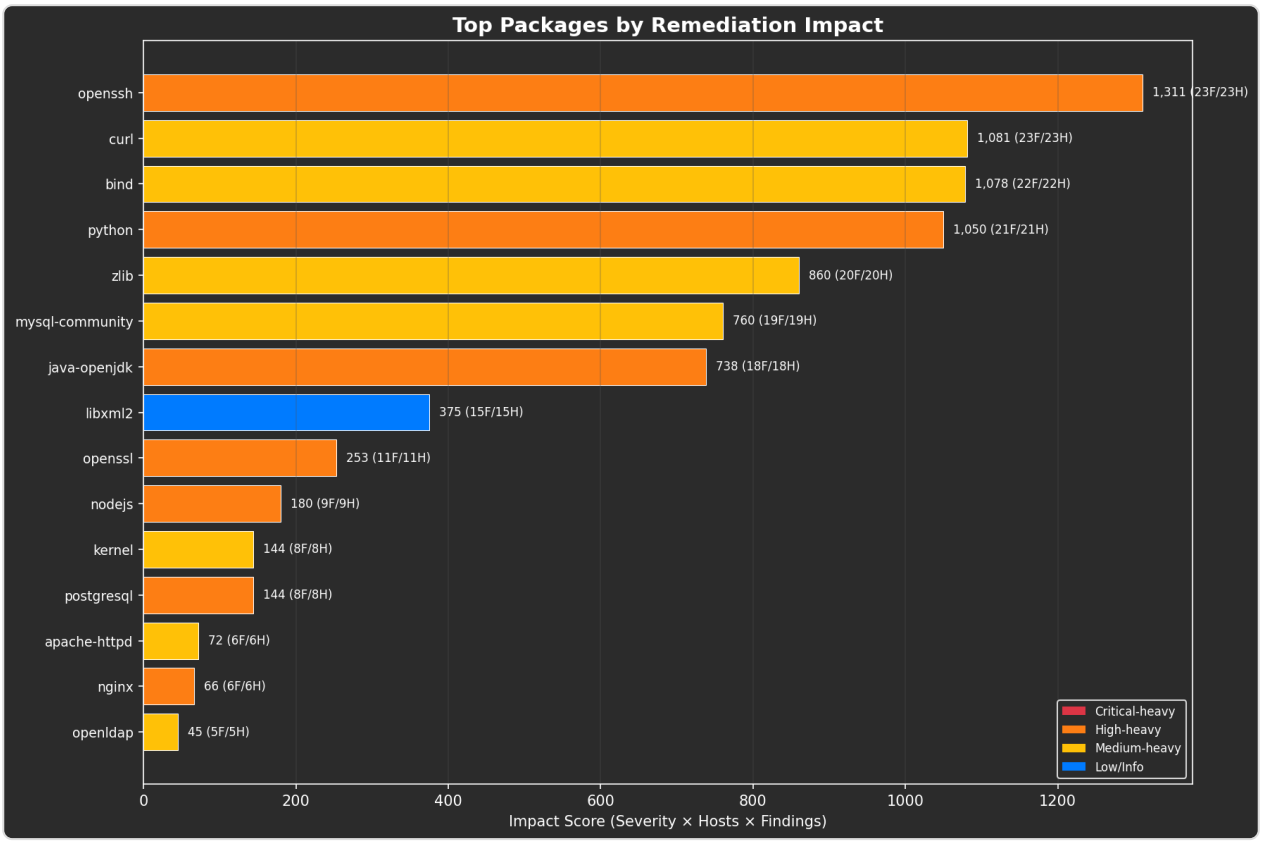
Visualizations



Dashboard Components:

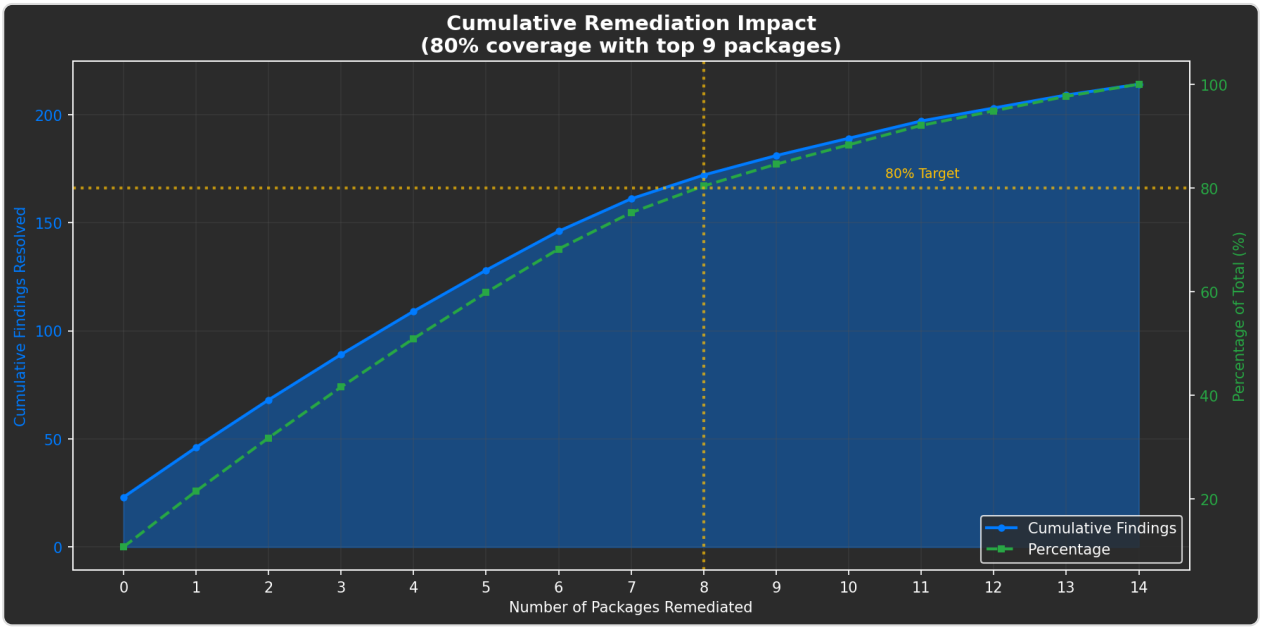
- Key metrics summary
- Top packages by impact
- Cumulative impact curve (80/20 analysis)
- Severity distribution

