VersionIntel Complete Export

Exported on: 2025-07-15 10:41:37

# Export Summary

|  |  |
| --- | --- |
| **Metric** | **Count** |
| Total Vendors | 2 |
| Total Products | 9 |
| Total Methods | 0 |
| Total Guides | 0 |

# Vendor: Apache

## Product: Answer

|  |  |
| --- | --- |
| Category | Web Application |
| Description | # Apache Answer Version Detection Guide  ## 1. What is this product used for?  Apache Answer is a Q&A platform software for teams at any scales, whether it's a community forum, help center, or knowledge management platform. It enables organizations to build robust question-and-answer platforms for knowledge sharing, community building, and customer support.  ## 2. What is the type of this software?  \*\*Web Application\*\*  Apache Answer is mainly built using Golang for the backend and React.js for the frontend. It's a web application that provides both a web interface and REST API endpoints for Q&A platform functionality, running as a standalone web server.  ## 3. Remote Version Detection Methods (No Credentials Required)  ### HTTP Headers - \*\*Server Response Headers\*\*: Custom server headers that may include version information in HTTP responses - \*\*X-Powered-By Headers\*\*: Potential framework or version identification headers - \*\*Content-Type Headers\*\*: Application-specific content type handling that might vary between versions - \*\*API Response Headers\*\*: REST API endpoints may include version-specific response headers  ### HTML Body Detection - \*\*Meta Tags\*\*: HTML meta tags in the web interface that may contain version information - \*\*Footer Information\*\*: Version details displayed in page footers or about sections - \*\*JavaScript Variables\*\*: Client-side JavaScript that might expose version constants - \*\*Error Pages\*\*: Custom error pages that may reveal version information - \*\*Installation Page\*\*: Initial setup at /install endpoint may expose version details  ### API Endpoints - \*\*Installation Endpoint\*\*: /install endpoint used for initial setup and configuration - \*\*Health Check Endpoints\*\*: Standard health check routes that may include version metadata - \*\*API Version Endpoints\*\*: REST API routes that might expose version information - \*\*Configuration Endpoints\*\*: Admin API endpoints that could reveal version-specific configurations - \*\*User Registration/Login\*\*: Authentication endpoints that may include version information in responses - \*\*Plugin API Endpoints\*\*: Plugin system endpoints that might expose version compatibility information  ### Static Asset Paths (JS/CSS) - \*\*Frontend Assets\*\*: React.js frontend assets with version-specific file naming or content hashes - \*\*JavaScript Bundle Names\*\*: Webpack or build tool generated filenames with version information - \*\*CSS Stylesheets\*\*: Version-specific styling that might include build numbers or hashes - \*\*Image Assets\*\*: Logo or branding images that change between versions - \*\*Favicon\*\*: Version-specific favicon files or paths  ### Error Messages or Stack Traces - \*\*Go Runtime Errors\*\*: Go-based backend may expose stack traces with version-specific package information - \*\*Database Connection Errors\*\*: Error messages that might reveal version-specific database handling - \*\*API Error Responses\*\*: JSON error responses with version-dependent field structures - \*\*Frontend JavaScript Errors\*\*: React.js error messages that might include build or version information  ### TLS Certificate Clues - \*\*Certificate Subject/Issuer\*\*: TLS certificates that might contain organization-specific version information - \*\*Certificate Extensions\*\*: Custom certificate fields that could indicate version details - \*\*Certificate Common Names\*\*: Version-specific certificate naming conventions  ### robots.txt or sitemap.xml - \*\*Robots.txt\*\*: May reveal version-specific admin paths or restricted areas - \*\*Sitemap.xml\*\*: Could expose version-specific URL structures - \*\*Security.txt\*\*: Modern security disclosure files that might include version information  ### Raw Socket/Banner Grabs - \*\*HTTP Port Banner\*\*: Default port 80 (or 9080 for Docker) HTTP server banner information - \*\*TLS Handshake\*\*: SSL/TLS connection handshake that might reveal server version details - \*\*HTTP/2 or HTTP/3\*\*: Protocol version support that might indicate specific Apache Answer versions  ### Custom Protocol Detection - \*\*REST API Protocol\*\*: JSON-based REST API that follows specific version patterns - \*\*WebSocket Endpoints\*\*: Real-time communication endpoints that might include version headers - \*\*Plugin Communication\*\*: Plugin system communication protocols that might reveal version compatibility  ### Java-Specific Techniques \*\*Java techniques are NOT applicable\*\* for Apache Answer since it's built with Golang and React.js, not Java. Go-specific detection methods would be more relevant.  ## 4. Indirect or Fingerprintable Signs of Version/Presence  ### Static Asset Naming/Hash Patterns - \*\*React.js Build Hashes\*\*: Content-based hashing for JavaScript and CSS files that change between versions - \*\*Webpack Bundle Names\*\*: Build tool generated asset names with version-specific patterns - \*\*Source Map Files\*\*: Development artifacts that might expose version information  ### Unique HTML Structures - \*\*Component Architecture\*\*: React.js component structures that evolve between versions - \*\*CSS Class Names\*\*: Version-specific CSS class naming conventions - \*\*DOM Element Patterns\*\*: HTML structure patterns unique to specific versions - \*\*Form Field Arrangements\*\*: UI form layouts that change between releases  ### Field Names in API Responses - \*\*User Object Fields\*\*: JSON response fields for user data that might be version-specific - \*\*Question/Answer Fields\*\*: Q&A specific API response structures that evolve over time - \*\*Configuration Parameters\*\*: System configuration fields exposed through API responses - \*\*Plugin Configuration\*\*: Plugin system configuration fields that indicate version capabilities  ### Plugin Paths - \*\*Plugin Installation Paths\*\*: Plugin system paths that might reveal version-specific plugin architecture - \*\*Plugin API Endpoints\*\*: Version-specific plugin communication endpoints - \*\*Plugin Asset Paths\*\*: Static assets served by plugins with version-dependent structures  ### Stack Traces or Internal File References - \*\*Go Package Paths\*\*: Internal Go package references that might expose version information - \*\*Source Code References\*\*: Debug information or error messages revealing internal file structures - \*\*Configuration File Paths\*\*: References to version-specific configuration file locations - \*\*Database Schema References\*\*: Error messages exposing version-specific database structures  ### Error Page Structure - \*\*HTTP Status Code Pages\*\*: Custom error page designs that change between versions - \*\*JSON Error Formats\*\*: API error response structures that evolve over time - \*\*React.js Error Boundaries\*\*: Frontend error handling patterns specific to certain versions  ### Response Headers or Clues - \*\*Custom Application Headers\*\*: Apache Answer specific HTTP headers - \*\*Session Management Headers\*\*: Cookie or session handling that might be version-dependent - \*\*CORS Headers\*\*: Cross-origin resource sharing configuration that could indicate version - \*\*Security Headers\*\*: Security-related headers that might change between versions  ## 5. Summary Table  | Detection Vector | Endpoint/Method | Expected Output/Signature | |------------------|-----------------|---------------------------| | \*\*Installation Page\*\* | `GET /install` | Setup wizard interface with potential version information | | \*\*Default Home Page\*\* | `GET /` | Q&A platform interface with footer/meta version details | | \*\*API Base Endpoint\*\* | `GET /api/` | REST API base route with potential version headers | | \*\*Static Assets\*\* | `GET /assets/\*`, `/static/\*` | React.js bundles with version-specific naming patterns | | \*\*Health Check\*\* | `GET /health`, `/status` | System status with potential version metadata | | \*\*User Registration\*\* | `POST /api/user/register` | API response structure that might include version info | | \*\*Login Endpoint\*\* | `POST /api/user/login` | Authentication API with version-specific response format | | \*\*Questions API\*\* | `GET /api/questions` | Q&A data structure that might reveal version capabilities | | \*\*Plugin Endpoints\*\* | `GET /api/plugins/\*` | Plugin system API with version compatibility information | | \*\*Admin Interface\*\* | `GET /admin/\*` | Administrative interface with version details | | \*\*Configuration API\*\* | `GET /api/config` | System configuration with version-specific parameters | | \*\*Error Responses\*\* | Invalid API calls | Go runtime errors with package version information | | \*\*WebSocket Endpoints\*\* | WebSocket upgrade requests | Real-time communication with potential version headers | | \*\*Database Errors\*\* | Database connection failures | Error messages revealing version-specific database handling |  \*\*Note\*\*: Version detection success depends on configuration settings, whether debugging information is exposed, and specific deployment methods. The /install endpoint is only available during initial setup and may be disabled after configuration is complete. Docker deployments may run on different ports (9080) compared to binary installations (80). |

