Automating Trivy vulnerability reporting

End-to-end intelligent automation of security vulnerability reporting using Jenkins and UST SmartOps

ust.com

UST

Table of contents

Introduction

In modern enterprise software delivery pipelines, container security is a non-negotiable operational discipline. Organizations running microservice architectures at scale routinely depend on open-source vulnerability scanning tools such as Trivy to assess the security posture of their container images. While such tools are effective at surfacing vulnerability data, the downstream process of aggregating, classifying, and distributing that data into actionable intelligence has, in many organizations, remained stubbornly manual.

This white paper presents an end-to-end automation solution for Trivy vulnerability reporting, developed as part of an innovation initiative within the UST SmartOps practice. The solution leverages the Jenkins Build Metadata API in conjunction with the UST SmartOps Automation Story framework to fully automate the vulnerability report generation and stakeholder notification lifecycle. The result is a deterministic, standardized, and fully orchestrated reporting pipeline that eliminates manual intervention, reduces operational risk, and ensures executive-ready vulnerability intelligence is delivered automatically on a recurring schedule.

KEY INSIGHT

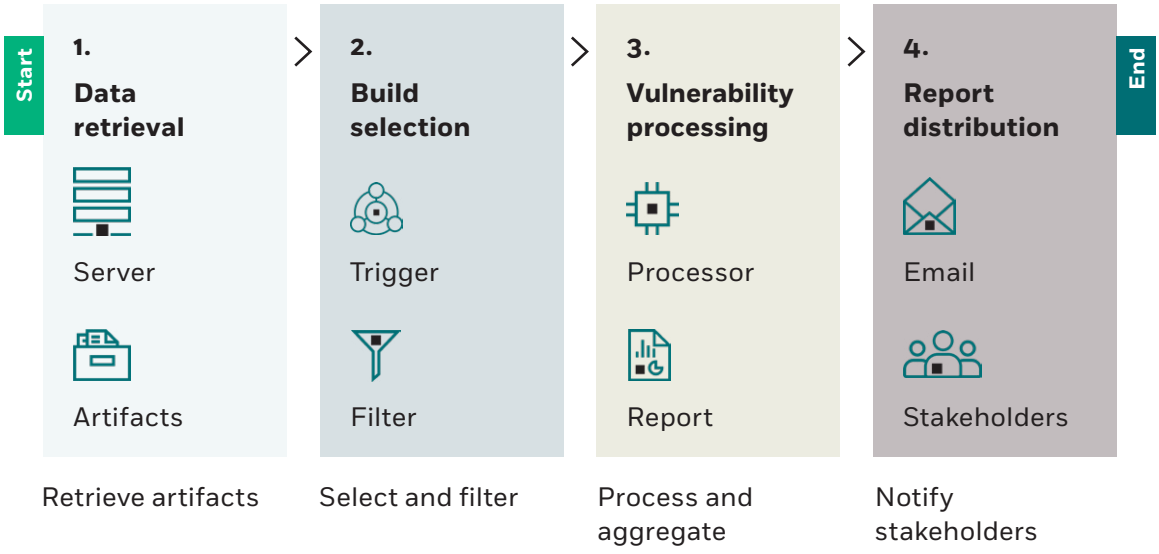
Organizations scanning hundreds of containers across multiple deployment environments generate thousands of vulnerability records per cycle. Without automation, converting this raw scan data into actionable executive reporting requires hours of expert-level manual effort every reporting cycle — effort that introduces error, inconsistency, and operational dependency on a single individual.

The challenge: The business risk of manual vulnerability reporting

The baseline process for generating a Trivy vulnerability report in a multi-environment container deployment comprises seven discrete manual steps, each requiring careful execution and domain expertise:

1	Identify and select the correct Jenkins build artifact for the reporting period.
2	Download the artifact ZIP file for each deployment environment separately.
3	Extract files from nested ZIP archives to access the raw vulnerability JSON outputs.
4	Manually review thousands of individual vulnerability records across all container images.
5	Aggregate CRITICAL and HIGH severity findings, segmented by OS vs. Library type and by environment.
6	Construct the summarized Excel report manually, applying classification and exclusion rules.
7	Distribute the completed report to stakeholders via email.

The core difficulty in this workflow arises not from the scanning process itself, but from the downstream aggregation and classification of the scanner output. The Trivy tool generates machine-readable JSON files that can span thousands of records per environment. Converting this raw output into a structured, stakeholder-ready report requires significant domain knowledge and is highly susceptible to human error at multiple stages.



Key deficiencies in the current approach

A detailed analysis of the manual process reveals five critical deficiencies, each representing a measurable operational risk:

1. Single point of failure

The current workflow is entirely dependent on one subject matter expert who holds the knowledge to select the correct build artifact, apply container classification rules, and execute the exclusion logic for /xd/ containers. If that individual is unavailable, the reporting cycle cannot run. This is a gap in the organization's security governance posture.