- Quick wins identification
- Version consolidation opportunities

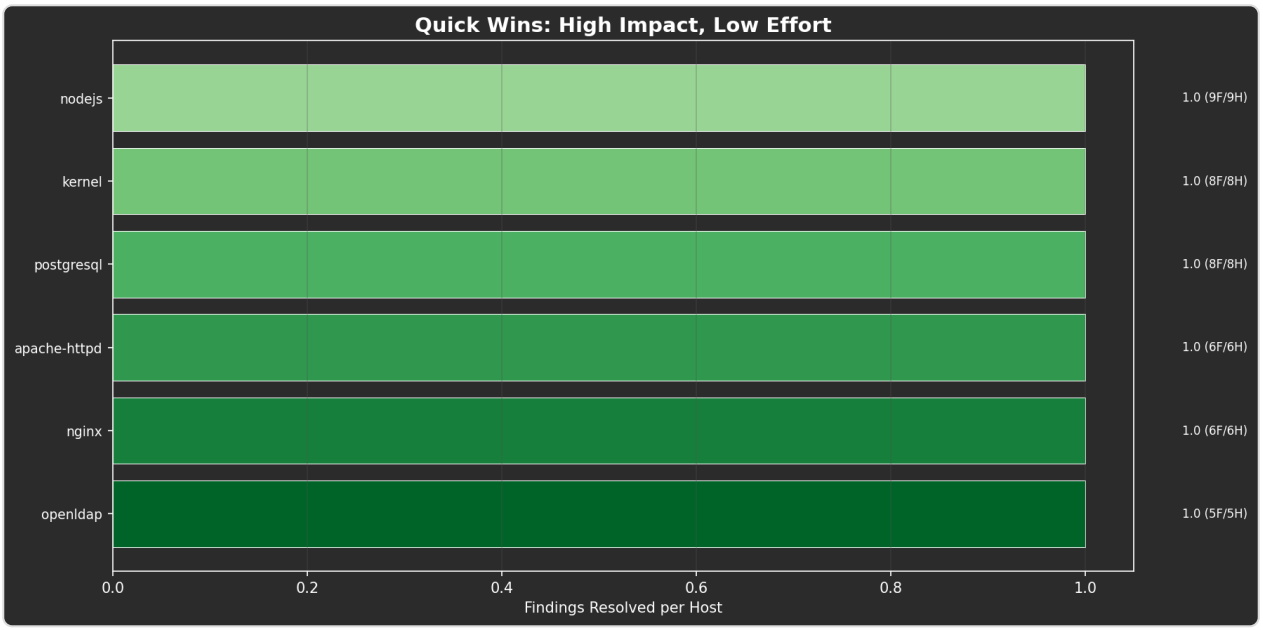


Reading the Chart:

- Bar color indicates severity composition (red = Critical-heavy)
- Bar length shows impact score
- Labels show: `Impact (Findings/Hosts)`



Key Insight: Identifies how many packages needed for 80% remediation coverage.



Quick Win Criteria:

- ≤10 hosts affected (low deployment effort)
- ≥5 findings resolved (significant impact)
- High efficiency ratio (findings per host)

CVE Database Validation

Validate that target versions resolve identified CVEs using NVD.

Validation Process

```
from refactored_app.analysis import (
    CVEValidator,
    create_cve_validation_report
)

# Initialize validator
validator = CVEValidator(
    nvd_api_key='your-api-key'  # Optional, increases rate limit
)

# Validate single package
result = validator.validate_package_versions(
    package_name='openssl',
    target_version='1.1.1w',
    cve_list=['CVE-2023-0286', 'CVE-2022-3602']
)

print(f"Valid: {result.is_valid}")
print(f"Resolved CVEs: {result.cves_resolved}")
print(f"Unresolved: {result.cves_unresolved}")

# Batch validation
packages = [
    {'package_name': 'openssl', 'target_version': '1.1.1w', 'cves': ['CVE-2023-0286']},
    {'package_name': 'curl', 'target_version': '8.5.0', 'cves': ['CVE-2023-38545']}
]
results = validator.batch_validate(packages)

# Generate report
report = create_cve_validation_report(results)
print(report['summary'])
```

NVD API Rate Limits

Access Type	Rate Limit
Without API key	5 requests / 30 seconds
With API key	50 requests / 30 seconds

Get free API key: https://nvd.nist.gov/developers/request-an-api-key

Filtering & Search

Filter Operators

Operator	Description	Example
`EQUALS`	Exact match	`hostname EQUALS "server01"`
`NOT_EQUALS`	Not matching	`severity NOT_EQUALS "Info"`
`CONTAINS`	Substring match	`name CONTAINS "SSL"`
`NOT_CONTAINS`	Excludes substring	`hostname NOT_CONTAINS "test"`
`STARTS_WITH`	Prefix match	`hostname STARTS_WITH "web"`
`ENDS_WITH`	Suffix match	`hostname ENDS_WITH ".com"`
`IN_LIST`	In set of values	`severity IN_LIST ["Critical", "High"]`
`NOT_IN_LIST`	Not in set	`status NOT_IN_LIST ["Resolved"]`
`GREATER_THAN`	Numeric >	`cvss3_base_score GREATER_THAN 7.0`
`LESS_THAN`	Numeric <	`days_open LESS_THAN 30`
`BETWEEN`	Range inclusive	`cvss3_base_score BETWEEN [7.0, 9.0]`
`IS_NULL`	Null/empty check	`cves IS_NULL`
`NOT_NULL`	Has value	`opdir_number NOT_NULL`

Using the Filter Engine

```
from refactored_app.filters import FilterEngine, FilterCriteria, FilterOperator

engine = FilterEngine()

# Add filters
engine.add_filter(FilterCriteria(
    column='severity_text',
    operator=FilterOperator.IN_LIST,
    value=['Critical', 'High']
))

engine.add_filter(FilterCriteria(
    column='status',
    operator=FilterOperator.EQUALS,
    value='Active'
))

engine.add_filter(FilterCriteria(
    column='days_open',
    operator=FilterOperator.GREATER_THAN,
    value=30
))

# Apply filters
filtered_df = engine.apply(lifecycle_df)
```