## Product: pulsar

|  |  |
| --- | --- |
| Category | Service Application |
| Description | ## Apache Pulsar Version Detection Guide  ## 1. What is this product used for?  Apache Pulsar is an open-source, distributed messaging and streaming platform built for the cloud that provides multi-tenancy, high-performance messaging with tiered storage capabilities. It serves as an all-in-one messaging and streaming platform for real-time data processing, event streaming, and pub-sub messaging across distributed systems.  ## 2. What is the type of this software?  \*\*Service Application\*\*  Apache Pulsar is a service application that runs as a distributed system with multiple components including: - Pulsar brokers (stateless message handling servers) - Apache BookKeeper bookies (persistent storage) - Apache ZooKeeper (metadata coordination) - Optional web interfaces and admin consoles  ## 3. Remote Version Detection Methods (No Credentials Required)  ### HTTP Headers - \*\*Admin API Server Headers\*\*: Pulsar brokers expose an HTTP server with a REST interface for administration on port 8080 by default - \*\*Server Response Headers\*\*: Custom server headers may be present in HTTP responses from admin endpoints - \*\*CORS Headers\*\*: Cross-origin headers that might contain version-specific information  ### HTML Body Detection - \*\*Admin Console UI\*\*: Third-party web interfaces like Pulsar Manager, Pulsar Express, or DataStax Admin Console may expose version information in HTML meta tags or JavaScript - \*\*Error Pages\*\*: HTTP error responses from the admin API may contain version information in HTML format - \*\*Built-in Web Interfaces\*\*: If enabled, built-in web interfaces may show version information  ### API Endpoints - \*\*Health Check Endpoint\*\*: `/admin/v2/brokers/health` - The health check endpoint available via REST API - \*\*Broker Information\*\*: `/admin/v2/brokers/configuration` - Dynamic configuration endpoint that may reveal version-specific parameters - \*\*Metrics Endpoint\*\*: `/metrics` - Prometheus format metrics that may include version information - \*\*Broker Stats\*\*: `/admin/v2/broker-stats/\*` endpoints that may include version metadata - \*\*Cluster Information\*\*: `/admin/v2/clusters` endpoint for cluster configuration details - \*\*Schema Registry\*\*: `/admin/v2/schemas/\*` endpoints that might reveal version-specific functionality  ### Static Asset Paths (JS/CSS) - \*\*Admin Console Assets\*\*: JavaScript and CSS files from web management interfaces may contain version information in filenames or content - \*\*WebSocket Proxy Assets\*\*: If WebSocket service is enabled, static assets may be served with version-specific paths - \*\*Monitoring Dashboard Assets\*\*: Grafana or custom monitoring dashboards may expose version information  ### Error Messages or Stack Traces - \*\*Java Stack Traces\*\*: Pulsar error messages may expose Java stack traces revealing specific jar file versions like "com.datastax.oss-pulsar-broker-2.8.0.1.1.22.jar" - \*\*Authentication Errors\*\*: Failed authentication attempts may reveal version-specific error message formats - \*\*API Version Errors\*\*: Invalid API calls may return errors indicating supported API versions - \*\*Connection Errors\*\*: Network connection failures may expose internal component versions  ### TLS Certificate Clues - \*\*Certificate Subject/Issuer\*\*: TLS certificates may contain organization or version-specific information - \*\*Certificate Extensions\*\*: Custom certificate extensions that might indicate Pulsar version - \*\*Certificate Common Names\*\*: Server certificates might use version-specific naming conventions  ### robots.txt or sitemap.xml - \*\*Robots.txt\*\*: May reveal version-specific paths or admin interface locations - \*\*Sitemap.xml\*\*: Could expose admin interface URLs with version information  ### Raw Socket/Banner Grabs - \*\*Binary Protocol Port (6650)\*\*: Pulsar's binary protocol typically runs on port 6650 and may return version information during connection handshake - \*\*Admin HTTP Port (8080)\*\*: Default admin API port that may return server banners - \*\*WebSocket Port\*\*: If WebSocket proxy is enabled, connection handshakes may reveal version details - \*\*BookKeeper Ports\*\*: BookKeeper components may expose version information on their respective ports  ### Custom Protocol Detection - \*\*Pulsar Binary Protocol\*\*: Pulsar uses a custom binary protocol for client communication on port 6650 - \*\*BookKeeper Protocol\*\*: BookKeeper uses its own protocol that may reveal version information - \*\*WebSocket Protocol\*\*: WebSocket API provides simple interaction mechanism that might include version headers  ### Java-Specific Techniques \*\*Java techniques ARE applicable\*\* for Apache Pulsar since it's written in Java: - \*\*JMX Endpoints\*\*: Java Management Extensions may expose version information - \*\*Java System Properties\*\*: Error messages or debug output may reveal Java-specific version details - \*\*JAR File Manifests\*\*: Stack traces often reveal JAR file names with embedded version numbers - \*\*Java Security Manager Errors\*\*: Security-related errors may expose Java-specific version information  ## 4. Indirect or Fingerprintable Signs of Version/Presence  ### Static Asset Naming/Hash Patterns - \*\*JavaScript File Hashes\*\*: Admin console JS files may use content-based hashing that changes between versions - \*\*CSS Versioning\*\*: Stylesheet versions that correspond to specific Pulsar releases - \*\*Image Asset Versioning\*\*: Logo or icon files that change between major versions  ### Unique HTML Structures - \*\*Admin Console Layout\*\*: Different versions of Pulsar Manager or other UIs may have distinct HTML structures - \*\*Error Page Formats\*\*: Version-specific error page layouts and styling - \*\*Form Field Names\*\*: Admin interface forms may use version-specific field naming conventions  ### Field Names in API Responses - \*\*Configuration Parameters\*\*: API responses may include version-specific configuration parameters like "brokerShutdownTimeoutMs" - \*\*Metrics Field Names\*\*: Prometheus metrics may include version-specific metric names - \*\*Status Response Fields\*\*: Health check and status endpoints may return version-specific JSON fields  ### Plugin Paths - \*\*Function Worker Paths\*\*: Pulsar Functions may expose version-specific plugin directory paths - \*\*Connector Paths\*\*: I/O connector paths that vary between versions - \*\*Authentication Plugin Paths\*\*: Security plugin paths that indicate version capabilities  ### Stack Traces or Internal File References - \*\*Java Package Paths\*\*: Stack traces revealing internal package structures that change between versions - \*\*Configuration File Paths\*\*: Error messages exposing version-specific configuration file locations - \*\*Log File References\*\*: Internal references to log files with version-specific naming  ### Error Page Structure - \*\*HTTP Status Code Formats\*\*: Version-specific error response formatting - \*\*JSON Error Structures\*\*: API error responses with version-dependent field arrangements - \*\*Exception Class Names\*\*: Java exception types that may be version-specific  ### Response Headers or Clues - \*\*Custom HTTP Headers\*\*: Pulsar-specific headers that might include version information - \*\*API Version Headers\*\*: REST API versions like "v2" in endpoint paths that indicate compatibility levels - \*\*Content-Type Headers\*\*: Version-specific content type handling - \*\*Cache Control Headers\*\*: Caching strategies that may be version-dependent  ## 5. Summary Table  | Detection Vector | Endpoint/Method | Expected Output/Signature | |------------------|-----------------|---------------------------| | \*\*Health Check API\*\* | `GET /admin/v2/brokers/health` | `"ok"` or JSON status with potential version info | | \*\*Metrics Endpoint\*\* | `GET /metrics` | Prometheus format metrics, may include version labels | | \*\*Broker Config\*\* | `GET /admin/v2/brokers/configuration` | JSON configuration with version-specific parameters | | \*\*Binary Protocol\*\* | TCP connection to port 6650 | Protocol handshake may reveal version information | | \*\*Admin API Root\*\* | `GET /admin/` | May redirect or return version-specific API documentation | | \*\*Error Responses\*\* | Invalid API calls | Stack traces with JAR versions like `pulsar-broker-2.8.0.jar` | | \*\*WebSocket Handshake\*\* | WebSocket connection (if enabled) | Protocol upgrade headers with potential version info | | \*\*Static Assets\*\* | `/assets/`, `/static/`, `/ui/` | JavaScript/CSS files with version-specific content | | \*\*Schema Registry\*\* | `GET /admin/v2/schemas/\*` | API responses with version-specific field structures | | \*\*Cluster Info\*\* | `GET /admin/v2/clusters` | Cluster configuration that may include version metadata | | \*\*BookKeeper Ports\*\* | TCP connections to bookie ports | BookKeeper-specific version banners | | \*\*TLS Certificates\*\* | SSL/TLS handshake | Certificate metadata with potential version clues |  \*\*Note\*\*: Version detection success depends on configuration, security settings, and whether administrative interfaces are enabled and accessible. Many detection methods may require the target Pulsar installation to have default configurations and accessible admin APIs. |

