Plugin History Analysis Tool

Complete Documentation

Version 2.0 - December 2024

Includes:
- User Guide
- Visualization Guide
- Data Science Analysis

Plugin History Analysis Tool - Complete User Guide

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')
```

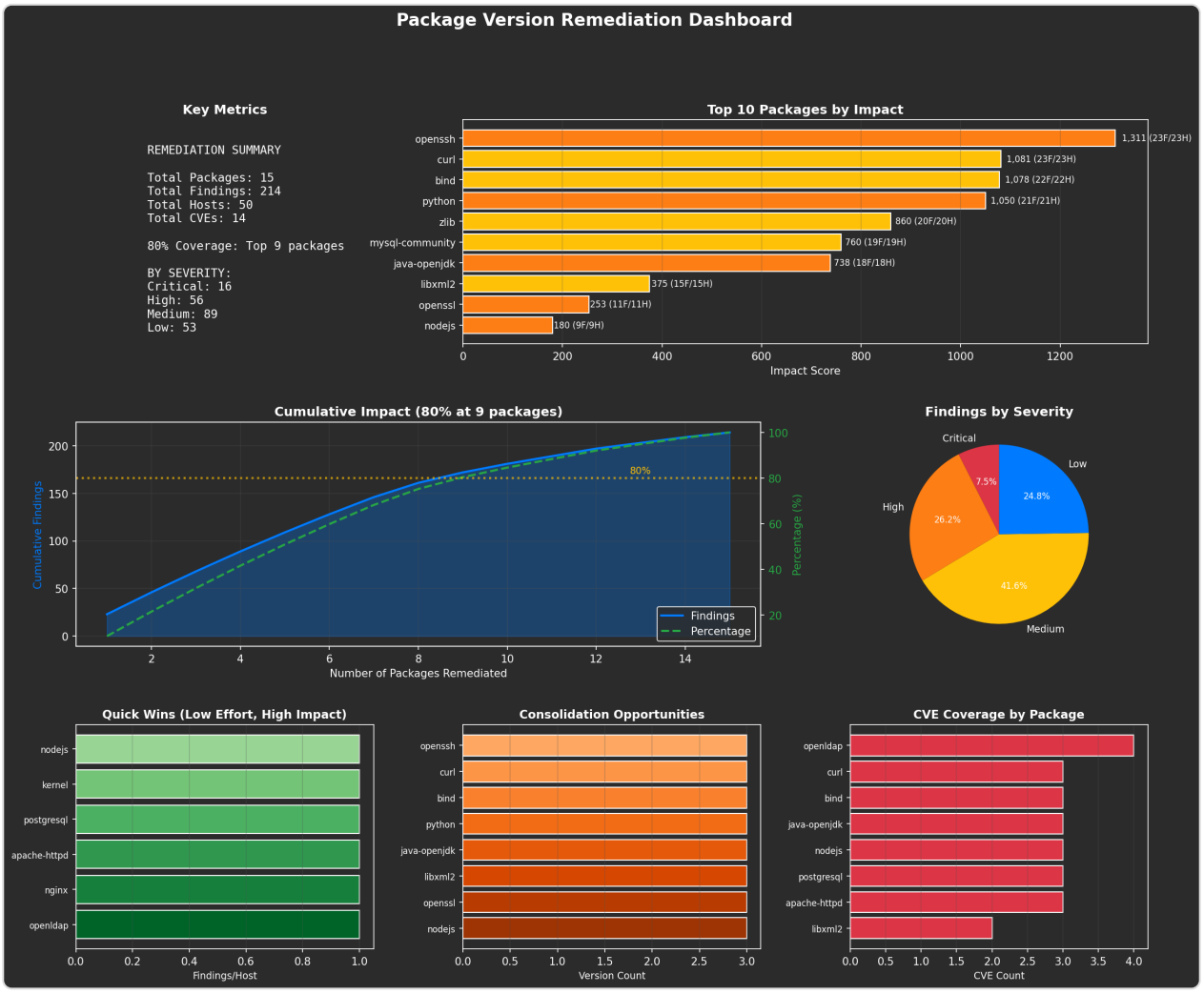
Impact Calculation Formula