Environment Classification

Hostnames are automatically classified based on the 9-character format: `LLLLTTCEP`

Position	Meaning	Example
L (1-4)	Location code	`DENV` = Denver
T (5-6)	Tier	`WB` = Web tier
C (7)	Cluster ID	`1` = Cluster 1
E (8)	Environment	`A-Z` = Production, `0-9` = Pre-prod
P (9)	Host type	`p` = Physical, `v` = Virtual

```
from refactored_app.filters import (
    parse_hostname,
    classify_environment_type,
    is_production_host
)

# Parse hostname
result = parse_hostname('DENVWB1AP')
print(f"Location: {result.location}")      # DENV
print(f"Tier: {result.tier}")              # WB
print(f"Environment: {result.environment}") # Production
print(f"Host Type: {result.host_type}")    # Physical

# Check if production
if is_production_host('DENVWB1AP'):
    print("This is a production host")
```

Custom Filter Lists

```
from refactored_app.filters import FilterListManager

manager = FilterListManager()

# Create saved list
manager.create_list('critical_servers', [
    'server001.example.com',
    'server002.example.com',
    'database01.example.com'
])

# Use in filtering
hosts = manager.get_list('critical_servers')
```

Visualizations

Chart Categories

Risk Tab

Chart	Description
CVSS Score Distribution	Histogram of CVSS v3 scores
MTTR by Severity	Average remediation time per severity
Findings by Age	Active findings by age bucket
Top Risky Hosts	Hosts ranked by risk score

Timeline Tab

Chart	Description
Total Findings Over Time	Finding count trend
Severity Over Time	Multi-line severity trend
New vs Resolved	Remediation velocity
Cumulative Risk Score	Total risk progression

SLA Tab

Chart	Description
SLA Compliance by Severity	Compliance percentage per severity
SLA Breach Distribution	Breach counts by category
Days Until Breach	Time remaining to SLA
SLA Trend	Compliance trend over time

OPDIR Tab

Chart	Description
OPDIR Compliance Status	On Track / Due Soon / Overdue
Overdue by OPDIR	Findings per overdue OPDIR
OPDIR Timeline	Due dates on timeline

Efficiency Tab

Chart	Description
Remediation Velocity	New vs Resolved ratio
Scan Coverage	Asset coverage consistency
Reappearance Rate	Vulnerability recurrence
MTTR Trend	Remediation time changes

Plugin Tab

Chart	Description
Top Plugins by Host Count	Most widespread vulnerabilities
Plugin Severity Distribution	Severity breakdown
New Plugins Over Time	Discovery of new vuln types
Plugin Persistence	Long-standing vulnerabilities

Creating Charts Programmatically

```
from refactored_app.visualization import (
    create_severity_pie_chart,
    create_cvss_distribution,
    create_host_risk_bar_chart,
    create_executive_dashboard,
    create_remediation_impact_dashboard
)

# Individual charts
fig = create_severity_pie_chart(findings_df, title="Severity Distribution")
fig.savefig('severity_pie.png', dpi=150)

# Full dashboard
fig = create_executive_dashboard(historical_df, lifecycle_df)
fig.savefig('executive_dashboard.png', dpi=150, bbox_inches='tight')

# Remediation dashboard
fig = create_remediation_impact_dashboard(remediation_plan)
fig.savefig('remediation_dashboard.png', dpi=150)
```

Chart Interaction

Pop-out View:
- Double-click any chart to open in larger window
- Supports zoom, pan, and save

Export Options:
- Right-click → Save as PNG/PDF/SVG
- File → Export Charts → PDF (all charts)

AI Analysis Features

OpenWebUI Integration

Configure AI-powered analysis through OpenWebUI.