## Product: Superset

|  |  |
| --- | --- |
| Category | Web Application |
| Description | # Apache Superset Version Detection Guide  ## 1. What is this product used for?  Apache Superset is a modern, open-source data exploration and visualization platform. It helps organizations visualize and interact with their data through interactive dashboards, charts, and reports. Superset can replace or augment proprietary business intelligence tools, providing everything from simple line charts to complex geospatial visualizations.  ## 2. What is the type of this software?  \*\*Web Application\*\*  Apache Superset is a web application built on Python Flask for the backend with a React.js frontend. It runs as a Flask WSGI application and provides both a web interface and REST API endpoints for data visualization and business intelligence functionality.  ## 3. Remote Version Detection Methods (No Credentials Required)  ### HTTP Headers - \*\*Server Response Headers\*\*: Flask application headers that may include framework version information - \*\*X-Powered-By Headers\*\*: Flask or custom headers that might reveal version details - \*\*Content-Type Headers\*\*: Application-specific content handling that could vary between versions - \*\*API Response Headers\*\*: REST API endpoints may include version-specific response headers  ### HTML Body Detection - \*\*Meta Tags\*\*: HTML meta tags in the web interface containing version or build information - \*\*Footer Information\*\*: Version details displayed in page footers or about sections - \*\*JavaScript Variables\*\*: Client-side JavaScript that might expose version constants or build hashes - \*\*Error Pages\*\*: Flask error pages that may reveal framework and application version information - \*\*Login Page\*\*: Authentication interface that might include version details in HTML source  ### API Endpoints - \*\*Health Check Endpoint\*\*: `/health` endpoint returns 200 response with "OK" if webserver is running - \*\*Swagger Documentation\*\*: `/swagger/v1` interactive API documentation that may expose version information - \*\*API Base Routes\*\*: `/api/v1/\*` REST API endpoints following OpenAPI specification - \*\*Security Login\*\*: `/api/v1/security/login` authentication endpoint that may include version metadata - \*\*Dashboard API\*\*: `/api/v1/dashboard/` endpoints that could reveal version-specific capabilities - \*\*Chart API\*\*: `/api/v1/chart/` endpoints with version-dependent response structures - \*\*Database API\*\*: `/api/v1/database/` database connection endpoints - \*\*Available Engines\*\*: `/available` endpoint that shows supported database engines  ### Static Asset Paths (JS/CSS) - \*\*Webpack Bundle Names\*\*: React.js frontend assets with content-based hashing (e.g., `static/assets/[hash].js`) - \*\*CSS Stylesheets\*\*: Version-specific styling with build numbers or content hashes - \*\*JavaScript Chunks\*\*: Code-split React bundles with version-specific naming patterns - \*\*Font Files\*\*: Web fonts served from static asset directories - \*\*Image Assets\*\*: Logo files, icons, and other static images that might change between versions  ### Error Messages or Stack Traces - \*\*Flask Runtime Errors\*\*: Python Flask error messages with version-specific package information - \*\*SQLAlchemy Errors\*\*: Database connection errors revealing version-specific database handling - \*\*React.js Console Errors\*\*: Frontend JavaScript errors that might include build or version information - \*\*API Error Responses\*\*: JSON error responses with version-dependent field structures and Flask-AppBuilder details  ### TLS Certificate Clues - \*\*Certificate Subject/Issuer\*\*: TLS certificates that might contain organization-specific version information - \*\*Certificate Extensions\*\*: Custom certificate fields that could indicate deployment version - \*\*Certificate Common Names\*\*: Version-specific certificate naming conventions  ### robots.txt or sitemap.xml - \*\*Robots.txt\*\*: May reveal version-specific admin paths or restricted API endpoints - \*\*Sitemap.xml\*\*: Could expose version-specific URL structures for dashboards or charts - \*\*Security.txt\*\*: Modern security disclosure files that might include version information  ### Raw Socket/Banner Grabs - \*\*HTTP Port Banner\*\*: Default port 8088 (development) or deployed port HTTP server banner - \*\*Gunicorn Server\*\*: Production deployments typically use Gunicorn WSGI server which may expose version - \*\*TLS Handshake\*\*: SSL/TLS connection details that might reveal server version information  ### Custom Protocol Detection - \*\*REST API Protocol\*\*: JSON-based OpenAPI-compliant REST API with version-specific patterns - \*\*WebSocket Endpoints\*\*: Real-time dashboard updates that might include version headers - \*\*Flask-SocketIO\*\*: Real-time communication protocols that might reveal version compatibility  ### Python-Specific Techniques \*\*Python techniques ARE applicable\*\* for Apache Superset since it's built with Python Flask: - \*\*Flask Framework Detection\*\*: Flask-specific error messages and stack traces - \*\*SQLAlchemy ORM Signatures\*\*: Database abstraction layer version information - \*\*Flask-AppBuilder Traces\*\*: Web framework stack traces with version details - \*\*Python Package Versions\*\*: Error messages or debug output revealing specific package versions - \*\*Werkzeug Server Info\*\*: Development server information that might include version details  ## 4. Indirect or Fingerprintable Signs of Version/Presence  ### Static Asset Naming/Hash Patterns - \*\*Webpack Content Hashes\*\*: Frontend assets with SHA-based content hashing that changes between builds - \*\*React.js Build Artifacts\*\*: Build tool generated asset names with version-specific patterns - \*\*CSS Class Names\*\*: Styled-components or CSS modules with version-specific naming patterns - \*\*Source Map Files\*\*: Development artifacts that might expose version and build information  ### Unique HTML Structures - \*\*React Component Architecture\*\*: Component hierarchies and DOM structures that evolve between versions - \*\*Flask-AppBuilder Templates\*\*: Template structures specific to certain FAB versions - \*\*Dashboard Layout Patterns\*\*: Grid systems and layout components unique to specific versions - \*\*Form Field Arrangements\*\*: UI form layouts and field structures that change between releases  ### Field Names in API Responses - \*\*Dashboard Object Fields\*\*: JSON response fields for dashboard metadata that might be version-specific - \*\*Chart Configuration Fields\*\*: Chart definition structures that evolve over time - \*\*Database Connection Fields\*\*: Database configuration parameters exposed through API responses - \*\*User Object Fields\*\*: User management fields that indicate Flask-AppBuilder version capabilities - \*\*Security Permission Fields\*\*: RBAC permission structures that change between versions  ### Plugin Paths - \*\*Visualization Plugin Paths\*\*: Plugin system paths revealing version-specific plugin architecture - \*\*Database Driver Paths\*\*: SQLAlchemy driver references that might indicate version capabilities - \*\*Custom Visualization Assets\*\*: Plugin-specific static assets with version-dependent structures  ### Stack Traces or Internal File References - \*\*Python Package Paths\*\*: Flask, SQLAlchemy, and other package references in error messages - \*\*Source Code References\*\*: Debug information revealing internal file structures and versions - \*\*Configuration File Paths\*\*: References to version-specific configuration file locations (superset\_config.py) - \*\*Database Schema References\*\*: Error messages exposing version-specific database schema structures  ### Error Page Structure - \*\*Flask Error Pages\*\*: Standard Flask error page designs that change between Flask versions - \*\*JSON API Error Formats\*\*: REST API error response structures that evolve over time - \*\*React.js Error Boundaries\*\*: Frontend error handling patterns specific to certain React versions - \*\*Authentication Error Pages\*\*: Flask-AppBuilder authentication error formats  ### Response Headers or Clues - \*\*Flask Framework Headers\*\*: Flask-specific HTTP headers that might include version information - \*\*CORS Headers\*\*: Cross-origin resource sharing configuration that could indicate version - \*\*Security Headers\*\*: CSP, HSTS, and other security headers that might change between versions - \*\*Cache Control Headers\*\*: Caching strategies that may be version-dependent - \*\*Session Management Headers\*\*: Cookie handling that might reveal Flask-AppBuilder version  ## 5. Summary Table  | Detection Vector | Endpoint/Method | Expected Output/Signature | |------------------|-----------------|---------------------------| | \*\*Health Check Endpoint\*\* | `GET /health` | `200 OK` response with `"OK"` body if webserver running | | \*\*Swagger Documentation\*\* | `GET /swagger/v1` | Interactive OpenAPI documentation with potential version info | | \*\*API Base Route\*\* | `GET /api/v1/` | REST API base endpoint with version-specific capabilities | | \*\*Login Endpoint\*\* | `GET /login/` | Flask-AppBuilder authentication interface | | \*\*Static Assets\*\* | `GET /static/assets/\*` | Webpack-built React.js bundles with content hashes | | \*\*Security Login API\*\* | `POST /api/v1/security/login` | Authentication API with version-specific response format | | \*\*Dashboard List API\*\* | `GET /api/v1/dashboard/` | Dashboard metadata with version-dependent field structures | | \*\*Chart List API\*\* | `GET /api/v1/chart/` | Chart definitions revealing version-specific capabilities | | \*\*Database API\*\* | `GET /api/v1/database/` | Database connection metadata with version info | | \*\*Available Engines\*\* | `GET /available` | Supported database engines list | | \*\*Error Responses\*\* | Invalid API calls | Flask/Python stack traces with package version information | | \*\*Default Home Page\*\* | `GET /` | Superset main interface with potential version details | | \*\*Admin Interface\*\* | `GET /admin/` | Flask-AppBuilder admin interface | | \*\*Favicon\*\* | `GET /favicon.ico` | Version-specific favicon file |  \*\*Note\*\*: Version detection success depends on configuration settings, whether debug mode is enabled, and specific deployment methods. The `/health` endpoint is commonly used by load balancers and should be accessible without authentication. Development servers (port 8088) may expose more debugging information than production deployments using Gunicorn or other WSGI servers. Flask applications often expose more version information in development mode than in production configurations. |