```
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
```

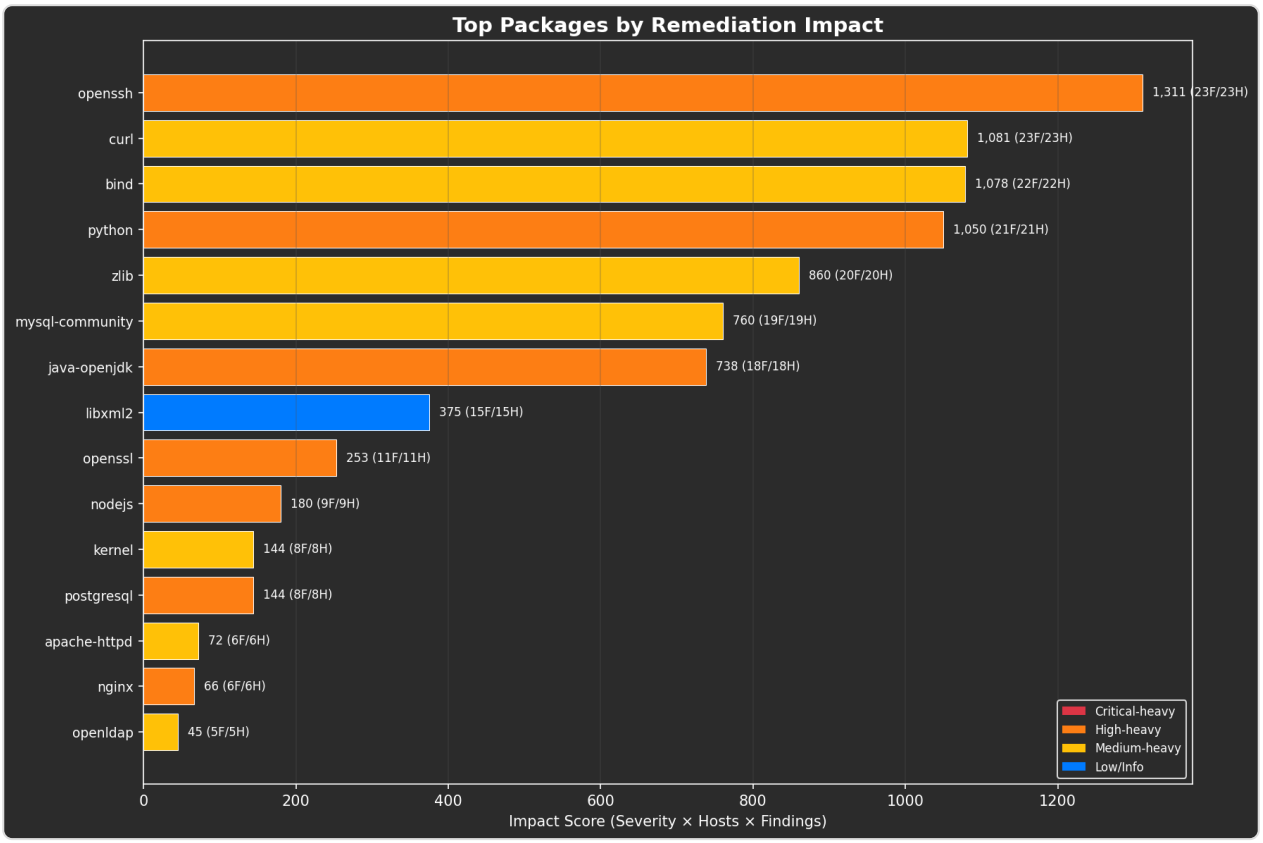
Visualizations



Dashboard Components:

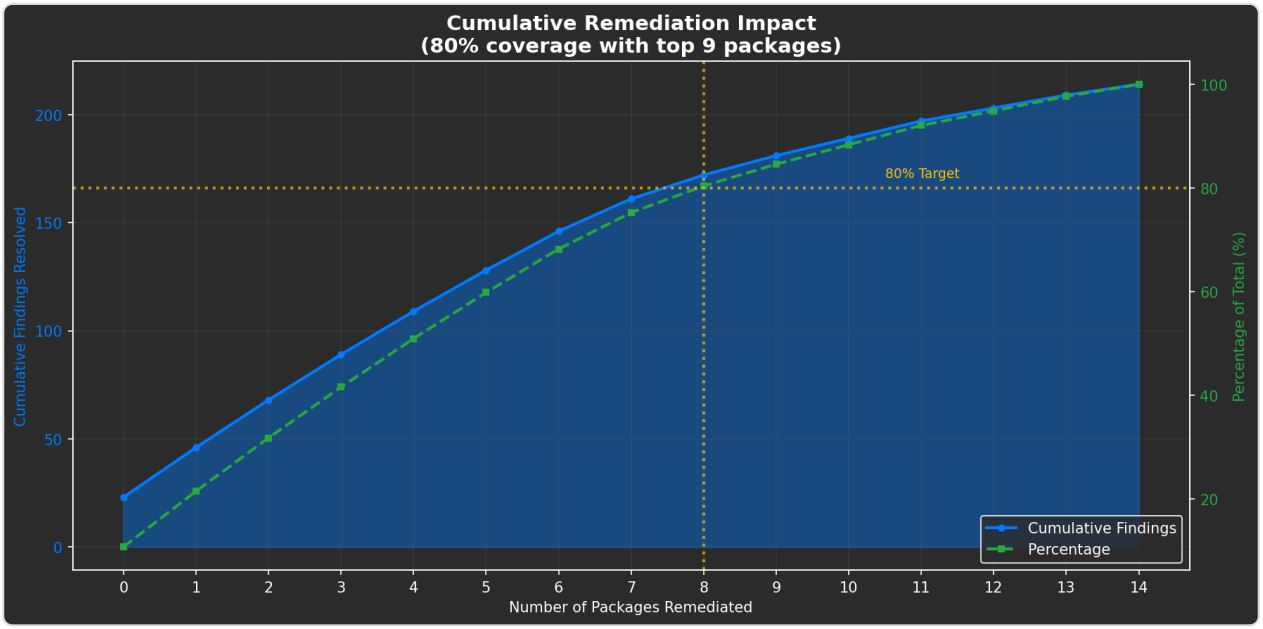
- Key metrics summary
- Top packages by impact
- Cumulative impact curve (80/20 analysis)
- Severity distribution

- Quick wins identification
- Version consolidation opportunities

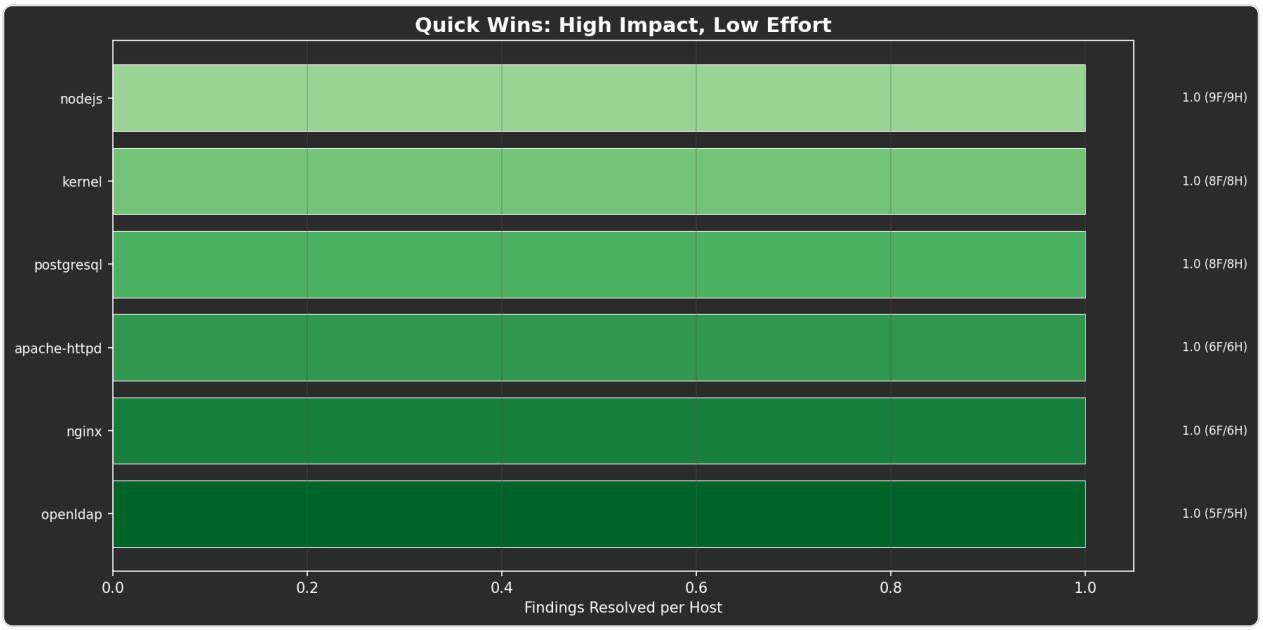


Reading the Chart:

- 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

Vulnerability Analysis Visualization Guide

Vulnerability Analysis Visualization Guide

This guide documents all visualizations available in the Plugin History Analysis Tool, explaining their purpose, data inputs, interpretation methods, and cybersecurity value.

Table of Contents

Risk Tab

CVSS Score Distribution

What it shows: Histogram displaying the distribution of CVSS v3 base scores across all findings. Bars are color-coded by severity:
- **Critical** (red): 9.0+
- **High** (orange): 7.0-8.9
- **Medium** (yellow): 4.0-6.9
- **Low** (green): 0-3.9

Data Inputs:
- `cvss3_base_score` from vulnerability scan data

Cybersecurity Value:
Helps identify the overall risk profile of your environment. A distribution skewed toward higher scores indicates systemic security issues requiring immediate attention. Use this to prioritize remediation efforts and justify security investments to leadership.

How to Interpret:
- Peak location shows typical vulnerability severity in your environment
- Wide spread suggests diverse risk levels requiring varied response strategies
- Right-skewed distributions (more high scores) are concerning and warrant investigation
- Compare against industry benchmarks for your sector

Available Filters: Severity, Status, Date Range, Environment

Mean Time to Remediation (MTTR)

What it shows: Bar chart showing average days to resolve vulnerabilities grouped by severity level. Only includes resolved findings.

Data Inputs:
- `days_open` - Calculated as: current_date - first_observed_date (from scanner output)
- `severity_text` - Severity classification
- `status` - Must be "Resolved"

Cybersecurity Value:
Key metric for security operations effectiveness. Compare against SLA targets to identify compliance gaps. Critical and High MTTR exceeding SLAs indicates remediation process failures that may require resource reallocation or process improvement.

How to Interpret:
- Lower bars = faster remediation (better)
- Critical should have lowest MTTR (highest priority)
- Compare to your SLA targets:
- Critical: 15 days
- High: 30 days
- Medium: 60 days
- Low: 90 days

Available Filters: Severity, Date Range, Environment

Findings by Age

What it shows: Stacked bar chart showing active findings bucketed by age:
- 0-30 days (newest)
- 31-60 days
- 61-90 days
- 90+ days (aging)

Data Inputs:
- `days_open` - Calculated as: current_date - first_observed_date (from scanner output)
- `status` - Active findings only

Cybersecurity Value:
Aging vulnerabilities represent increased risk exposure. Findings over 90 days likely indicate process failures, lack of ownership, or technical barriers. Use for compliance reporting and risk acceptance decisions with leadership.

How to Interpret:
- Most findings should be in 0-30 day bucket
- Large 90+ day bucket indicates remediation backlog requiring management attention
- Track this over time to show remediation velocity improvement

Available Filters: Severity, Status, Environment

Top Risky Hosts

What it shows: Horizontal bar chart showing hosts with highest cumulative risk scores, colored by environment type:
- **Production** (green)
- **PSS/Pre-Production** (blue)
- **Shared** (yellow)

Data Inputs:
- `hostname` - Asset identifier
- `severity_value` - Numeric severity weight (Critical=4, High=3, Medium=2, Low=1)
- `environment_type` - Derived from hostname pattern

Cybersecurity Value:
Identifies assets requiring immediate security attention. Production hosts with high risk scores should be prioritized. Helps target penetration testing and security assessments. Use for asset-centric remediation planning.

How to Interpret:
- Focus remediation on top hosts first
- Consider isolating high-risk production systems
- Environment coloring helps prioritize based on business impact
- A single host with extreme risk may indicate compromise or critical misconfiguration

Available Filters: Severity, Status, Environment, Host Type

Timeline Tab

Total Findings by Period

What it shows: Line chart showing total vulnerability count over time, grouped by selected interval (daily, weekly, monthly). Includes trend indicator.

Data Inputs:
- `scan_date` - When vulnerability was detected
- Finding counts aggregated by period

Cybersecurity Value:
Shows overall security posture trajectory. Increasing trend indicates growing attack surface or inadequate remediation. Flat or decreasing trend suggests security program effectiveness.

How to Interpret:
- Upward trend = increasing risk exposure
- Spikes may indicate:
- New vulnerability scanner deployment
- Expanded scan scope
- Major vulnerability disclosures (e.g., Log4j)
- Compare to security events or infrastructure changes

Available Filters: Date Range, Severity, Environment

Findings by Severity Over Time

What it shows: Multi-line chart tracking Critical, High, Medium, Low findings over time. Each severity has distinct color for easy tracking.

Data Inputs:
- `scan_date` - Detection timestamp
- `severity_text` - Severity classification

Cybersecurity Value:
Monitors severity distribution changes. Sudden Critical/High spikes may indicate zero-days or new attack vectors. Helps demonstrate remediation progress by severity tier to stakeholders.

How to Interpret:
- Critical/High lines should trend downward over time
- Watch for correlation between severity levels
- Diverging trends may indicate prioritization issues (e.g., only remediating low severity)
- Sudden spikes warrant immediate investigation

Available Filters: Date Range, Environment

New vs Resolved

What it shows: Grouped bar chart comparing:
- **New findings** discovered per period (red)
- **Findings resolved** per period (green)
- Net change shown

Data Inputs:
- `scan_changes` table - New/Resolved status transitions

Cybersecurity Value:
Core metric for security program health. Resolved > New indicates reducing risk. Persistent deficit suggests inadequate resources or process issues requiring management escalation.

How to Interpret:
- Green bars (resolved) should exceed red bars (new)
- Calculate velocity: Resolved/New ratio
- Ratio > 1.0 = improving posture
- Ratio < 1.0 = falling behind
- Ratio = 1.0 = treading water

Available Filters: Date Range, Severity, Environment

Cumulative Risk Score

What it shows: Area chart showing total severity-weighted risk score over time.

Calculation:

```
Risk Score = Σ (severity_value for all active findings)
where:
  Critical = 4 points
  High = 3 points
  Medium = 2 points
  Low = 1 point
```

Cybersecurity Value:
Single metric capturing overall organizational risk. Use for executive reporting and risk trending. Enables comparison across time periods and business units.

How to Interpret:
- Downward slope = risk reduction
- Plateaus indicate stagnation
- Sharp increases require immediate investigation
- Set target risk score thresholds for alerts

Available Filters: Date Range, Environment

SLA Tab

SLA Compliance Overview

What it shows: Stacked bar chart showing:
- **Compliant** (green) - On track to meet SLA
- **At-Risk** (yellow) - Approaching deadline (within 25%)
- **Breached** (red) - Past SLA deadline

Grouped by severity level.

Data Inputs:
- `severity_text` - Severity classification
- `sla_status` - Calculated from days_open vs SLA targets

SLA Targets (Configurable):
Severity	Days
Critical	15
High	30
Medium	60
Low	90

Cybersecurity Value:
Critical for compliance reporting and audit evidence. SLA breaches may trigger contractual penalties or regulatory findings. Track for continuous improvement initiatives.

How to Interpret:
- Focus on reducing red segments, especially for Critical/High
- Yellow segments need proactive attention before breach
- 100% green is the goal

Available Filters: Severity, Environment

SLA Breaches by Severity

What it shows: Bar chart showing count of SLA-breached findings per severity level.

Data Inputs:
- `severity_text` - Severity classification
- `days_open` - Current age of finding
- `sla_targets` - Configured thresholds

Cybersecurity Value:
Direct compliance risk indicator. High breach counts require escalation and resource allocation. Document for audit trail and management reporting.

How to Interpret:
- Critical breaches are highest priority
- Zero breaches is the goal
- Track month-over-month improvement
- Investigate root causes for persistent breaches

Available Filters: Date Range, Environment

SLA Approaching Deadline

What it shows: List/bar of findings within warning threshold of SLA deadline (typically within 25% of remaining time).

Data Inputs:
- `severity_text` - Severity classification
- `days_open` - Current age
- `sla_targets` - Configured thresholds
- `warning_threshold` - Typically 25%

Cybersecurity Value:
Early warning system for potential breaches. Enables proactive remediation before SLA violation. Helps resource planning and workload distribution.

How to Interpret:
- These findings need immediate attention
- Sort by days remaining
- Assign owners and track daily
- Use for daily standup prioritization

Available Filters: Severity, Environment

Days to SLA Deadline

What it shows: Distribution chart showing days remaining until SLA deadline.
- Positive values = time remaining
- Negative values = days overdue

Data Inputs:
- `sla_deadline` - Calculated from first_seen + SLA days
- `current_date` - Today

Cybersecurity Value:
Visualizes remediation urgency across entire portfolio. Helps identify systemic issues (e.g., all Critical findings overdue).

How to Interpret:
- Distribution should be right-skewed (most findings have time remaining)
- Left tail (negative values) represents breaches
- Bimodal distribution may indicate batch remediation patterns

Available Filters: Severity, Status, Environment

OPDIR Tab

OPDIR Coverage

What it shows: Pie chart showing:
- Findings mapped to OPDIR directives
- Unmapped findings

Data Inputs:
- `opdir_number` - OPDIR directive reference (presence indicates mapping)

Cybersecurity Value:
OPDIR directives are authoritative remediation requirements. Unmapped findings may lack official guidance. Coverage indicates compliance posture with mandated security requirements.

How to Interpret:
- Higher mapped percentage = better compliance coverage
- Unmapped findings need manual assessment
- Low coverage may indicate new vulnerability types not yet in OPDIR guidance

Available Filters: Severity, Status

OPDIR Compliance Status

What it shows: Donut chart showing:
- **Overdue** (red) - Past OPDIR deadline
- **Due Soon** (yellow) - Approaching deadline
- **On Track** (green) - Meeting timeline

Data Inputs:
- `opdir_due_date` - Mandated remediation deadline
- `current_date` - Today

Cybersecurity Value:
Direct compliance measurement against authoritative directives. Overdue findings may result in audit findings or security incidents. Critical for regulatory compliance.

How to Interpret:
- Minimize red (overdue) segment
- Yellow indicates upcoming deadlines needing attention
- Green shows compliant items
- Track weekly improvement

Available Filters: Date Range, Environment

OPDIR Finding Age Distribution

What it shows: Histogram of days since discovery for OPDIR-mapped findings.

Data Inputs:
- `first_seen` - Discovery date
- `opdir_number` - Must have OPDIR mapping

Cybersecurity Value:
Shows remediation velocity for mandated vulnerabilities. Long-standing OPDIR findings indicate serious compliance issues requiring escalation.

How to Interpret:
- Distribution should skew left (newer findings)
- Long tail indicates remediation challenges
- Investigate outliers for root cause

Available Filters: OPDIR Status, Severity

Findings by OPDIR Year

What it shows: Grouped bar showing findings by OPDIR directive release year.

Data Inputs:
- `opdir_number` - Year extracted from directive number

Cybersecurity Value:
Older directives with open findings suggest persistent compliance gaps. Helps identify historical remediation debt requiring special attention.

How to Interpret:
- Findings from older years indicate long-standing issues
- Recent years should have fewer findings (newer directives)
- Large counts for old years = technical debt

Available Filters: Status, Severity

Efficiency Tab

Scan Coverage Consistency

What it shows: Distribution of hosts by number of scans they appear in.

Data Inputs:
- `hostname` - Asset identifier
- `scan_date` - Unique scans per host count

Cybersecurity Value:
Identifies gaps in vulnerability scanning program. Hosts scanned infrequently may harbor undetected vulnerabilities creating blind spots.

How to Interpret:
- Peak should be at high scan counts (consistent coverage)
- Left tail indicates under-scanned assets
- Investigate hosts with low scan counts

Available Filters: Date Range, Host Type

Vulnerability Reappearance

What it shows: Chart showing vulnerabilities that were resolved but reappeared in subsequent scans.

Data Inputs:
- `scan_changes` - Status transitions tracking Resolved → New

Cybersecurity Value:
Indicates ineffective remediation or regression. High reappearance suggests root cause not addressed or change management issues requiring process improvement.

How to Interpret:
- Lower is better
- Recurring findings need root cause analysis
- May indicate:
- Patch rollback
- Configuration drift
- Incomplete remediation
- Re-imaging with old images

Available Filters: Severity, Date Range

Vulnerabilities per Host

What it shows: Distribution showing how vulnerabilities are spread across hosts.

Data Inputs:
- `hostname` - Asset identifier
- Finding count per host

Cybersecurity Value:
Identifies concentration risk. Few hosts with many vulnerabilities are high-value targets for attackers. May indicate compromised or misconfigured systems.

How to Interpret:
- Right-skewed = few problematic hosts (concentrate efforts)
- Flat distribution = systemic issues (need broad remediation)
- Identify outliers for investigation

Available Filters: Severity, Environment, Host Type

Resolution Velocity

What it shows: Distribution of time-to-resolution for remediated vulnerabilities.

Data Inputs:
- `days_open` - For resolved findings only

Cybersecurity Value:
Measures remediation efficiency. Compare to industry benchmarks and SLA targets. Use for process improvement and resource planning.

How to Interpret:
- Peak location shows typical remediation time
- Long tail indicates outliers needing investigation
- Track shift leftward over time (faster remediation)

Available Filters: Severity, Date Range, Environment

Network Tab

Top Subnets by Vulnerability

What it shows: Horizontal bar chart showing network subnets with most vulnerabilities.

Data Inputs:
- `ip_address` - Subnet extracted (first 3 octets)

Cybersecurity Value:
Identifies network segments requiring security focus. May indicate:
- Vulnerable applications
- Outdated infrastructure
- Inadequate segmentation
- Shadow IT

How to Interpret:
- Focus network security efforts on top subnets
- Consider additional segmentation for high-risk segments
- Investigate common vulnerability patterns

Available Filters: Severity, Status, Environment

Subnet Risk Scores

What it shows: Risk-weighted view of network segments using severity scoring.

Data Inputs:
- `ip_address` - Subnet identifier
- `severity_value` - Weighted score

Cybersecurity Value:
Prioritizes network segments by actual risk, not just count. Critical vulnerabilities weight higher than informational. Better for risk-based prioritization.

How to Interpret:
- High-risk subnets need immediate attention regardless of count
- May justify network redesign or additional controls
- Compare risk density (risk per host)

Available Filters: Severity, Status

Host Criticality Distribution

What it shows: Distribution of cumulative risk scores across hosts.

Data Inputs:
- `hostname` - Asset identifier
- `severity_value` - Summed per host

Cybersecurity Value:
Visualizes risk concentration across infrastructure. Tail represents high-value targets requiring immediate attention or isolation.

How to Interpret:
- Right tail hosts are critical priority
- Average line shows typical risk level
- Investigate hosts above 2 standard deviations

Available Filters: Environment, Host Type

Environment Distribution

What it shows: Pie/bar chart showing findings by environment type:
- Production
- PSS (Pre-Production)
- Shared
- Unknown

Data Inputs:
- `hostname` → `environment_type` mapping

Cybersecurity Value:
Production vulnerabilities have highest business impact. Shared infrastructure affects multiple environments, creating broader risk exposure.

How to Interpret:
- Production findings need prioritization
- Shared findings may have broader impact
- PSS can be used for patch testing

Available Filters: Severity, Status

Plugin Tab

Top 15 Most Common Plugins

What it shows: Horizontal bar chart showing most frequently detected vulnerability types (by Plugin ID).

Data Inputs:
- `plugin_id` - Unique vulnerability identifier
- `plugin_name` - Human-readable name
- Count of occurrences

Cybersecurity Value:
Identifies systemic vulnerabilities affecting many hosts. These are often:
- Misconfigurations
- Missing patches
- Default credentials
- Outdated software

Single remediation action can affect many hosts (high ROI).

How to Interpret:
- Top plugins may have single remediation action
- High count + high severity = critical priority
- Look for patterns (same application, same OS)

Available Filters: Severity, Status, Environment

Findings by Severity

What it shows: Bar chart showing total findings per severity level.

Data Inputs:
- `severity_text` - Severity classification

Cybersecurity Value:
Quick view of severity distribution. Critical and High counts drive risk posture. Use for executive dashboards.

How to Interpret:
- Healthy: Pyramid shape (more Low, fewer Critical)
- Concerning: Inverted pyramid
- Track ratios over time

Available Filters: Status, Environment

Plugins Affecting Most Hosts

What it shows: Plugins ranked by number of unique hosts affected.

Data Inputs:
- `plugin_id` - Vulnerability identifier
- `hostname` - Unique count per plugin

Cybersecurity Value:
Wide-spread vulnerabilities indicate systemic issues. High host count + high severity = critical priority requiring immediate action.

How to Interpret:
- Top plugins affect most infrastructure
- Single fix can reduce risk across many assets
- Prioritize by (host_count × severity_weight)

Available Filters: Severity, Status

Plugins with Longest Average Age

What it shows: Plugins ranked by average days open.

Color Coding:
- **Red**: >90 days average
- **Orange**: >30 days average
- **Green**: <30 days average

Data Inputs:
- `plugin_id` - Vulnerability identifier
- `days_open` - Averaged per plugin

Cybersecurity Value:
Long-standing vulnerability types may indicate:
- Remediation barriers (no patch available)
- False positives needing tuning
- Process failures
- Resource constraints

How to Interpret:
- Red items need investigation
- May be unfixable or require significant effort
- Consider risk acceptance for very old items

Available Filters: Status, Environment

Priority Tab

Remediation Priority Matrix

What it shows: Scatter plot with:
- **X-axis**: CVSS score (severity)
- **Y-axis**: Days open (age)
- **Point color**: Severity level

Data Inputs:
- `cvss3_base_score` - Numeric severity
- `days_open` - Age of finding
- `severity_text` - Color coding

Cybersecurity Value:
Visual prioritization tool combining severity and urgency.

Quadrant Analysis:
Quadrant	CVSS	Age	Priority
Upper-Right	High	Old	CRITICAL - Fix immediately
Lower-Right	High	New	HIGH - Fix soon

| Upper-Left | Low | Old | MEDIUM - Plan remediation |
| Lower-Left | Low | New | LOW - Schedule later |

How to Interpret:
- Focus on upper-right quadrant first
- Track movement toward lower-left over time

Available Filters: Severity, Status, Environment

Priority Distribution

What it shows: Pie chart showing findings by calculated priority bucket:
- **Urgent** - High severity + old
- **High** - High severity OR old
- **Medium** - Moderate risk
- **Low** - Low severity + new

Data Inputs:
- `priority_score` - Calculated from CVSS + age

Cybersecurity Value:
Summary view for resource planning. Urgent items need immediate attention and dedicated resources.

How to Interpret:
- Track urgent reduction over time
- Healthy distribution has small urgent slice (<10%)
- Large urgent slice requires escalation

Available Filters: Severity, Status

Top 10 Priority Findings

What it shows: List of highest priority findings based on CVSS score and age combination.

Data Inputs:
- `priority_score` - Ranking metric
- `plugin_name` - Vulnerability description
- `hostname` - Affected asset

Cybersecurity Value:
Action list for remediation teams. These should be assigned and tracked daily in standups.

How to Interpret:
- Start remediation from top
- Check for common threads:
- Same host (concentrate remediation)
- Same vulnerability (single fix, multiple hosts)
- Update daily

Available Filters: Environment, Host Type

Priority Score by Severity

What it shows: Average priority score grouped by severity level.

Data Inputs:
- `priority_score` - Calculated metric
- `severity_text` - Grouping

Cybersecurity Value:
Shows if high-severity items are being addressed quickly.

How to Interpret:
- Critical should have LOWEST priority score (newest = being fixed fast)
- Higher bars for Critical/High = aging high-severity items = BAD
- Investigate if Critical bar > Low bar

Available Filters: Status, Environment

Host Tracking Tab

Missing Hosts

What it shows: Hosts not seen in recent scans that previously appeared.

Data Inputs:
- `hostname` - Asset identifier
- `last_seen_date` - Most recent scan appearance

Cybersecurity Value:
Missing hosts may be:
- Decommissioned (verify with CMDB)
- Renamed (update records)
- Dropped from scan scope (configuration error)
- Network isolated (may need agent-based scanning)

Security risk if active but unscanned!

How to Interpret:
- Verify status of each missing host
- Update inventory or scan configuration
- Document decommissioned hosts

Available Filters: Date Range, Environment

Hosts per Scan Over Time

What it shows: Line chart showing unique host count per scan over time.

Data Inputs:
- `scan_date` - Scan timestamp
- `hostname` - Unique count per scan

Cybersecurity Value:
Monitors scan scope consistency. Changes may indicate:
- Infrastructure changes
- Scanner issues
- Network problems
- Credential failures

How to Interpret:
- Stable line is good
- Sudden drops need investigation
- Gradual increase = growing infrastructure

Available Filters: Date Range

Declining Scan Coverage

What it shows: Hosts showing decreased scan frequency or intermittent coverage.

Data Inputs:
- `hostname` - Asset identifier
- Scan appearance frequency calculation

Cybersecurity Value:
Intermittent scanning creates blind spots. Attackers can exploit gaps in visibility. Critical hosts should have consistent coverage.

How to Interpret:
- Investigate cause for each declining host
- May need scanner configuration changes
- Consider agent-based scanning for mobile assets

Available Filters: Date Range, Environment

Host Status Overview

What it shows: Distribution of hosts by scanning status:
- **Active** - Seen in recent scans
- **Intermittent** - Inconsistent appearance
- **Missing** - Not seen recently

Data Inputs:
- `hostname` - Asset identifier
- Scan frequency classification

Cybersecurity Value:
Quick health check of vulnerability management program coverage. High missing percentage indicates scanning program issues.

How to Interpret:
- Maximize Active percentage
- Minimize Missing percentage
- Set thresholds based on scan schedule

Available Filters: Environment, Host Type

Metrics Tab

Remediation vs Active by Severity

What it shows: Grouped bar comparing:
- **Resolved** (green) findings per severity
- **Active** (red) findings per severity

Data Inputs:
- `severity_text` - Severity classification
- `status` - Active or Resolved

Cybersecurity Value:
Shows remediation progress across severity tiers. Higher resolved:active ratio is better. Use for monthly reporting.

How to Interpret:
- Green should exceed red, especially for Critical/High
- Calculate ratio per severity
- Track improvement over time

Available Filters: Date Range, Environment

Organization Risk Trend

What it shows: Line chart showing overall risk score over time with trend line.

Data Inputs:
- `scan_date` - Timestamp
- `severity_value` - Summed risk score

Cybersecurity Value:
Executive-level metric for security program effectiveness. Use for board reporting and budget justification.

How to Interpret:
- Downward trend shows improvement
- Flat indicates stagnation
- Rising requires action and escalation

Available Filters: Date Range

SLA Compliance by Severity

What it shows: Stacked percentage bar showing SLA compliance rate per severity.

Data Inputs:
- `severity_text` - Severity classification
- `sla_status` - Compliance status

Cybersecurity Value:
Compliance metric for regulatory and contractual requirements. Target 100% compliance, especially for Critical.

How to Interpret:
- Track improvement over time
- Identify severity levels with compliance issues
- Set improvement targets

Available Filters: Date Range, Environment

Vulnerabilities per Host Trend

What it shows: Average vulnerabilities per host over time.

Calculation:

```
Vulns per Host = Total Findings / Unique Hosts
```

Cybersecurity Value:
Normalized metric accounting for infrastructure growth. Better for comparison across time periods and organizations.

How to Interpret:
- Decreasing trend shows per-asset risk reduction
- Compare to industry benchmarks
- Use for maturity assessment

Available Filters: Date Range, Environment

Advanced Charts

Vulnerability Density Heatmap

What it shows: Grid showing vulnerability density by host and severity. Darker cells = more findings.

Data Inputs:
- `hostname` - Row identifier
- `severity_text` - Column identifier
- Count - Cell value

Cybersecurity Value:
Visual pattern recognition for concentrated risk areas. Quickly identify problematic hosts.

How to Interpret:
- Dark cells are priority
- Patterns across rows = systemic host issues
- Patterns down columns = severity-specific issues

Available Filters: Severity, Status, Environment

Bubble Chart

What it shows: Multi-dimensional visualization:
- **X-axis**: CVSS score
- **Y-axis**: Age (days open)
- **Bubble size**: Hosts affected
- **Color**: Severity

Cybersecurity Value:
Rich visualization for executive presentations. Shows multiple risk dimensions in single view.

How to Interpret:
- Large red bubbles in upper-right = critical priorities
- Focus on reducing bubble count and size
- Track movement toward origin (lower-left)

Available Filters: Severity, Status

Lifecycle Flow (Sankey)

What it shows: Flow diagram showing vulnerability progression:
- Discovery → Active
- Active → Resolved
- Resolved → Reappeared

Data Inputs:
- Status transitions over time

Cybersecurity Value:
Process visualization for remediation workflow analysis. Identifies bottlenecks and inefficiencies.

How to Interpret:
- Thick flows to Resolved = good
- Thin flows to Resolved = bottleneck
- Flows to Reappeared = remediation quality issue

Available Filters: Date Range, Severity

Category Treemap

What it shows: Hierarchical view of vulnerabilities by plugin family/category.

Data Inputs:
- `plugin_family` - Category grouping
- Count per category
- Severity weighting

Cybersecurity Value:
Identifies vulnerability categories requiring attention. Helps focus remediation by vulnerability type.

How to Interpret:
- Large tiles represent significant categories
- Color indicates severity mix
- Click to drill down

Available Filters: Severity, Status

SLA Breach Prediction

What it shows: Forecast of upcoming SLA breaches based on current trajectory.

Data Inputs:
- `days_to_sla` - Time remaining
- `remediation_velocity` - Historical fix rate

Cybersecurity Value:
Proactive risk management. Plan resources before breaches occur. Enables preventive action.

How to Interpret:
- Rising line = increasing future breaches
- Take action before predicted breach
- Use for resource planning

Available Filters: Severity

Period Comparison

What it shows: Side-by-side comparison of metrics between two time periods.

Metrics Compared:
- Total findings
- Risk score
- MTTR
- SLA compliance
- New vs Resolved ratio

Cybersecurity Value:
Demonstrates program improvement for reporting and audits. Shows trend direction.

How to Interpret:
- Green indicators = improvement
- Red indicators = regression
- Use for quarterly/annual reporting

Available Filters: Date Range Selection

Smart Filtering

Overview

Smart filtering automatically overrides the UI status filter for specific visualizations that require certain data regardless of user selection. This ensures accurate metrics even when the user has filtered to view only Active or Remediated findings.

Why Smart Filtering Matters

Some metrics can only be calculated with specific data:
- **Remediation Rate**: Requires BOTH Active and Remediated findings to calculate the percentage
- **MTTR (Mean Time to Remediation)**: Requires only Remediated findings (can't measure time to fix if not fixed)
- **Reopen Rate**: Requires both statuses to track findings that were remediated then reappeared

Without smart filtering, viewing only "Active" findings would show 0% remediation rate, which is misleading.

Visualizations Using Smart Filtering

Visualization	Smart Filter	Reason
MTTR by Severity	Remediated Only	Can only calculate fix time for fixed items
Remediation Rate	Both Statuses	Need both to calculate Active vs Remediated ratio
Remediation Status by Severity	Both Statuses	Compares Active vs Remediated counts
Reopen Rate	Both Statuses	Tracks remediated items that became active again
Resolution Velocity	Remediated Only	Distribution of time-to-fix

Filter Behavior

When smart filtering is active:
- **Date Range**: Still respected (metrics are within selected date range)
- **Severity Filter**: Still respected (can filter to Critical only, etc.)
- **Environment Filter**: Still respected (can filter to Production only)
- **Status Filter**: Overridden to ensure accurate metrics

User Experience

Smart filtering is automatic and transparent. When viewing a chart that uses smart filtering:
1. The data shown reflects accurate metrics regardless of the Status filter selection
2. Other filters (date, severity, environment) still apply normally
3. No user action is required - the system handles this automatically

This ensures that executive dashboards and compliance reports always show accurate remediation metrics, even if an analyst has temporarily filtered to view only active findings for triage work.

Environment Filtering

Environment Types

The tool supports classification of hosts by environment:

Environment	Description	Priority
Production	Live business systems	Highest
PSS	Pre-production/staging	Medium
Shared	Infrastructure used by multiple environments	High (broad impact)
Unknown	Unclassified hosts	Review needed

Hostname Detection

Environments are detected from hostname patterns (9-character format: LLLLTTCEP):

- Position 8: Letter (A-Z) = Production, Number (0-9) = PSS
- Custom mappings can override auto-detection

Configuration

Access environment configuration via the gear icon next to the Environment filter.
Options include:
- Custom environment types
- Explicit hostname mappings
- Pattern-based rules

Using the Visualizations

Interaction Features

All Charts:
- **Double-click**: Open enlarged pop-out view
- **Hover**: See detailed tooltips

Pop-out Windows:
- **Zoom**: +/- buttons for magnification
- **Pan**: Click and drag to move view
- **Labels toggle**: Show/hide data labels
- **Status filter**: Active/Resolved/All
- **Mode filter**: Filtered/All Data/Unique
- **Date filter**: Custom date range
- **Copy to Clipboard**: Save chart image
- **Info button**: View chart documentation

Best Practices

1. **Start with Risk Tab** - Understand overall posture
2. **Use Timeline Tab** - Identify trends and patterns
3. **Check SLA Tab** - Verify compliance status
4. **Drill into Plugin Tab** - Find systemic issues
5. **Review Host Tracking** - Ensure complete coverage

Reporting Recommendations

Weekly:
- New vs Resolved trend
- SLA approaching findings
- Top risky hosts

Monthly:
- MTTR by severity
- Risk trend
- Environment distribution

Quarterly:
- Period comparison
- OPDIR compliance
- Category treemap

Technical Reference

Data Sources

Table	Description
`historical_findings`	All findings with metadata
`finding_lifecycle`	Status and age tracking
`scan_changes`	New/Resolved events
`host_presence`	Scan coverage tracking

Date Field Clarification

Important distinction:
- `first_observed_date`: The timestamp when the vulnerability was first detected by the scanner (from scan output). Used for calculating `days_open`.
- `scan_date`: The date of the scan report. Used for filtering and timeline visualizations.

`days_open` Calculation:

```
days_open = current_date - first_observed_date
```

- Uses the scanner's first observation timestamp, NOT the report date
- Date only, no time-of-day consideration
- More accurate representation of actual exposure duration

Severity Values

Severity	Value	SLA (days)
Critical	4	15
High	3	30
Medium	2	60
Low	1	90

Risk Score Calculation

```
Host Risk = Σ severity_value for all active findings on host
Org Risk = Σ severity_value for all active findings
```

Document generated from chart_descriptions.py
For questions or feedback, consult your security team or tool administrators.

Vulnerability Remediation Analytics: A Data Science Technical Reference

Vulnerability Remediation Analytics: A Data Science Technical Reference

For Data Scientists, Security Analysts, and Technical Stakeholders

Executive Summary

This document provides a comprehensive technical analysis of the vulnerability remediation prioritization system. It covers the mathematical models, algorithms, data structures, and visualization techniques used to transform raw vulnerability scan data into actionable remediation intelligence.

Core Objective: Identify optimal package upgrade sequences that maximize security improvement (findings resolved) while minimizing deployment effort (hosts affected).

Table of Contents

1. Problem Definition

1.1 Business Context

Organizations face a combinatorial optimization challenge: thousands of vulnerabilities across hundreds of hosts, each requiring specific remediation actions. The goal is to:

1. **Maximize security improvement** per unit of deployment effort
2. **Consolidate fragmented versions** to reduce operational complexity
3. **Prioritize by actual risk** (severity-weighted) rather than simple counts

1.2 Mathematical Formulation

Let:
- $P = \{p_1, p_2, ..., p_n\}$ be the set of packages requiring remediation
- $H(p_i)$ = set of hosts affected by package p_i
- $F(p_i)$ = set of findings (vulnerability instances) resolved by upgrading p_i
- $S(f)$ = severity weight of finding f where $S \in \{Critical=4, High=3, Medium=2, Low=1\}$

Objective Function:

```
Maximize: Σᵢ Impact(pᵢ) × Priority(pᵢ)

Where:
  Impact(pᵢ) = Σ_{f ∈ F(pᵢ)} S(f) × |H(pᵢ)|
  Priority(pᵢ) = f(Impact, Effort, Risk)
```

Constraint:

```
Minimize deployment windows = Minimize |{unique hosts touched}|
```

1.3 Key Insight

The system recognizes that upgrading one package (e.g., OpenSSL to 1.1.1w) on one host may resolve multiple findings (CVE-2023-0286, CVE-2022-3602, etc.). The **consolidation multiplier** captures this efficiency:

```
Efficiency(pᵢ) = |F(pᵢ)| / |H(pᵢ)|
```

Higher efficiency = more findings resolved per host deployment.

---

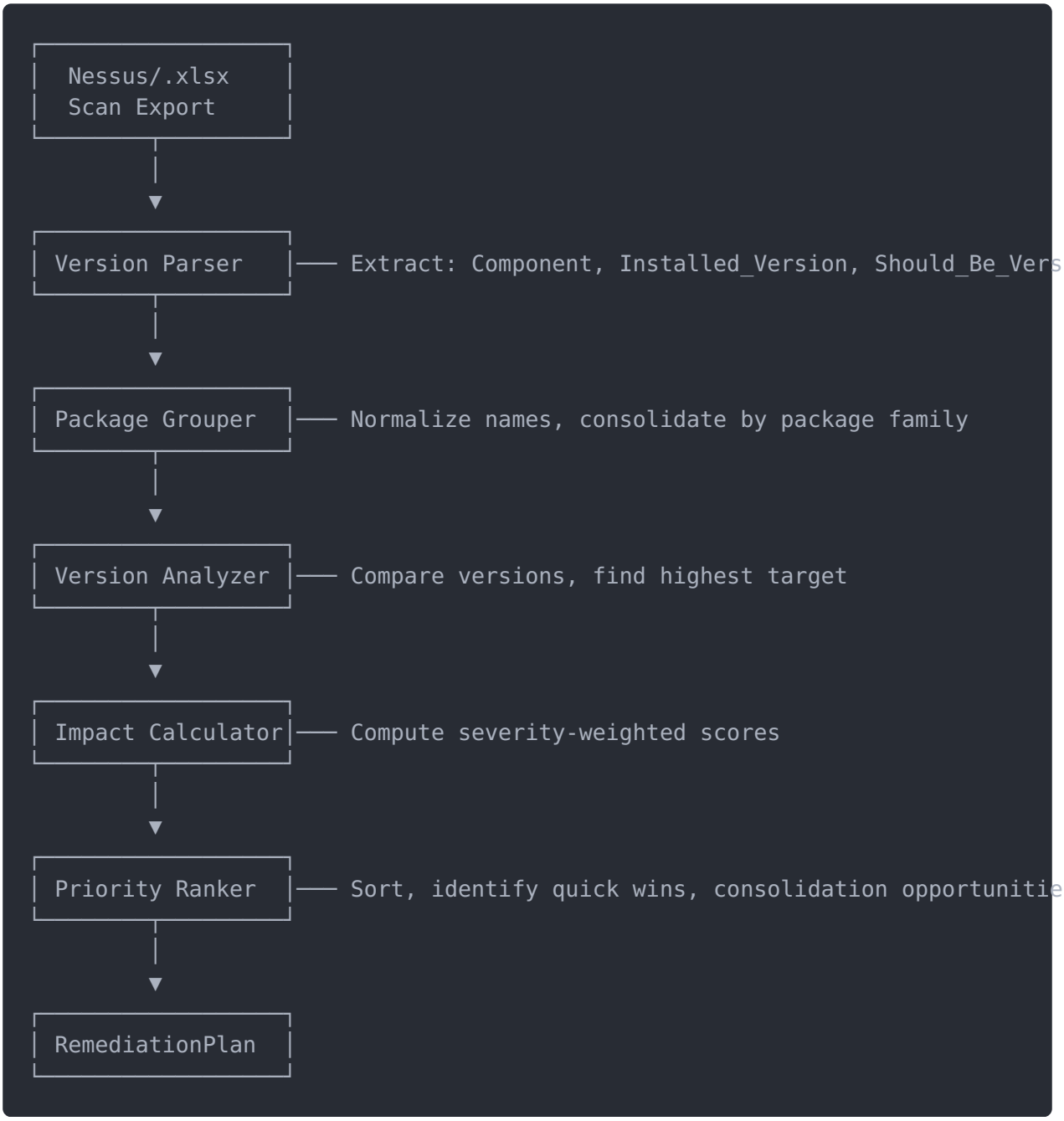
2. Data Model Architecture

2.1 Core Data Classes

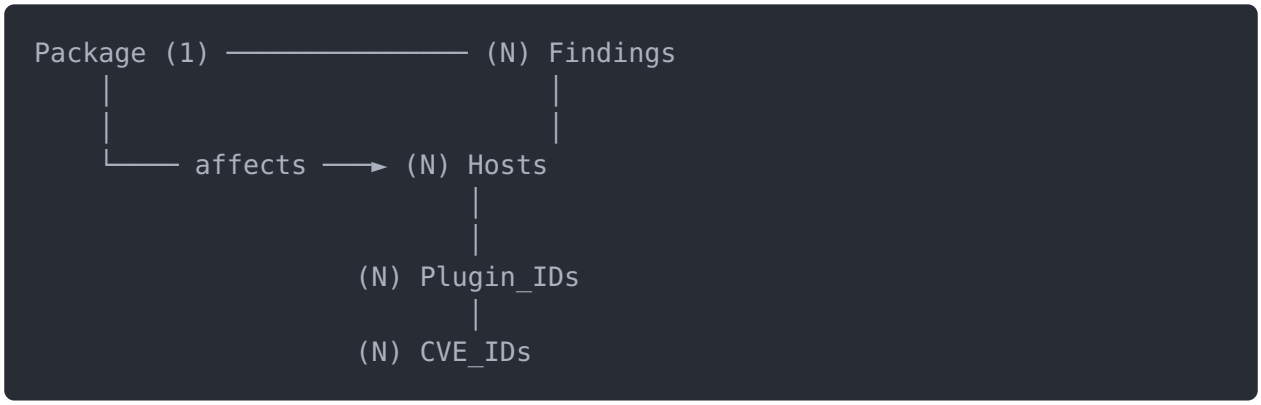
```
@dataclass
class PackageVersionInfo:
    """Information about a specific package version upgrade."""
    package_name: str           # Normalized package identifier
    current_versions: List[str] # All versions found in environment
    target_version: str         # Highest version that resolves all findings
    affected_hosts: int         # |H(p)| - unique hosts
    affected_findings: int      # Unique (host, plugin) combinations
    total_impact: int           # Total finding instances resolved
    plugin_ids: List[str]       # Nessus plugin IDs triggering this finding
    cves: List[str]             # Associated CVE identifiers
    severity_breakdown: Dict[str, int]  # {Critical: n, High: m, ...}
    hosts_list: List[str]       # Enumerated hostname list
```

```
@dataclass
class RemediationPlan:
    """A prioritized remediation plan."""
    packages: List[PackageVersionInfo]  # Sorted by impact_score descending
    total_findings_resolved: int
    total_hosts_affected: int
    total_unique_cves: int
    generated_at: datetime
```

2.2 Data Flow Pipeline



2.3 Entity Relationships



---

3. Impact Scoring Algorithm

3.1 Severity Weighting Rationale

The severity weights follow a non-linear scale reflecting actual risk impact:

Severity	Weight	Rationale
Critical	4	Exploitable remotely, no user interaction, direct system compromise
High	3	Significant impact but may require specific conditions
Medium	2	Limited impact or requires local access
Low	1	Informational or theoretical risk
Info	0	No security impact, compliance/informational

Weight Derivation:
The 4:3:2:1 ratio approximates the CVSS v3 base score distribution:
- Critical (9.0-10.0) → ~4x impact of Low
- High (7.0-8.9) → ~3x impact of Low
- Medium (4.0-6.9) → ~2x impact of Low

3.2 Impact Score Formula

```
@property
def impact_score(self) -> float:
    """Calculate weighted impact score."""
    severity_weights = {
        'Critical': 4,
        'High': 3,
        'Medium': 2,
        'Low': 1,
        'Info': 0
    }

    # Weighted sum of severity counts
    weighted_score = sum(
        count * severity_weights.get(sev, 0)
        for sev, count in self.severity_breakdown.items()
    )

    # Multiply by host count (deployment impact)
    return weighted_score * self.affected_hosts
```

Mathematical Expression:

```
Impact(p) = (4×C + 3×H + 2×M + 1×L) × |Hosts|

Where:
  C = count of Critical findings
  H = count of High findings
  M = count of Medium findings
  L = count of Low findings
  |Hosts| = number of unique hosts affected
```

3.3 Example Calculation

Scenario: OpenSSL package affecting 25 hosts with:
- 15 Critical findings
- 20 High findings
- 10 Medium findings
- 5 Low findings

```
Weighted Sum = (4×15) + (3×20) + (2×10) + (1×5)
             = 60 + 60 + 20 + 5
             = 145

Impact Score = 145 × 25 = 3,625
```

Interpretation: Upgrading OpenSSL across 25 hosts yields 3,625 impact points - a high-priority action.

3.4 Ranking Algorithm

```
# Sort packages by impact score (descending)
sorted_packages = sorted(
    package_analysis.values(),
    key=lambda x: x.impact_score,
    reverse=True
)
```

Time Complexity: O(n log n) where n = number of unique packages

---

4. Version Comparison & Consolidation

4.1 Version String Parsing

Semantic version parsing handles diverse formats:

```
def parse_version_string(version_str: str) -> Tuple[List[int], str]:
    """
    Parse version string into comparable components.

    Examples:
      "1.2.3" → ([1, 2, 3], "1.2.3")
      "java-1.8.0_321" → ([1, 8, 0, 321], "java-1.8.0_321")
      "openssl-1.0.2k" → ([1, 0, 2], "openssl-1.0.2k")
    """
    # Extract all numeric components
    version_match = re.findall(r'(\d+)', str(version_str))
    version_parts = [int(v) for v in version_match] if version_match else []
    return (version_parts, str(version_str))
```

4.2 Version Comparison Logic

```
def compare_versions(v1: str, v2: str) -> int:
    """
    Compare two version strings lexicographically by numeric components.

    Returns: -1 if v1 < v2, 0 if equal, 1 if v1 > v2
    """
    parts1, _ = parse_version_string(v1)
    parts2, _ = parse_version_string(v2)

    # Pad with zeros for equal-length comparison
    max_len = max(len(parts1), len(parts2))
    parts1.extend([0] * (max_len - len(parts1)))
    parts2.extend([0] * (max_len - len(parts2)))

    for p1, p2 in zip(parts1, parts2):
        if p1 < p2: return -1
        elif p1 > p2: return 1
    return 0
```

Example:

```
compare_versions("1.8.0_321", "1.8.0_311") → 1  (321 > 311)
compare_versions("1.1.1k", "1.1.1w") → -1  (k < w numerically? No, 1.1.1 = 1.1.1)
```

4.3 Highest Version Selection

```
def get_highest_version(versions: List[str]) -> str:
    """
    Find the highest version from a list using pairwise comparison.
    """
    valid_versions = [v for v in versions if v and str(v).strip()]
    if not valid_versions:
        return ""

    highest = valid_versions[0]
    for v in valid_versions[1:]:
        if compare_versions(v, highest) > 0:
            highest = v

    return highest
```

Time Complexity: O(n × k) where n = number of versions, k = average version string length

4.4 Consolidation Detection

The system identifies packages with excessive version fragmentation:

```
Fragmentation Index = |unique current versions|

High fragmentation (>5 versions) indicates:
  1. Inconsistent patching practices
  2. Multiple application teams with different upgrade schedules
  3. Technical debt requiring standardization
```

---

5. Pareto Analysis (80/20 Rule)

5.1 Cumulative Impact Calculation

```
def calculate_cumulative_impact(plan: RemediationPlan) -> pd.DataFrame:
    """
    Calculate cumulative remediation coverage as packages are addressed.