Settings → AI Settings:

Setting	Description	Default
Base URL	OpenWebUI instance URL	http://localhost:3000
API Key	Authentication key	-
Model	Analysis model	llama3.1:8b
Temperature	Randomness (0-1)	0.15 (deterministic)
Max Tokens	Response length limit	8000
Analysis Mode	Quick or Comprehensive	Quick

Analysis Types

Type	Description	Use Case
`FULL_ANALYSIS`	Complete vulnerability assessment	Monthly reports
`TIME_TO_REMEDIATE`	MTTR predictions	Sprint planning
`SLA_BREACH_FORECAST`	Predict upcoming breaches	Risk management
`PRIORITIZATION`	AI-recommended order	Daily triage
`TREND_ANALYSIS`	Pattern identification	Strategic planning

Using AI Analysis

```
from refactored_app.ai import VulnerabilityPredictor, AnalysisRequest
from refactored_app.ai import PredictionType, AnalysisMode

predictor = VulnerabilityPredictor(client)

request = AnalysisRequest(
    prediction_type=PredictionType.FULL_ANALYSIS,
    mode=AnalysisMode.COMPREHENSIVE,
    include_recommendations=True
)

result = predictor.analyze(
    findings_df=findings_df,
    lifecycle_df=lifecycle_df,
    request=request
)

print(result.content)

# Follow-up question
followup = predictor.follow_up("What should I prioritize first?")
print(followup.content)
```

Threat Intelligence Integration

Supported Feeds

Feed	Description	Default
CISA KEV	Known Exploited Vulnerabilities	Enabled
EPSS	Exploit Prediction Scoring	Enabled
NVD	National Vulnerability Database	Enabled
DISA IAVM	Defense Information Systems Agency	Disabled
Plugins DB	Local Nessus plugins	Enabled

Configuration

Settings → Threat Intelligence:

```
from refactored_app.ai import ThreatIntelManager

manager = ThreatIntelManager()

# Configure feeds
manager.configure_feed('CISA_KEV', enabled=True)
manager.configure_feed('EPSS', enabled=True)
manager.configure_feed('NVD', enabled=True, api_key='your-key')

# Sync threat intel
manager.sync_all_feeds(mode='incremental')  # Last 30 days

# Or full sync
manager.sync_all_feeds(mode='full')
```

Using Threat Data

```
# Check if CVE is in KEV
is_actively_exploited = manager.check_kev('CVE-2023-44487')

# Get EPSS score
epss_score = manager.get_epss_score('CVE-2023-38545')
print(f"Exploitation probability: {epss_score['probability']}")
print(f"Percentile: {epss_score['percentile']}")
```

Exporting Data

Export Formats

Format	Best For	Function
Excel (`.xlsx`)	Stakeholder reports	`export_to_excel()`
SQLite (`.db`)	Large datasets, queries	`export_to_sqlite()`
JSON (`.json`)	API integration	`export_to_json()`
CSV (`.csv`)	Universal compatibility	DataFrame.to_csv()
PDF	Charts and dashboards	`export_dashboard_to_pdf()`

Excel Export

```
from refactored_app.export import export_to_excel

export_to_excel(
    output_path='vulnerability_report.xlsx',
    historical_df=historical_df,
    lifecycle_df=lifecycle_df,
    host_presence_df=host_presence_df,
    scan_changes_df=scan_changes_df,
    opdir_df=opdir_df
)
```

Included Sheets:
- Summary (key metrics)
- Finding_Lifecycle
- Host_Presence
- Scan_Changes
- Latest_Findings
- Severity_Summary (pivot table)
- OPDIR_Mapping

SQLite Export

```
from refactored_app.export import export_to_sqlite

export_to_sqlite(
    output_path='vulnerability_data.db',
    historical_df=historical_df,
    lifecycle_df=lifecycle_df,
    opdir_df=opdir_df,
    iavm_df=iavm_df
)
```

Created Tables:
- `historical_findings`
- `finding_lifecycle`
- `host_presence`
- `scan_changes`
- `opdir_mapping`
- `iavm_notices`
- `export_metadata`

Indexed Columns:
- hostname, plugin_id, scan_date, ip_address
- status, opdir_number, opdir_status

Remediation Plan Export

```
from refactored_app.analysis import export_remediation_plan

# Excel with multiple sheets
export_remediation_plan(plan, 'remediation.xlsx', format='xlsx')

# CSV (summary only)
export_remediation_plan(plan, 'remediation.csv', format='csv')

# JSON (full details)
export_remediation_plan(plan, 'remediation.json', format='json')
```