## Product: Spark

|  |  |
| --- | --- |
| Category | Service Application |
| Description | # Apache Spark Version Detection Guide  ## 1. What is this product used for?  Apache Spark is a multi-language engine for executing data engineering, data science, and machine learning on single-node machines or clusters. It provides a unified analytics engine for large-scale data processing with built-in modules for streaming, SQL, machine learning, and graph processing. Spark processes both batch and real-time data across distributed computing environments.  ## 2. What is the type of this software?  \*\*Service Application\*\*  Apache Spark is a distributed computing framework that runs as a service application across clusters of machines. It consists of multiple components including the Spark driver, executors, cluster manager, and web UI servers. It's designed to run on various cluster managers (Standalone, YARN, Kubernetes, Mesos) and provides both interactive and batch processing capabilities.  ## 3. Remote Version Detection Methods (No Credentials Required)  ### HTTP Headers - \*\*Web UI Server Headers\*\*: Spark driver web UI typically on port 4040 with server identification headers - \*\*History Server Headers\*\*: Spark History Server on port 18080 with version-specific response headers - \*\*Jetty Server Headers\*\*: Embedded Jetty server headers that may include version information - \*\*API Response Headers\*\*: REST API endpoints that might include Spark version in custom headers  ### HTML Body Detection - \*\*Web UI Interface\*\*: Driver web UI at port 4040 containing version information in HTML content - \*\*Environment Tab\*\*: Web UI Environment tab displaying Runtime Information including Java and Scala versions - \*\*History Server Interface\*\*: History Server UI at port 18080 with potential version details - \*\*Error Pages\*\*: Spark web UI error pages that may reveal framework version information - \*\*Job Details Pages\*\*: Application details pages that might include version metadata  ### API Endpoints - \*\*REST API Base\*\*: `/api/v1/` endpoints for both running applications and History Server - \*\*Applications Endpoint\*\*: `/api/v1/applications` providing application metadata with potential version info - \*\*Jobs Endpoint\*\*: `/api/v1/applications/{app-id}/jobs` with job execution details - \*\*Stages Endpoint\*\*: `/api/v1/applications/{app-id}/stages` revealing version-specific stage information - \*\*Executors Endpoint\*\*: `/api/v1/applications/{app-id}/executors` with executor runtime details - \*\*Environment Endpoint\*\*: `/api/v1/applications/{app-id}/environment` exposing Spark properties and versions - \*\*Storage Endpoint\*\*: `/api/v1/applications/{app-id}/storage/rdd` with RDD storage information  ### Static Asset Paths (JS/CSS) - \*\*Web UI Assets\*\*: JavaScript and CSS files served by the Spark web UI with version-specific content - \*\*Jetty Static Resources\*\*: Static resources served by embedded Jetty server - \*\*History Server Assets\*\*: Static files served by the History Server interface - \*\*Bootstrap/jQuery Libraries\*\*: Third-party library versions that might correlate with Spark versions  ### Error Messages or Stack Traces - \*\*Java Stack Traces\*\*: Comprehensive Java stack traces revealing specific Spark package versions and JAR files - \*\*Scala Runtime Errors\*\*: Scala-specific error messages that might include version information - \*\*Serialization Errors\*\*: ClassNotFoundException or serialization errors exposing internal class versions - \*\*Driver Log Messages\*\*: Driver process logs containing initialization messages with version details - \*\*Executor Failures\*\*: Executor failure messages that might reveal version-specific class paths  ### TLS Certificate Clues - \*\*Self-Signed Certificates\*\*: Default certificates that might include organizational version information - \*\*Certificate Subject Names\*\*: Certificate details that could indicate Spark deployment version - \*\*Certificate Validity Periods\*\*: Certificate dates that might correlate with specific Spark releases  ### robots.txt or sitemap.xml - \*\*Web UI Robots\*\*: May reveal restricted paths or admin interfaces specific to certain versions - \*\*History Server Robots\*\*: Could expose version-specific URL patterns or endpoints  ### Raw Socket/Banner Grabs - \*\*Driver Port (4040)\*\*: HTTP banner information from the Spark driver web UI - \*\*History Server Port (18080)\*\*: Banner information from the History Server - \*\*Spark Master Port (8080)\*\*: Standalone cluster master web UI (if accessible) - \*\*Worker Web UI Ports\*\*: Worker node web interfaces that might expose version information  ### Custom Protocol Detection - \*\*Spark RPC Protocol\*\*: Internal RPC communication that might reveal version compatibility information - \*\*Cluster Manager Communication\*\*: Communication with YARN, Kubernetes, or Mesos that could expose version details - \*\*Driver-Executor Communication\*\*: Internal Spark communication protocols with version handshakes  ### Java-Specific Techniques \*\*Java techniques ARE highly applicable\*\* for Apache Spark since it's built primarily on Java/Scala: - \*\*JAR File Manifests\*\*: Stack traces often reveal JAR files with embedded version numbers (e.g., `spark-core\_2.12-3.4.0.jar`) - \*\*JVM System Properties\*\*: Java system properties exposed through Environment tab or API endpoints - \*\*Java Package Versions\*\*: Exception stack traces revealing specific package versions and dependencies - \*\*Scala Version Detection\*\*: Scala-specific version information in class names and packages - \*\*JMX Endpoints\*\*: Java Management Extensions that might expose version and runtime information  ## 4. Indirect or Fingerprintable Signs of Version/Presence  ### Static Asset Naming/Hash Patterns - \*\*Web UI JavaScript\*\*: Version-specific JavaScript files and their content hashes - \*\*CSS Stylesheet Versions\*\*: Styling files that change between Spark UI versions - \*\*Image Asset Versions\*\*: Icons and images in web UI that might be version-specific - \*\*Font Files\*\*: Web fonts served by the UI that could indicate version changes  ### Unique HTML Structures - \*\*Web UI Layout\*\*: HTML structure patterns unique to specific Spark versions - \*\*Table Formatting\*\*: Job, stage, and executor table layouts that evolve between versions - \*\*Navigation Elements\*\*: Menu structures and navigation patterns specific to versions |