    Demonstrates: How many packages needed to resolve X% of findings
    """
    cumulative_findings = 0
    cumulative_hosts = set()
    cumulative_cves = set()

    data = []
    for i, pkg in enumerate(plan.packages):  # Already sorted by impact
        cumulative_findings += pkg.total_impact
        cumulative_hosts.update(pkg.hosts_list)
        cumulative_cves.update(pkg.cves)

        data.append({
            'Priority': i + 1,
            'Package': pkg.package_name,
            'Findings_Resolved': pkg.total_impact,
            'Cumulative_Findings': cumulative_findings,
            'Cumulative_Findings_Pct': round(
                cumulative_findings / plan.total_findings_resolved * 100, 1
            ),
            'Cumulative_Hosts': len(cumulative_hosts),
            'Cumulative_CVEs': len(cumulative_cves)
        })

    return pd.DataFrame(data)
```

5.2 Finding the 80% Threshold

```
# Find number of packages needed for 80% coverage
packages_for_80 = next(
    (i + 1 for i, pct in enumerate(y_pct) if pct >= 80),
    len(plan.packages)
)
```

5.3 Typical Results

In most vulnerability datasets, Pareto's principle holds:
- **20% of packages** resolve **~80% of findings**
- Top 5-10 packages often provide maximum efficiency

Example Output:

```
Priority  Package       Findings  Cumulative%
1         openssl       450       22.5%
2         java-openjdk  380       41.5%
3         curl          210       52.0%
4         glibc         180       61.0%
5         kernel        150       68.5%
...
10        apache        80        83.2%  ← 80% threshold reached
```

5.4 Visualization

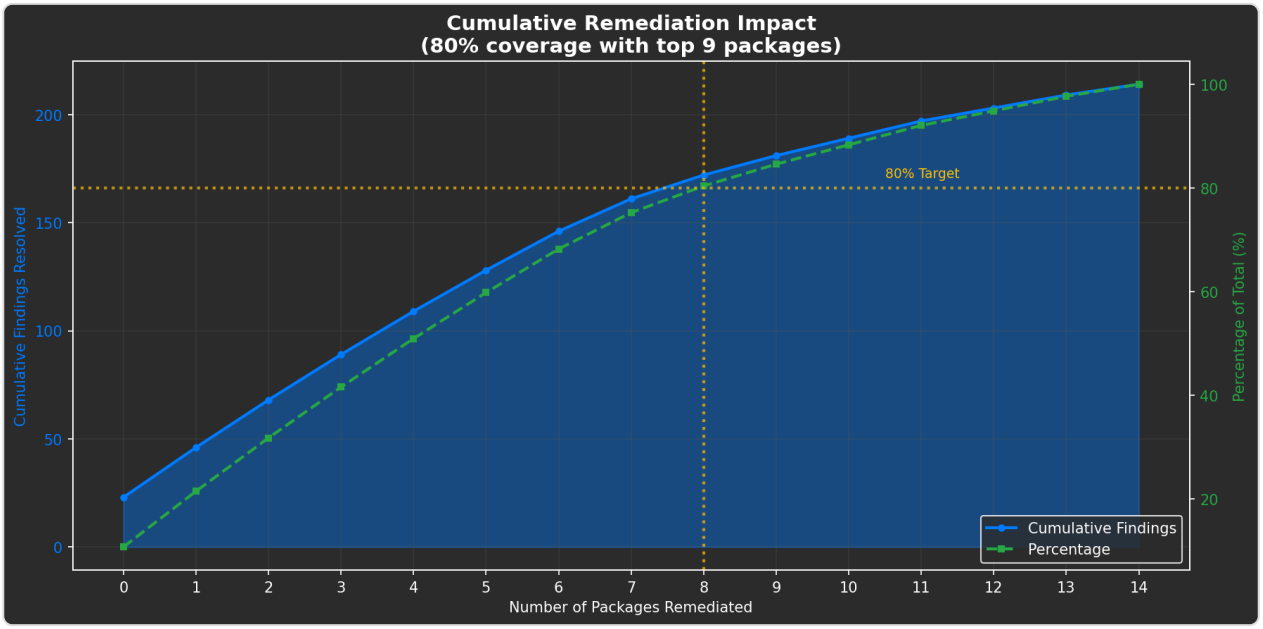


Chart Elements:
- **Blue area/line:** Cumulative findings resolved
- **Green dashed line:** Percentage of total
- **Yellow dotted line:** 80% target threshold
- **Vertical line:** Point where 80% is achieved

---

6. Quick Wins Optimization

6.1 Definition

Quick Win: A package upgrade that:
- Affects **few hosts** (≤10) → Low deployment complexity
- Resolves **many findings** (≥5) → High security value

6.2 Efficiency Metric

```
efficiency = total_impact / max(affected_hosts, 1)
```

Interpretation: Findings resolved per host deployment

6.3 Identification Algorithm

```
def identify_quick_wins(plan, max_hosts=10, min_findings=5):
    """
    Filter packages meeting quick-win criteria and sort by efficiency.
    """
    quick_wins = [
        p for p in plan.packages
        if p.affected_hosts <= max_hosts and p.total_impact >= min_findings
    ]

    return sorted(
        quick_wins,
        key=lambda x: x.total_impact / max(x.affected_hosts, 1),
        reverse=True
    )[:15]
```

6.4 Business Value

Quick wins provide:
1. **Fast initial progress** for stakeholder reporting
2. **Low-risk deployments** to build team confidence
3. **Maximum ROI** on limited maintenance windows

Example Quick Win:

```
Package: libxml2
Hosts: 3
Findings: 45
Efficiency: 15.0 findings/host

Action: Upgrade 3 hosts → Resolve 45 vulnerabilities
```

---

7. CVE Validation Framework

7.1 NVD API Integration

```
class CVEValidator:
    """
    Validate package versions against CVE data from NVD.
    """

    def __init__(self, nvd_api_key: str = None):
        self.base_url = "https://services.nvd.nist.gov/rest/json/cves/2.0"
        self.api_key = nvd_api_key
        self.cache = {}  # {cve_id: response}

    def validate_package_versions(
        self,
        package_name: str,
        target_version: str,
        cve_list: List[str]
    ) -> ValidationResult:
        """
        Verify target version resolves listed CVEs.
        """
        resolved = []
        unresolved = []

        for cve in cve_list:
            cve_data = self._fetch_cve(cve)
            if self._version_resolves_cve(target_version, cve_data):
                resolved.append(cve)
            else:
                unresolved.append(cve)

        return ValidationResult(
            package_name=package_name,
            target_version=target_version,
            cves_resolved=resolved,
            cves_unresolved=unresolved,
            is_valid=len(unresolved) == 0
        )
```

7.2 Rate Limiting

NVD API has strict rate limits:

Access Level	Limit
Anonymous	5 requests/30 seconds
With API Key	50 requests/30 seconds

Implementation:

```
# 0.6 second delay between requests (anonymous)
time.sleep(0.6)

# Caching to avoid repeated lookups
if cve_id in self.cache:
    return self.cache[cve_id]
```

7.3 Validation Output

```
{
  "package": "openssl",
  "target_version": "1.1.1w",
  "cves_resolved": ["CVE-2023-0286", "CVE-2022-3602", "CVE-2022-3786"],
  "cves_unresolved": [],
  "is_valid": true,
  "confidence": 0.95
}
```

---

8. Visualization Design Rationale

8.1 Dark Theme Design System

All visualizations use a consistent dark theme optimized for:
- **Extended viewing** (reduced eye strain)
- **Executive presentations** (professional appearance)
- **Data density** (dark backgrounds allow more contrast)