2. Calculation errors at scale

Manual summation of vulnerability metrics across hundreds of containers, multiple environments, and two severity levels introduces compounding error risk. A transcription error in a critical count doesn't just produce an inaccurate report; it can misdirect security remediation prioritization, with downstream consequences for compliance and risk management.

3. Inconsistent report structure

Report format varies depending on who runs the process, differences in classification logic, environment block organization, and exclusion presentation. This inconsistency makes cross-period trend analysis unreliable and complicates governance auditing. A security posture report that can't be compared month-over-month has limited strategic value.

4. Manual build selection

Identifying the correct reporting build requires verifying three independent criteria simultaneously: SUCCESS status, correct calendar date (first Monday of the month), and artifact availability. This multi-step manual verification is error-prone and incurs recurring time costs, with no equivalent automation in the current process.

5. No orchestration layer

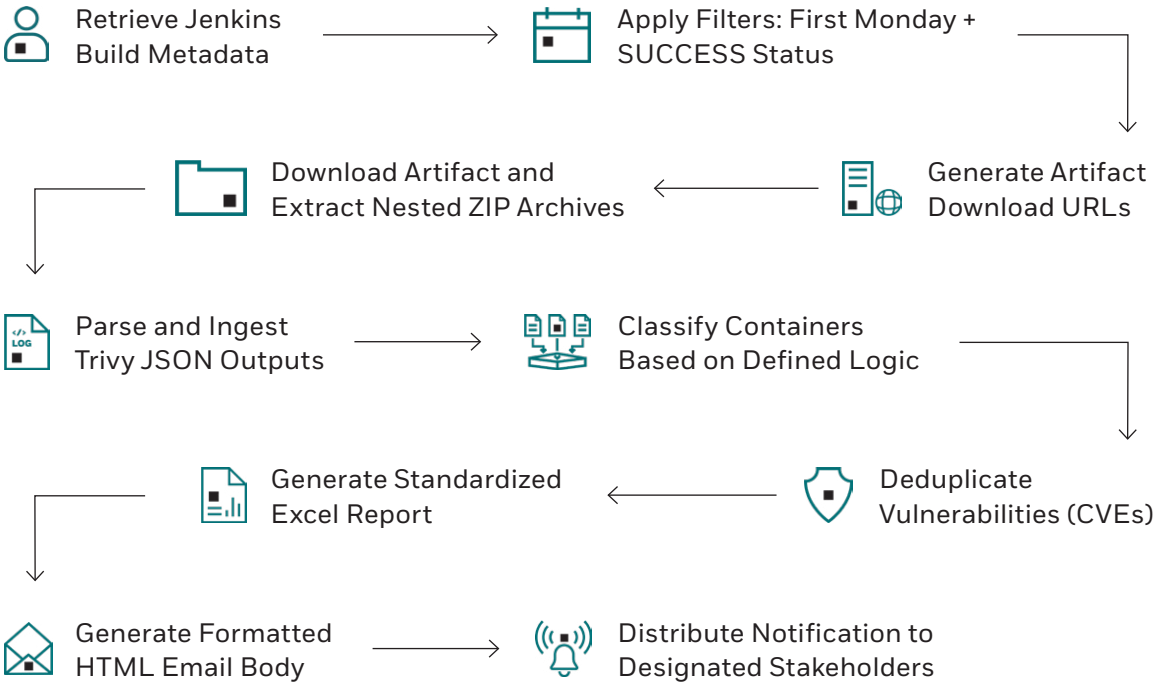
Every step in the current workflow: build selection, artifact retrieval, data processing, report generation, and email distribution is a disconnected manual activity. There is no single execution framework, no state tracking, and no failure management. This architecture cannot scale and cannot be audited end-to-end.

Proposed automated solution

The proposed solution establishes a fully autonomous, end-to-end vulnerability reporting pipeline within the UST SmartOps Automation Story framework. The pipeline comprises ten sequential steps that, once triggered, execute without manual intervention from build selection through stakeholder notification.

STEP 1	Retrieve Jenkins build metadata via the REST API.
STEP 2	Apply deterministic filters: build month + build SUCCESS status.
STEP 3	Programmatically generate artifact download URLs for qualifying builds.
STEP 4	Download and extract nested ZIP archives for all environments.
STEP 5	Parse and ingest Trivy JSON vulnerability outputs.
STEP 6	Classify containers based on defined environment and category keyword logic.
STEP 7	Deduplicate vulnerabilities using a composite key per CVE.logic.
STEP 8	Generate a standardized, fully formatted Excel vulnerability report.
STEP 9	Generate a structured HTML email body mirroring the Excel report.
STEP 10	Distribute the report and notification automatically to all stakeholders.