## Product: Log4Net

|  |  |
| --- | --- |
| Category | Library |
| Description | # Apache Log4Net - Version Detection Information  ## 1. What is this product used for?  Apache log4net is a .NET logging library that provides flexible logging capabilities for .NET applications. It allows developers to add comprehensive logging functionality to their applications with configurable output formats, multiple logging levels, and various output destinations (files, databases, console, etc.).  ## 2. What is the type of this software?  \*\*Library\*\*  Log4net is a logging library that gets embedded into .NET applications rather than being a standalone application or service.  ## 3. Remote Version Detection Methods (No Credentials Required)  ### HTTP Headers - \*\*X-Powered-By\*\*: May reveal .NET framework version but not log4net specifically - \*\*Server\*\*: IIS server headers might indicate .NET but not log4net version - \*\*Custom logging headers\*\*: Some applications might add custom headers that reference log4net  ### HTML Body - \*\*Error pages\*\*: ASP.NET error pages may contain stack traces mentioning log4net - \*\*Debug pages\*\*: Development environments might expose log4net configuration details - \*\*Comments\*\*: Rarely, HTML comments might reference log4net version in development builds  ### API Endpoints - \*\*Logging endpoints\*\*: Some applications expose `/logs` or `/api/logs` endpoints that might reveal log4net usage - \*\*Health check endpoints\*\*: May include logging subsystem information - \*\*Admin/debug endpoints\*\*: Could expose log4net configuration or version details  ### Static Asset Paths - \*\*Not applicable\*\*: Log4net doesn't typically serve static assets via web paths - \*\*Configuration files\*\*: Accidentally exposed `log4net.config` files might reveal version info  ### Error Messages or Stack Traces - \*\*ASP.NET error pages\*\*: Stack traces showing `log4net.dll` with version numbers - \*\*Exception details\*\*: Logging-related exceptions that expose log4net assembly information - \*\*Custom error handlers\*\*: Applications using log4net for error logging might expose version info  ### TLS Certificate Clues - \*\*Not applicable\*\*: Log4net doesn't influence TLS certificates  ### robots.txt or sitemap.xml - \*\*Not applicable\*\*: Log4net doesn't typically affect these files  ### Raw Socket/Banner Grabs - \*\*Not applicable\*\*: Log4net is a library, not a network service  ### Custom Protocol - \*\*Not applicable\*\*: Log4net doesn't implement custom network protocols  ### Java-specific Techniques - \*\*Not applicable\*\*: Log4net is a .NET library, not Java. Java-specific techniques do not apply.  ## 4. Indirect or Fingerprintable Signs of Version/Presence  ### Static Asset Naming/Hash Patterns - Log4net doesn't serve static assets, so this doesn't apply  ### Unique HTML Structures - Error page formats that indicate log4net-formatted error logging - Specific timestamp formats in logged error messages displayed to users  ### Field Names in API Responses - Log entries in API responses with log4net-specific field names like:  - `%timestamp`  - `%thread`  - `%logger`  - `%level`  - `%message`  ### Plugin Paths - References to log4net appenders in configuration endpoints - Assembly loading paths mentioning specific log4net versions  ### Stack Traces or Internal File References - Assembly version information: `log4net, Version=X.X.X.X` - File paths: `log4net.dll`, `log4net.config` - Method names from log4net namespace: `log4net.Core.\*`, `log4net.Appender.\*`  ### Error Page Structure - .NET error pages with log4net in the stack trace - Custom error pages that use log4net formatting patterns  ### Response Headers or Clues - Custom application headers that might include build information referencing log4net - Timing headers that might correlate with log4net performance characteristics  ## 5. Summary Table  | Detection Vector | Signature/Output Example | Reliability | |------------------|---------------------------|-------------| | Stack Traces | `log4net, Version=2.0.12.0, Culture=neutral` | High | | Error Pages | `.NET exception with log4net.dll references` | High | | API Responses | JSON logs with log4net field patterns | Medium | | Configuration Exposure | `log4net.config` file accessible via web | High | | Assembly Information | `log4net.Core.LoggingEvent` in stack traces | High | | Custom Headers | Application-specific headers mentioning log4net | Low | | Debug Endpoints | `/debug` or `/health` showing logging subsystem info | Medium | | Exception Messages | Logging-related errors exposing log4net classes | Medium |  \*\*Note\*\*: Since log4net is a library embedded in applications, direct version detection is primarily possible through error conditions, misconfigurations, or debug interfaces that expose internal application details. |