```
def get_dark_style():
    return {
        'figure.facecolor': '#2b2b2b',
        'axes.facecolor': '#2b2b2b',
        'axes.edgecolor': 'white',
        'axes.labelcolor': 'white',
        'text.color': 'white',
        'xtick.color': 'white',
        'ytick.color': 'white',
        'grid.color': '#555555',
        'legend.facecolor': '#2b2b2b',
        'legend.edgecolor': 'white'
    }
```

8.2 Severity Color Encoding

Consistent semantic colors across all visualizations:

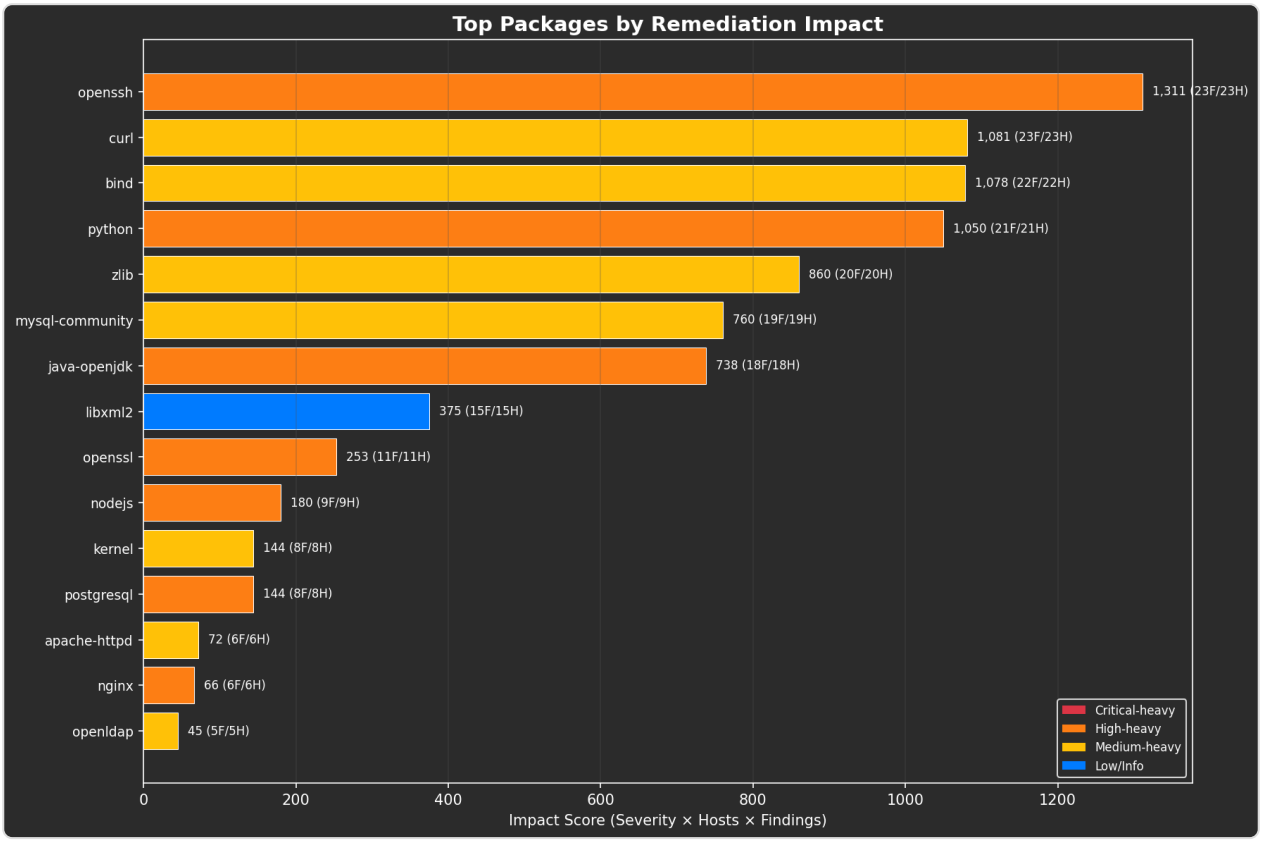
```
SEVERITY_COLORS = {
    'Critical': '#dc3545',  # Red - danger
    'High': '#fd7e14',      # Orange - warning
    'Medium': '#ffc107',    # Yellow - caution
    'Low': '#007bff',       # Blue - informational
    'Info': '#6c757d'       # Gray - neutral
}
```

8.3 Chart Type Selection Matrix

Data Type	Best Visualization	Rationale
Ranked comparison	Horizontal bar	Easy label reading
Trend over time	Line/Area chart	Shows trajectory
Part-to-whole	Stacked bar	Shows composition
Multi-dimensional	Bubble chart	Encodes 3+ variables
Distribution	Histogram	Shows spread
Cumulative progress	Area chart + line	Shows acceleration

8.4 Visualization Portfolio

A. Package Impact Ranking

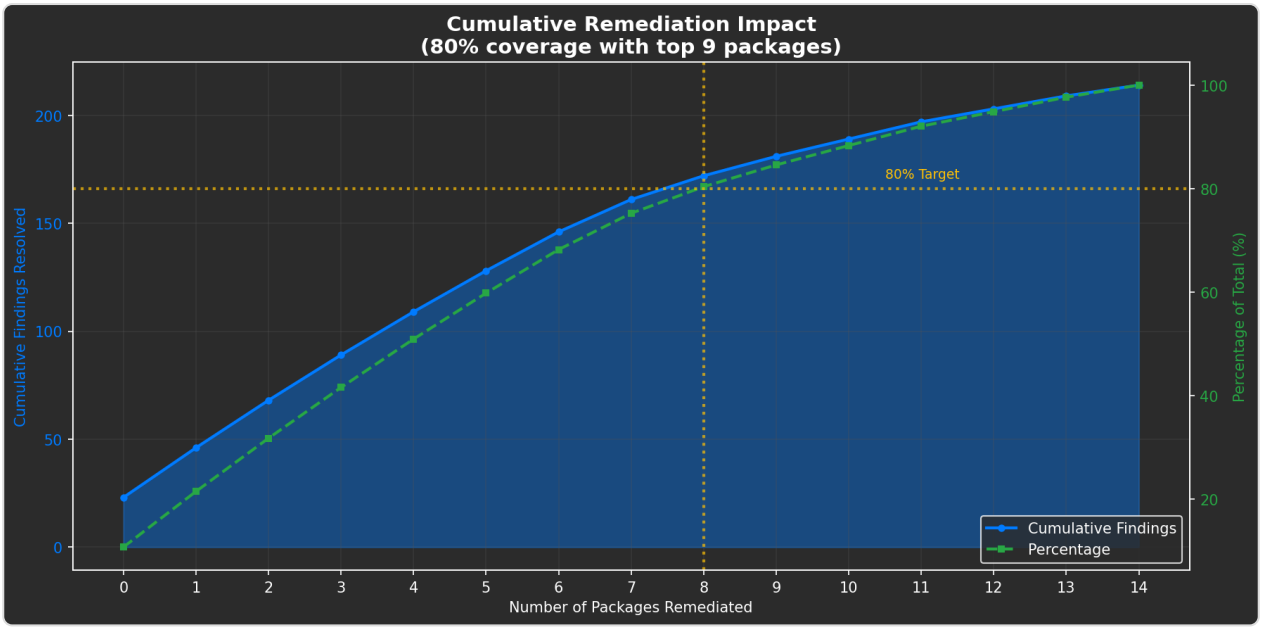


Purpose: Rank packages by remediation impact

Encoding:

- Bar length = Impact score
- Bar color = Primary severity (red=Critical-heavy)
- Labels = "Impact (Findings/Hosts)"

B. Cumulative Impact (Pareto)

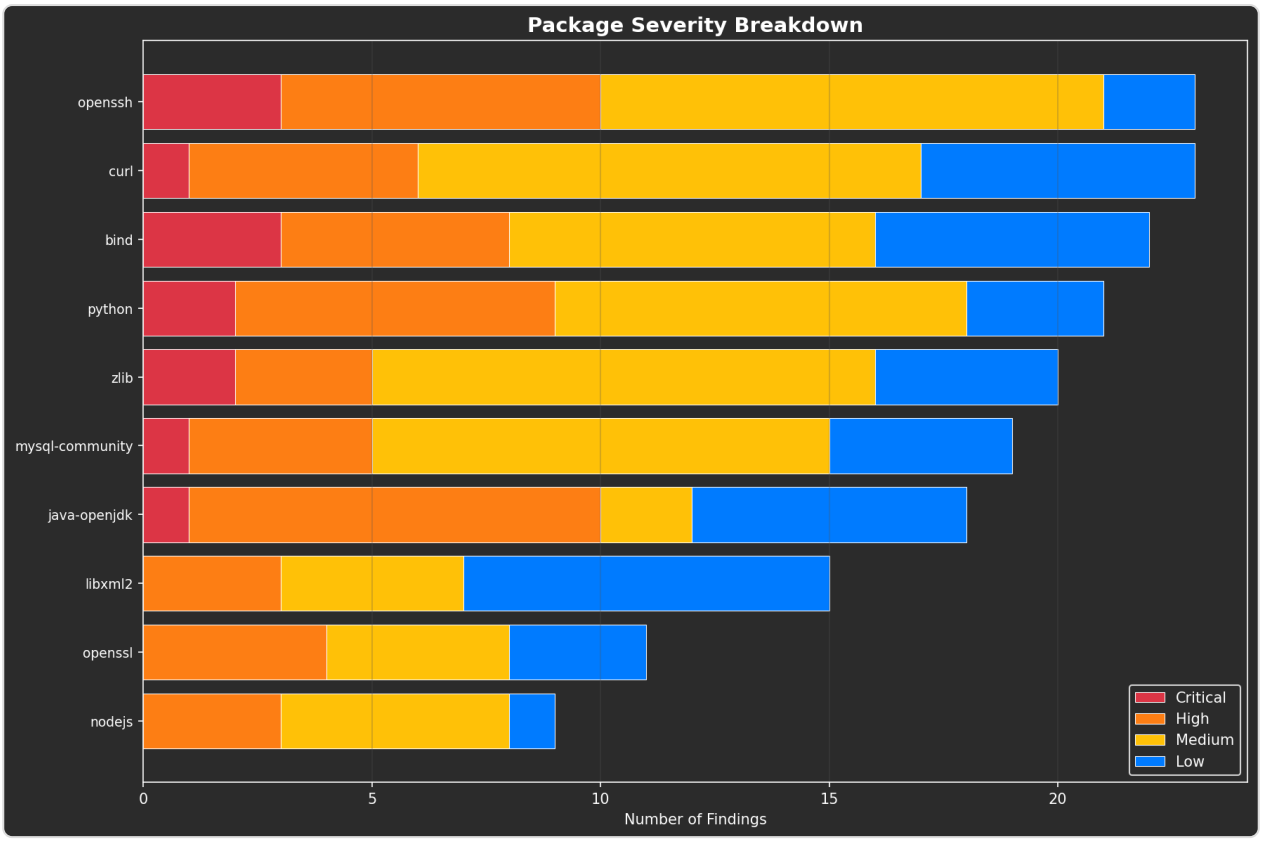


Purpose: Show 80/20 analysis

Encoding:

- Blue area = Cumulative findings
- Green line = Percentage of total
- Yellow line = 80% threshold

C. Severity Breakdown (Stacked)

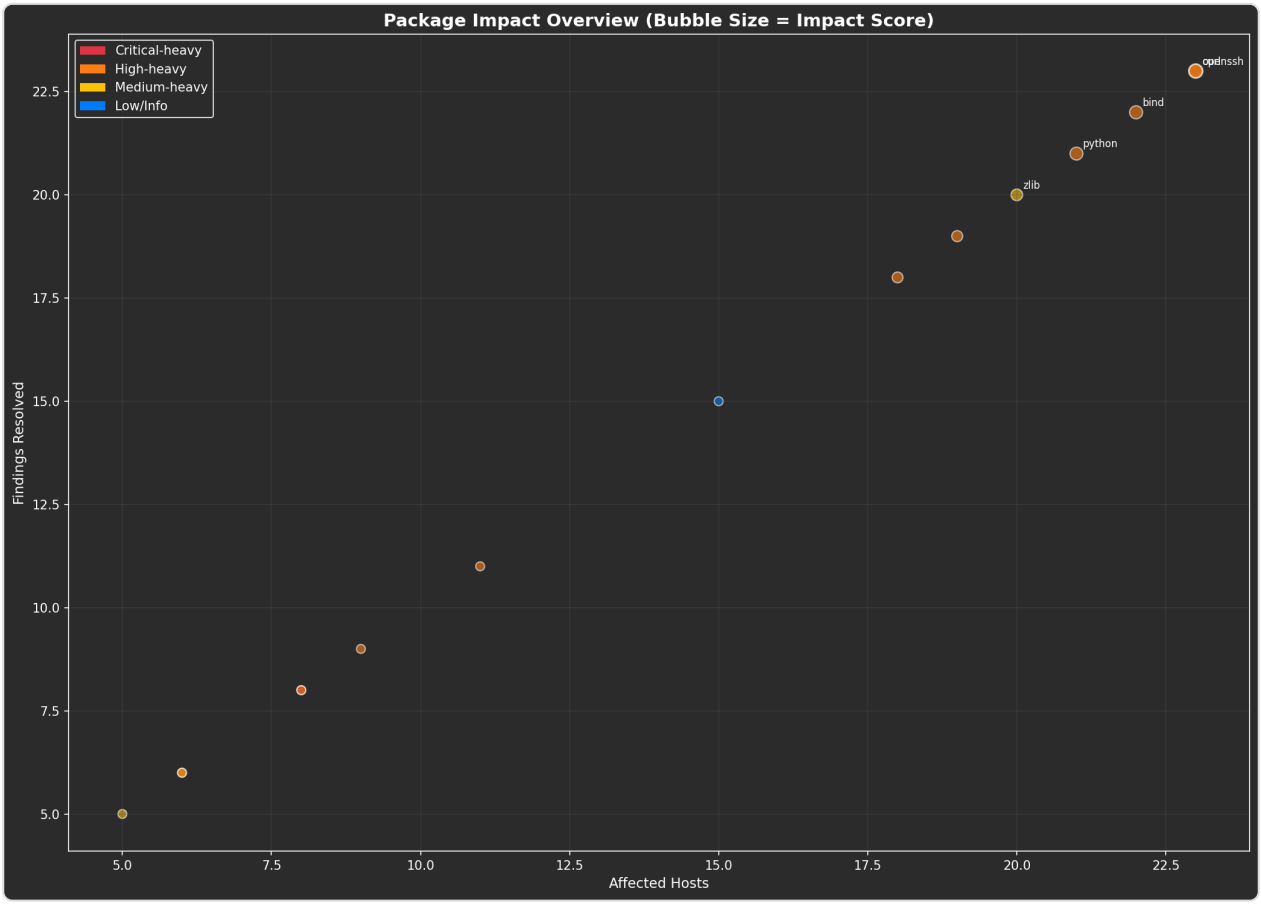


Purpose: Show severity composition per package

Encoding:

- Each segment = One severity level
- Colors follow severity palette

D. Impact Bubble Chart

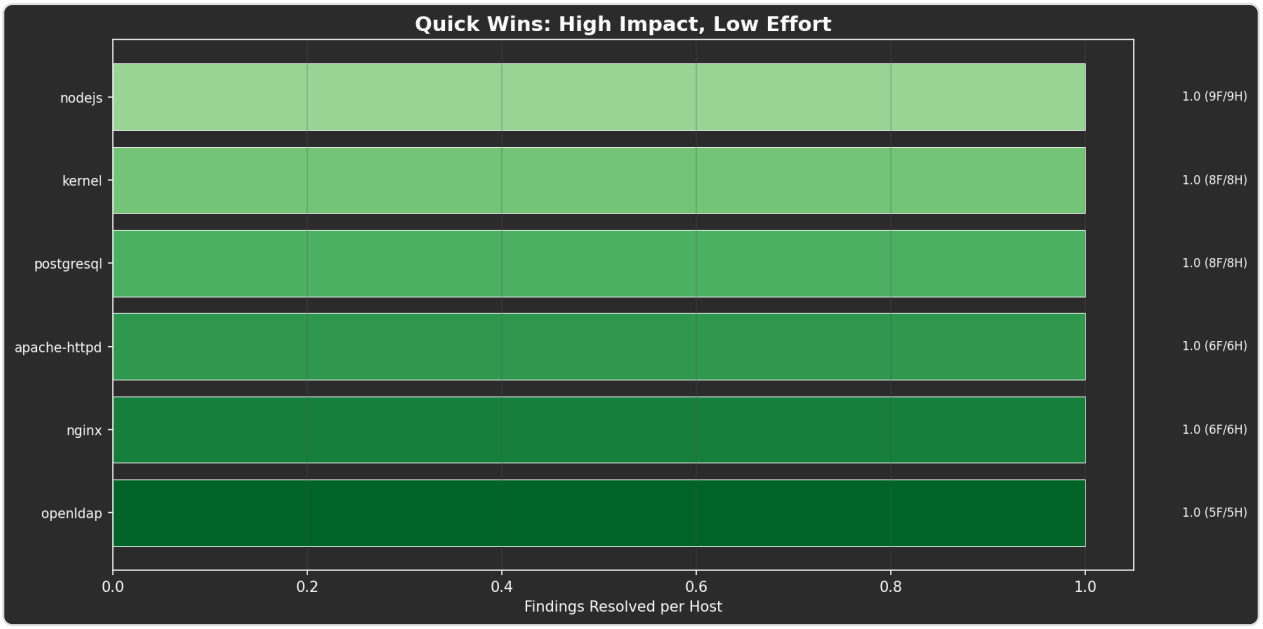


Purpose: Multi-dimensional view

Encoding:

- X-axis = Hosts affected
- Y-axis = Findings resolved
- Bubble size = Impact score
- Color = Primary severity

E. Quick Wins



Purpose: Identify high-efficiency targets

Encoding:

- Bar length = Efficiency (findings/host)
- Color gradient = Green (efficiency)
- Labels = "Efficiency (Findings/Hosts)"

---

9. Statistical Methods

9.1 Descriptive Statistics

For each remediation plan:

```
def calculate_statistics(plan):
    impacts = [p.impact_score for p in plan.packages]
    hosts = [p.affected_hosts for p in plan.packages]
    findings = [p.total_impact for p in plan.packages]

    return {
        'impact': {
            'mean': np.mean(impacts),
            'median': np.median(impacts),
            'std': np.std(impacts),
            'min': np.min(impacts),
            'max': np.max(impacts),
            'quartiles': np.percentile(impacts, [25, 50, 75])
        },
        'hosts': {
            'total_unique': plan.total_hosts_affected,
            'mean_per_package': np.mean(hosts),
            'max_concentration': np.max(hosts)
        },
        'findings': {
            'total': plan.total_findings_resolved,
            'mean_per_package': np.mean(findings)
        }
    }
```

9.2 Distribution Analysis

The impact score distribution typically follows a **power law**:
- Few packages have very high impact
- Many packages have low impact

This validates the Pareto (80/20) approach.

9.3 Correlation Analysis

Useful correlations to examine:

Variables	Expected Correlation	Interpretation
Hosts × Findings	Positive	More hosts = more findings
Severity × Age	Positive	Critical items should be newer (faster remediation)
Version Count × Impact	Positive	More fragmentation = more findings

9.4 Effort Estimation Model

```
def estimate_remediation_effort(plan):
    total_packages = len(plan.packages)

    # Effort categories based on package count
    if total_packages <= 5:
        effort_level = 'Low'       # 1-2 maintenance windows
    elif total_packages <= 15:
        effort_level = 'Medium'    # 1 week of work
    elif total_packages <= 30:
        effort_level = 'High'      # 2-4 weeks
    else:
        effort_level = 'Very High' # Major initiative

    return effort_level
```

10. Code Reference

10.1 Module Structure

```
refactored_app/
├── analysis/
│   ├── package_version_impact.py   # Core analysis
│   │   ├── PackageVersionInfo      # Data class
│   │   ├── RemediationPlan         # Result container
│   │   ├── analyze_package_version_impact()  # Main entry
│   │   ├── create_remediation_summary_df()   # DataFrame export
│   │   ├── calculate_cumulative_impact()     # Pareto analysis
│   │   └── estimate_remediation_effort()     # Effort model
│   │
│   └── cve_validation.py           # CVE verification
│       ├── CVEValidator            # NVD API client
│       └── create_cve_validation_report()
│
├── visualization/
│   ├── package_impact_charts.py    # Individual charts
│   │   ├── create_package_impact_bar_chart()
│   │   ├── create_cumulative_impact_chart()
│   │   ├── create_severity_breakdown_chart()
│   │   ├── create_host_distribution_chart()
│   │   ├── create_version_consolidation_chart()
│   │   ├── create_cve_coverage_chart()
│   │   ├── create_impact_bubble_chart()
│   │   └── create_quick_wins_chart()
│   │
│   └── remediation_dashboard.py    # Multi-chart dashboards
│       ├── create_remediation_impact_dashboard()
│       ├── create_executive_remediation_summary()
│       └── create_host_impact_dashboard()
│
└── gui/
    └── package_impact_dialog.py    # Interactive GUI
        └── PackageImpactDialog     # Tkinter dialog
```

10.2 Key Function Signatures

```
# Main analysis entry point
def analyze_package_version_impact(
    version_df: pd.DataFrame,       # Input: version extractor output
    findings_df: pd.DataFrame = None, # Optional: for severity lookup
    min_impact: int = 1             # Filter threshold
) -> RemediationPlan:
    """
    Returns prioritized RemediationPlan sorted by impact_score descending.
    """

# Summary export
def create_remediation_summary_df(plan: RemediationPlan) -> pd.DataFrame:
    """
    Returns DataFrame with columns:
    Package, Target_Version, Affected_Hosts, Findings_Resolved,
    Impact_Score, Critical, High, Medium, Low, CVE_Count, Plugin_Count
    """

# Pareto analysis
def calculate_cumulative_impact(plan: RemediationPlan) -> pd.DataFrame:
    """
    Returns DataFrame with columns:
    Priority, Package, Findings_Resolved, Cumulative_Findings,
    Cumulative_Findings_Pct, Cumulative_Hosts, Cumulative_CVEs
    """
```

10.3 Usage Example

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

# Load version extractor output
version_df = pd.read_excel('version_data.xlsx')
findings_df = pd.read_excel('findings.xlsx')

# Analyze
plan = analyze_package_version_impact(version_df, findings_df)

# Summary statistics
print(f"Total packages: {len(plan.packages)}")
print(f"Total findings resolved: {plan.total_findings_resolved}")
print(f"Total hosts affected: {plan.total_hosts_affected}")
print(f"Total CVEs addressed: {plan.total_unique_cves}")

# Pareto analysis
cumulative = calculate_cumulative_impact(plan)
packages_for_80 = cumulative[cumulative['Cumulative_Findings_Pct'] >= 80].iloc[0]['Pric
print(f"Packages needed for 80% coverage: {packages_for_80}")

# Effort estimate
effort = estimate_remediation_effort(plan)
print(f"Effort level: {effort['effort_level']}")
print("Recommendations:")
for rec in effort['recommendations']:
    print(f"  - {rec}")

# Generate visualizations
fig = create_remediation_impact_dashboard(plan)
fig.savefig('dashboard.png', dpi=150, bbox_inches='tight')

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

11. Future Enhancements

11.1 Machine Learning Opportunities

1. **Remediation Time Prediction**
   - Features: Package type, host count, severity mix, historical data
   - Model: Random Forest or XGBoost regression
   - Target: Days to complete remediation

2. **Failure Risk Prediction**
   - Predict packages likely to fail validation or reappear
   - Features: Past reopen rate, package complexity, OS diversity

3. **Optimal Scheduling**
   - Constraint optimization for maintenance windows
   - Balance: Impact, dependencies, change freeze periods

11.2 Advanced Analytics

1. **Dependency Graph Analysis**
   - Track package dependencies to identify cascade effects
   - Recommend upgrade sequences that minimize conflicts

2. **Threat Intelligence Correlation**
   - Integrate CISA KEV, EPSS scores
   - Prioritize actively exploited vulnerabilities

3. **Anomaly Detection**
   - Flag unusual patterns (sudden version fragmentation, unexpected findings)
   - Alert on scan coverage gaps

11.3 Visualization Enhancements

1. **Interactive Dashboards**
   - Drill-down from package to hosts to specific findings
   - Filter by environment, date range, severity

2. **Network Visualization**
   - Show package-host relationships as graph
   - Cluster analysis for co-occurring packages

3. **Time Series Forecasting**
   - Predict future finding accumulation
   - SLA breach forecasting

Appendix A: Screenshot Reference

Screenshot	Description
`01_remediation_dashboard.png`	Full 6-panel remediation dashboard
`02_executive_summary.png`	Key metrics and top packages
`03_package_impact_ranking.png`	Ranked bar chart by impact
`04_cumulative_impact.png`	Pareto analysis chart
`05_severity_breakdown.png`	Stacked severity composition
`06_host_distribution.png`	Hosts per package analysis
`07_version_consolidation.png`	Version fragmentation view
`08_quick_wins.png`	High-efficiency packages
`09_impact_bubble.png`	Multi-dimensional bubble chart
`10_host_impact_dashboard.png`	Host-centric analysis
`11_remediation_list.png`	Prioritized table view
`12_effort_estimate.png`	Effort estimation summary

Appendix B: Glossary

Term	Definition
Impact Score	Severity-weighted measure of remediation value
Quick Win	Low-effort, high-value remediation target
Pareto Analysis	80/20 rule application to find highest-value items
Version Consolidation	Standardizing fragmented versions to single target
MTTR	Mean Time to Remediate
Finding	Single vulnerability instance on one host
Plugin	Nessus detection rule identifying vulnerabilities
CVE	Common Vulnerabilities and Exposures identifier

Document Version: 1.0
Last Updated: December 2024
Author: Plugin History Analysis Team