Configuration

Configuration File Location

Settings stored in `~/.nessus_tracker/settings.json`

SLA Targets

```
from refactored_app.settings import SettingsManager

settings = SettingsManager()

# Get current SLA targets
sla = settings.get_sla_targets()
# {'Critical': 15, 'High': 30, 'Medium': 60, 'Low': 90}

# Update SLA targets
settings.update_sla_targets({
    'Critical': 7,    # More aggressive
    'High': 14,
    'Medium': 30,
    'Low': 60
})
```

Environment Variables

Variable	Description
`NESSUS_TRACKER_CONFIG`	Config file path
`NVD_API_KEY`	NVD API key
`OPENWEBUI_URL`	OpenWebUI base URL
`OPENWEBUI_API_KEY`	OpenWebUI API key

Severity Configuration

```
# Custom severity colors
settings.update_severity_colors({
    'Critical': '#ff0000',
    'High': '#ff6600',
    'Medium': '#ffcc00',
    'Low': '#0066ff',
    'Info': '#666666'
})

# Custom severity weights (for risk calculation)
SEVERITY_WEIGHTS = {
    'Critical': 4,
    'High': 3,
    'Medium': 2,
    'Low': 1,
    'Info': 0
}
```

Reappearance Detection

```
# Default: 45 days gap = reappearance
REAPPEARANCE_GAP_DAYS = 45

# Customize in config
settings.update_reappearance_gap(60)  # 60 days
```

API Reference

Quick Reference

```
# Data Loading
from refactored_app.core import parse_multiple_nessus_files, load_plugins_database

# Lifecycle Analysis
from refactored_app.analysis import (
    analyze_finding_lifecycle,
    identify_reappearances,
    calculate_mttr,
    get_findings_by_age
)

# Host Analysis
from refactored_app.analysis import (
    create_host_presence_analysis,
    identify_missing_hosts,
    calculate_scan_coverage
)

# Advanced Metrics
from refactored_app.analysis import (
    get_all_advanced_metrics,
    calculate_reopen_rate,
    calculate_sla_breach_tracking
)

# Compliance
from refactored_app.analysis import (
    load_opdir_mapping,
    enrich_with_opdir,
    load_iavm_summaries
)

# Package Impact
from refactored_app.analysis import (
    analyze_package_version_impact,
    create_remediation_summary_df,
    export_remediation_plan
)

# CVE Validation
from refactored_app.analysis import (
    CVEValidator,
    create_cve_validation_report
)

# Filtering
from refactored_app.filters import (
    FilterEngine,
    FilterCriteria,
    FilterOperator
)
```

```
# Visualization
from refactored_app.visualization import (
    create_executive_dashboard,
    create_remediation_impact_dashboard,
    create_package_impact_bar_chart
)

# Export
from refactored_app.export import (
    export_to_excel,
    export_to_sqlite,
    export_to_json
)

# AI
from refactored_app.ai import (
    VulnerabilityPredictor,
    ThreatIntelManager,
    OpenWebUIClient
)
```

Troubleshooting

Common Issues

"No data available" in charts

1. Verify data is loaded (File menu)
2. Check filter settings (may be too restrictive)
3. Verify date range includes scan dates

Slow performance with large datasets

1. Export to SQLite for datasets > 100,000 findings
2. Use date range filtering
3. Enable pagination in lifecycle view

CVE validation timeout

1. NVD API rate limiting - wait 30 seconds
2. Add NVD API key for higher limits

3. Reduce number of CVEs per package

Import errors

```
# Install missing dependencies
pip install pandas matplotlib openpyxl numpy requests

# Check Python version (3.8+ required)
python --version
```

Memory issues

1. Process scans in smaller batches
2. Use SQLite export for persistence
3. Filter to specific date ranges before analysis

Getting Help

- **Documentation:** `/docs/VISUALIZATION_GUIDE.md`
- **Issues:** https://github.com/acasehs/PluginHistory/issues
- **Logs:** `~/.nessus_tracker/logs/`

Keyboard Shortcuts

Shortcut	Action
`Ctrl+O`	Open file
`Ctrl+S`	Save/Export
`Ctrl+F`	Focus search
`Ctrl+R`	Refresh data
`F5`	Reload charts
`Escape`	Close modal

Last updated: December 2024
Version: 2.0