## Product: Nifi

|  |  |
| --- | --- |
| Category | Web Application |
| Description | # Apache NiFi Version Detection and Fingerprinting Guide  ## 1. What is this product used for?  Apache NiFi is an easy to use, powerful, and reliable system to process and distribute data. Apache NiFi is a dataflow system based on the concepts of flow-based programming. It supports powerful and scalable directed graphs of data routing, transformation, and system mediation logic. It provides a web-based user interface for design, control, feedback, and monitoring of dataflows with configurable quality of service dimensions.  ## 2. What is the type of this software?  \*\*Web Application\*\*  NiFi includes an embedded Jetty web server and provides both direct user interaction and external integration through an extensive REST API. It runs as a Java-based web application with a browser-accessible interface and comprehensive REST API.  ## 3. Remote Version Detection Methods (No Credentials Required)  ### HTTP Headers - \*\*Server Response Headers\*\*: NiFi uses an embedded Jetty server which may expose server information in HTTP response headers - \*\*X-Frame-Options\*\*: NiFi 1.8.0 consistently applies security headers including X-Frame-Options - \*\*CSRF Headers\*\*: NiFi 1.15.0 introduced Cross-Site Request Forgery protection with custom HTTP headers on requests  ### HTML Body Detection - \*\*Meta Tags\*\*: Look for NiFi-specific meta tags or references in the HTML source - \*\*JavaScript References\*\*: Check for NiFi-specific JavaScript files and libraries - \*\*Error Pages\*\*: The message-page.jsp error page contains specific NiFi error handling structures  ### API Endpoints (Unauthenticated) - \*\*`/nifi-api/access/config`\*\*: Returns access configuration information including whether NiFi supports user login - \*\*`/nifi-api/system-diagnostics`\*\*: Provides system metrics and diagnostic information - \*\*`/nifi-api/flow/config`\*\*: Returns flow configuration details including timezone and other system information - \*\*`/nifi-api/about`\*\*: Contains version information, build details, and system metadata  ### Static Asset Paths - \*\*`/nifi/`\*\*: Default web context path for NiFi UI - \*\*CSS/JS Assets\*\*: NiFi-specific stylesheet and JavaScript file patterns - \*\*Resource Bundles\*\*: NAR (NiFi Archive) files and extension bundles  ### Error Messages and Stack Traces - NiFi error responses include specific patterns and may contain version-specific information in stack traces - \*\*404 Error Structure\*\*: NiFi has specific 404 error handling that may reveal system information - \*\*Validation Errors\*\*: API validation errors contain NiFi-specific error messages and structures  ### TLS Certificate Clues - Default certificate common names or organizational units that identify NiFi installations - Self-signed certificates with NiFi-specific details  ### robots.txt or sitemap.xml - May contain NiFi-specific paths or API endpoints - Could reveal the `/nifi-api/` structure  ### Raw Socket/Banner Grabs - \*\*Port 8080 (HTTP)\*\*: Default HTTP port for NiFi web interface - \*\*Site-to-Site Ports\*\*: Socket Port and HTTP(S) Port for Remote Transfers of Flow Files - \*\*Cluster Communication\*\*: NiFi cluster nodes communicate via specific ports  ### Custom Protocol (Site-to-Site) - NiFi supports Site-to-Site protocol for remote data transfer with specific handshake patterns - \*\*Remote Process Groups\*\*: Can be probed for version information during site-to-site communication  ### Java-Specific Techniques \*\*Yes, Java-specific techniques are highly applicable:\*\* - \*\*JAR Manifest Information\*\*: NiFi extensions and libraries contain manifest files with version details - \*\*JMX (Java Management Extensions)\*\*: NiFi exposes JMX metrics that may contain version and build information - \*\*Class Loading Information\*\*: Bundle and class loader diagnostics available through API - \*\*Memory and GC Patterns\*\*: Java garbage collection and memory usage patterns specific to NiFi  ## 4. Indirect or Fingerprintable Signs of Version/Presence  ### Static Asset Naming/Hash Patterns - NiFi uses versioned CSS and JavaScript files with specific naming conventions - Build-specific hash patterns in static resource URLs - Extension bundles (NAR files) follow specific versioning patterns like "nifi-standard-nar 1.28.0"  ### Unique HTML Structures - NiFi's Angular-based UI has distinctive DOM structures and element IDs - Component-specific HTML patterns for processors, connections, and process groups - Canvas and flow visualization elements with unique styling  ### Field Names in API Responses - Distinctive field names in JSON responses like "versionedComponentId", "bulletinLevel", "flowFileExpiration" - \*\*Component Types\*\*: Specific values like "PROCESSOR", "CONTROLLER\_SERVICE", "REMOTE\_INPUT\_PORT" - \*\*State Values\*\*: NiFi-specific states like "ENABLED", "DISABLED", "RUNNING", "STOPPED"  ### Plugin and Extension Paths - \*\*NAR Directory Structure\*\*: NiFi uses NAR (NiFi Archive) format for extensions - \*\*Processor Categories\*\*: Specific processor classes like "org.apache.nifi.processors.standard" - \*\*Controller Service APIs\*\*: Service implementations following NiFi naming conventions  ### Stack Traces and Internal References - Stack traces reveal internal NiFi package structures and class hierarchies - \*\*File Path References\*\*: Internal paths like `/opt/nifi/` or Windows equivalent paths - \*\*Java Package Names\*\*: References to `org.apache.nifi.\*` packages  ### Error Page Structure - NiFi has specific error page templates and messaging patterns - \*\*Bulletin System\*\*: Unique bulletin and notification structures - \*\*Validation Error Format\*\*: Consistent error message formatting across components  ### Response Headers and Patterns - \*\*Content-Type Patterns\*\*: Specific MIME types for NiFi-generated content - \*\*Custom Headers\*\*: X-ProxiedEntitiesChain and other NiFi-specific headers - \*\*Session Cookie Names\*\*: NiFi uses specific cookie naming conventions for session management  ### Timing and Behavioral Patterns - \*\*API Response Times\*\*: Characteristic response patterns for different endpoints - \*\*Resource Loading Sequence\*\*: Specific order of CSS/JS resource loading - \*\*WebSocket Connections\*\*: NiFi supports WebSocket connections with specific handshake patterns  ## 5. Summary Table  | Detection Vector | Signature/Output | Confidence | Notes | |-----------------|------------------|------------|-------| | `/nifi-api/about` | `{"about":{"version":"X.X.X","buildTag":"nifi-X.X.X"}}` | \*\*High\*\* | Direct version disclosure | | `/nifi-api/access/config` | `{"config":{"supportsLogin":true}}` | \*\*High\*\* | Confirms NiFi presence | | `/nifi-api/system-diagnostics` | JSON with heap, threads, repositories data | \*\*High\*\* | NiFi-specific system metrics | | Server Headers | `Server: Jetty/X.X.X` | \*\*Medium\*\* | Indicates embedded Jetty (NiFi uses this) | | Error Pages | NiFi-specific error templates with bulletin references | \*\*Medium\*\* | Version-specific error handling | | CSS/JS Paths | `/nifi/css/`, `/nifi/js/` with build hashes | \*\*Medium\*\* | Static asset patterns | | API Field Names | `versionedComponentId`, `bulletinLevel`, `flowFileExpiration` | \*\*High\*\* | NiFi-specific JSON fields | | Component Types | `PROCESSOR`, `CONTROLLER\_SERVICE`, `REMOTE\_INPUT\_PORT` | \*\*High\*\* | NiFi component enumeration | | NAR References | `nifi-standard-nar`, `org.apache.nifi.processors` | \*\*High\*\* | NiFi extension system | | Site-to-Site Protocol | Port probing on 8081/8082 with handshake | \*\*Medium\*\* | NiFi-specific protocol | | Canvas Elements | Angular components with flow visualization | \*\*Medium\*\* | UI-specific fingerprinting | | JMX Endpoints | Java management beans with NiFi metrics | \*\*Low\*\* | Requires JMX access | | Default Ports | 8080 (UI), 8443 (HTTPS), 8081 (Site-to-Site) | \*\*Low\*\* | Common but indicative | | Stack Traces | `org.apache.nifi.\*` package references | \*\*Medium\*\* | Java-specific detection |  ### Version-Specific Indicators  - \*\*NiFi 1.15.0+\*\*: CSRF protection headers and double submit cookie strategy - \*\*NiFi 1.16.0+\*\*: Configurable HTTP request logging support - \*\*NiFi 1.8.0+\*\*: Consistent X-Frame-Options header application - \*\*NiFi 2.0.0+\*\*: Major API version changes and component updates  ### High-Confidence Detection Workflow  1. \*\*Primary Check\*\*: `GET /nifi-api/about` → Direct version information 2. \*\*Secondary Check\*\*: `GET /nifi-api/access/config` → Confirms NiFi presence  3. \*\*Tertiary Check\*\*: `GET /nifi-api/system-diagnostics` → System-specific metrics 4. \*\*Fallback\*\*: Analyze error pages, static assets, and API response patterns for indirect indicators |

# Vendor: ELK

## Product: Elastic Search

|  |  |
| --- | --- |
| Category | Service Application |
| Description | # Elasticsearch by Elastic  ## 1. What is this product used for?  Elasticsearch is a distributed, RESTful search and analytics engine capable of addressing a growing number of use cases. It serves as the core component of the Elastic Stack and is primarily used for full-text search, log analytics, real-time data analysis, and application monitoring.  ## 2. What is the type of this software?  \*\*Service Application\*\* - Elasticsearch runs as a service/daemon that provides RESTful APIs over HTTP and can be deployed as a distributed cluster of nodes.  ## 3. Remote Version Detection Methods (No Credentials Required)  ### HTTP Headers ``` Server: Elasticsearch/7.10.1 X-Elastic-Product: Elasticsearch ```  ### API Endpoints ``` GET / - Main cluster information endpoint Response includes version in JSON: {  "version": {  "number": "7.10.1",  "build\_flavor": "default",  "build\_type": "tar",  "build\_hash": "1c34507e66d7db1211f66f3513706fdf548736aa",  "build\_date": "2020-12-05T01:00:33.671820Z",  "build\_snapshot": false,  "lucene\_version": "8.7.0"  } }  GET /\_cluster/health GET /\_nodes GET /\_cat/nodes?v GET /\_cat/health?v ```  ### HTML Body/Web Interface - Kibana interface may reveal Elasticsearch version - Default landing pages on port 9200 show cluster information - Stack trace errors may contain version information  ### Static Asset Paths ``` /\_plugin/[plugin-name]/ /\_cat/ /\_cluster/ ```  ### Error Messages/Stack Traces ``` ElasticsearchException stack traces Java class paths revealing version: org.elasticsearch.Version Build information in exception messages ```  ### TLS Certificate Clues - Certificate subjects may contain "elasticsearch" or "elastic" - Self-signed certificates with Elasticsearch-specific CNs  ### Raw Socket/Banner Grabs ``` Port 9200 (HTTP) - JSON response with version Port 9300 (Transport) - Binary protocol, may leak version in handshake ```  ### Java-Specific Techniques \*\*Applicable\*\* - Elasticsearch is Java-based: - JMX endpoints may expose version information - Java serialization fingerprinting - JVM version correlation in responses - Java stack traces in error responses  ## 4. Indirect or Fingerprintable Signs of Version/Presence  ### API Response Structure Fingerprinting ``` - Field names and structure changes between versions - "\_score" field presence/format - Mapping types (removed in 7.x+) - Query DSL syntax variations ```  ### Unique Response Headers ``` content-type: application/json; charset=UTF-8 X-content-type-options: nosniff ```  ### Error Page Structure ``` - JSON error format and field names - "error.type" field variations - Stack trace format and Java class paths ```  ### Plugin/Feature Detection ``` - X-Pack features availability - Security plugin responses - Watcher, Graph, ML endpoints availability - SQL plugin endpoint /\_sql/ ```  ### Build Hash Correlation ``` - build\_hash in responses can be correlated to specific versions - Lucene version correlation - build\_date patterns ```  ### Port and Service Patterns ``` - Default ports 9200 (HTTP), 9300 (Transport) - Cluster node discovery responses - Node role identification (master, data, ingest) ```  ## 5. Summary Table  | Detection Vector | Endpoint/Method | Expected Output/Signature | |------------------|-----------------|---------------------------| | \*\*Direct Version\*\* | `GET /` | `{"version":{"number":"7.10.1"}}` | | \*\*HTTP Headers\*\* | Any request | `Server: Elasticsearch/X.Y.Z` | | \*\*Cluster Health\*\* | `GET /\_cluster/health` | Cluster status with version hints | | \*\*Nodes Info\*\* | `GET /\_nodes` | Full node information including versions | | \*\*Cat API\*\* | `GET /\_cat/nodes?v` | Human-readable node information | | \*\*Build Hash\*\* | `GET /` | `"build\_hash": "1c34507e66..."` | | \*\*Lucene Version\*\* | `GET /` | `"lucene\_version": "8.7.0"` | | \*\*Error Responses\*\* | Invalid requests | Java stack traces with version info | | \*\*Plugin Detection\*\* | `GET /\_cat/plugins` | Installed plugins and versions | | \*\*X-Pack Features\*\* | `GET /\_xpack` | License and feature information | | \*\*Transport Port\*\* | Port 9300 scan | Binary protocol handshake | | \*\*Index Mapping\*\* | `GET /index/\_mapping` | Mapping format indicates version range | |

## Product: Logstash

|  |  |
| --- | --- |
| Category | Service Application |
| Description | # Logstash Detection and Fingerprinting Guide  ## 1. What is this product used for?  Logstash is a log aggregator and data processing engine that collects data from various input sources, executes transformations and enhancements, and ships the data to various supported output destinations. It is part of the ELK/Elastic Stack (Elasticsearch, Logstash, Kibana) used for centralized log management, monitoring, troubleshooting, and analytics.  ## 2. What is the type of this software?  \*\*Service Application\*\*  Logstash runs as a service/daemon that continuously processes data pipelines, listening on configured ports for incoming data and forwarding processed logs to output destinations like Elasticsearch.  ## 3. Remote Version Detection Methods (No Credentials Required)  ### HTTP API Endpoints (Port 9600)  By default, Logstash binds to port 9600 for its monitoring API. The root endpoint provides version information:  ```bash curl -XGET 'http://target:9600/?pretty' ```  \*\*Response contains:\*\* - `version`: Exact Logstash version - `host`: Hostname of the server - `http\_address`: API bind address  ### Specific API Endpoints  Additional endpoints for detailed information:  ```bash # Node information (includes version) GET http://target:9600/\_node?pretty  # Plugin information GET http://target:9600/\_node/plugins?pretty  # Statistics GET http://target:9600/\_node/stats?pretty  # Hot threads GET http://target:9600/\_node/hot\_threads?pretty ```  ### Raw Socket/Banner Grabs  \*\*Port 5044 (Beats Input)\*\* Port 5044 is the default port for Logstash Beats input plugin, which accepts encrypted log data from Beats agents.  \*\*Port 5140 (Syslog)\*\* Commonly configured UDP port for syslog input. Can reveal Logstash process information in netstat output.  \*\*Port 514 (Privileged Syslog)\*\* May be configured for syslog input, though often forwarded to higher ports due to privilege requirements.  ### Error Messages and Stack Traces  Logstash error messages and Java stack traces can reveal version information: - JRuby version information in stack traces - Plugin version information in error messages - Logstash-specific error patterns and log formats  ### TLS Certificate Clues  SSL certificates used for Beats communication may contain hostname information or organization details that indicate Logstash deployment.  ### Java-Specific Techniques  \*\*Applicable - Logstash runs on JRuby/JVM:\*\* - JMX endpoints (if exposed) - Java error stack traces revealing JRuby version - JVM memory patterns in stats API - Java process information in system monitoring  ## 4. Indirect or Fingerprintable Signs of Version/Presence  ### HTTP Response Headers and Patterns  API responses contain consistent JSON structure patterns: - Specific JSON field names (`http\_address`, `pipeline`, `jvm`) - Response header patterns - Error message formats  ### Process and Network Signatures  System-level indicators: - Java process running Logstash JAR files - Specific port binding patterns (9600-9700 range) - Memory usage patterns typical of JRuby applications  ### Configuration File References  Security scanners may detect Logstash based on Elasticsearch presence, as they're commonly deployed together in the ELK stack.  ### Log File Patterns and Paths  Common Logstash-specific log entries: - `/var/log/logstash/logstash-plain.log` - `[INFO ][logstash.outputs.elasticsearch]` patterns - `Successfully started Logstash API endpoint` messages  ### Fingerprint Filter Signatures  Logstash uses fingerprinting for deduplication, which may create recognizable hash patterns in downstream systems.  ### Plugin-Specific Indicators  Plugin information can be fingerprinted: - Specific plugin naming conventions (`logstash-codec-\*`, `logstash-filter-\*`) - Plugin version patterns - Default plugin configurations  ## 5. Summary Table  | \*\*Detection Vector\*\* | \*\*Port/Method\*\* | \*\*Signature/Output\*\* | \*\*Reliability\*\* | |---------------------|-----------------|---------------------|-----------------| | HTTP API Root | 9600/TCP | `{"host":"...", "version":"X.Y.Z", "http\_address":"..."}` | \*\*High\*\* | | Node Info API | 9600/TCP | Detailed version, JVM, OS information | \*\*High\*\* | | Plugin Info API | 9600/TCP | List of installed plugins with versions | \*\*High\*\* | | Stats API | 9600/TCP | Runtime statistics and performance metrics | \*\*Medium\*\* | | Hot Threads API | 9600/TCP | Java thread information revealing JRuby | \*\*Medium\*\* | | Beats Input Port | 5044/TCP | Encrypted beat protocol responses | \*\*Low\*\* | | Syslog Input Port | 5140/UDP | Syslog processing behavior | \*\*Low\*\* | | Process Detection | System | Java process with Logstash JARs | \*\*Medium\*\* | | Stack Traces | Logs/Errors | JRuby stack traces with version info | \*\*Medium\*\* | | ELK Co-location | 9200/TCP | Elasticsearch presence suggesting Logstash | \*\*Low\*\* | | SSL Certificates | 5044/TCP | Certificate CN/SAN for Beats communication | \*\*Low\*\* | | Configuration Files | File System | `logstash.yml`, plugin configs | \*\*High\*\* | | Log Patterns | File System | Characteristic log file formats | \*\*Medium\*\* |  \*\*Key Takeaways:\*\* - Port 9600 HTTP API is the most reliable detection method, providing exact version information - API binds to 127.0.0.1 by default but can be configured for external access - Often detected indirectly through Elasticsearch presence in ELK stack deployments - Java/JRuby stack traces are valuable for version fingerprinting when API is not accessible |

## Product: Kibana

|  |  |
| --- | --- |
| Category | Web Application |
| Description | ## ELK Kibana Version Detection Guide: Kibana is a powerful threat hunting and investigation tool when combined with Elasticsearch and log sources. It forms the frontend of many modern SIEM solutions, and helps analysts detect and understand threats faster. "Kibana turns raw security logs into actionable threat intelligence dashboards."  Kibana (Part of the ELK Stack) – Key Highlights:  | \*\*Aspect\*\* | \*\*Details\*\* | | ----------------------------- | --------------------------------------------------------------------------------------------------------------------------------------------------------- | | \*\*What is it?\*\* | \*\*Kibana\*\* is the \*\*visualization and dashboard tool\*\* for data stored in \*\*Elasticsearch\*\*, part of the \*\*ELK Stack\*\* (Elasticsearch, Logstash, Kibana). | | \*\*Primary Use\*\* | Explore, analyze, and visualize \*\*logs, events, and security telemetry data\*\*. | | \*\*Tech Stack\*\* | Built with \*\*Node.js\*\*, \*\*React\*\*, integrates with \*\*Elasticsearch (Lucene-based)\*\* for data search. | | \*\*Security Relevance\*\* | - Central to \*\*SIEM solutions\*\* like Elastic Security | | | - Used to visualize \*\*alerts, IOCs, anomalies, threat patterns\*\* | | | - Supports real-time \*\*log and event analysis\*\* from sources like Suricata, Zeek, Wazuh, Sysmon, etc. | | \*\*Elastic Security\*\* | - Integrated \*\*SIEM module\*\* in Kibana | | | - Threat hunting via \*\*detection rules, timeline analysis, MITRE ATT\&CK mapping\*\* | | | - Built-in \*\*threat intelligence feeds\*\*, correlation, and alerting | | \*\*Authentication/Access\*\* | - Supports \*\*RBAC\*\*, \*\*LDAP\*\*, \*\*SSO (SAML, OIDC)\*\* | | | - \*\*Spaces\*\* allow segregated views for different teams | | \*\*Security Features\*\* | - \*\*Audit logging\*\*, encrypted communications (TLS), \*\*API key management\*\* | | | - Secure saved objects and dashboards | | | - \*\*Alerting & Actions\*\* to trigger external workflows (Slack, PagerDuty, etc.) | | \*\*Deployment Cautions\*\* | - Do not expose Kibana directly to the internet without reverse proxy + auth | | | - \*\*Rate-limit APIs\*\*, monitor for brute-force or enumeration attempts | | | - Patch regularly: Kibana has had \*\*critical CVEs\*\* (e.g., RCEs via plugins/UI) | | \*\*Use Cases in Threat Intel\*\* | - Visualize malware/beaconing patterns | | | - Build dashboards for \*\*TTP monitoring\*\* | | | - Correlate threat data across sources (e.g., SIEM logs + threat feeds) | | | - Serve as \*\*case management + incident timeline tool\*\* | | \*\*Integrations\*\* | - Connects with \*\*Beats\*\*, \*\*Logstash\*\*, \*\*Wazuh\*\*, \*\*Zeek\*\*, \*\*Suricata\*\*, etc. | | | - Can enrich data with \*\*threat intel feeds\*\* (MISP, AbuseIPDB, etc.) | | \*\*Official Site\*\* | [https://www.elastic.co/kibana](https://www.elastic.co/kibana) | |