This architecture eliminates every manual touchpoint in the original seven-step process, replacing subjective human decision-making with deterministic, programmatic logic at each stage. Because the current Jenkins job runs on the first Monday of every month, we apply filters to process only the builds executed on that most recent Monday.



Key system enhancements

Deterministic first-Monday build selection

One of the most operationally significant improvements delivered by this automation is the programmatic identification of the designated monthly reporting build. The build selector component queries the Jenkins REST API and evaluates each build against four explicit criteria:

- Build result equals SUCCESS — ensuring only stable, complete builds are considered.
- Building equals false — excluding any in-progress builds from selection.
- Weekday() equals Monday — restricting selection to builds executed on Mondays.
- Day is less than or equal to 7 — further restricting to the first Monday of the calendar month.

The build timestamp returned by the Jenkins API (epoch milliseconds) is converted to a Python datetime object for calendar evaluation. This four-condition filter guarantees that the correct build is selected on every execution, removing the recurring manual verification burden entirely. These selection criteria are fully configurable and can be adjusted to target any specific weekday or reporting schedule as per requirements.

Fully standardized Excel report generation

The Excel report is generated entirely through structured, deterministic logic, ensuring bit-for-bit consistency in format and calculation across all reporting cycles. The standardized report includes the following structural elements:

- Three distinct environment blocks: AIOps, SV (SmartVision), and Core Platform.
- Category-level segmentation within each environment block.
- Explicit separation of OS-level and Library-level vulnerability counts.
- Discrete counts for CRITICAL and HIGH severity findings per category.
- Computed subtotals, category totals, and environment-level grand totals.
- A consolidated count of excluded /xd/ containers displayed at the base of the summary sheet.
- Uniform formatting: merged headers, consistent cell borders, standardized layout.

Additional detail sheets are generated per environment, enabling drill-down analysis of Library and OS vulnerabilities at the container level. The reproducibility of this report structure enables reliable cross-period security trend analysis for the first time.

HTML email body with Excel-mirrored structure

The automated notification includes a structured HTML email body that dynamically mirrors the Excel summary report. This ensures that stakeholders receive an immediately readable executive summary within the email client itself, without needing to open the Excel attachment. The HTML email body includes:

- Three environment-specific summary tables with full category breakdowns.
- OS vs. Library vulnerability split per category.
- CRITICAL and HIGH subtotals with environment grand totals.
- Consolidated /xd/ exclusion totals.
- Color-coded table headers for rapid visual parsing.
- An auto-generated, standardized system disclaimer.

The HTML structure uses col-span and row-span attributes to replicate the visual hierarchy of the Excel summary, ensuring structural and data consistency between the email body and the attached report. The email subject line is generated dynamically using the first Monday timestamp, ensuring date alignment across all distributed artifacts.

End-to-end workflow orchestration

The entire pipeline is executed through a single centralized Automation Story, eliminating the disconnected, step-by-step manual execution model of the current process. The orchestration layer controls the complete sequence from build metadata retrieval through stakeholder notification, covering:

- Build metadata retrieval and deterministic filtering.
- Authenticated artifact download and nested extraction.
- JSON parsing, vulnerability aggregation, and deduplication.
- Container classification and environment-level summary generation.
- Excel report creation with full formatting.
- HTML email body generation.
- Stakeholder notification via the configured email queue.

Once triggered via the Scheduler, the Automation Story executes autonomously without requiring any manual artifact handling, data aggregation, calculation, or email distribution. This reduces operational risk, improves reliability, and makes the reporting function fully scalable independent of individual availability.

Implementation architecture

The automated workflow is implemented through two primary Python executor components, each with a clearly defined responsibility.

Executor 1: **Build selector**

Queries the Jenkins REST API to identify the correct monthly reporting build. Produces three outputs: artifact download URLs for qualifying builds, the epoch timestamp of the qualifying build (used for subject line date consistency), and a diagnostic JSON array providing full traceability into the build selection decision.

```
/job/{job_name}/api/json?tree=builds[number,result,building,timestamp]
```

Executor 2: **Report generator**

Consumes the artifact URLs from Executor 1 and executes the complete data pipeline: artifact download and extraction, JSON parsing, vulnerability filtering (CRITICAL and HIGH only), deduplication via composite key (environment + container name + CVE ID), container classification, Excel report generation, and HTML email body construction.

Container classification logic

Containers are classified into three environments using keyword matching against the container name. Any container whose name contains the substring /xd/ is categorized as XD_EXCLUDED, removed from all environment totals, and counted separately for audit transparency. The classification mapping is as follows:

Environment	Categories	Keyword match
AIOps	AIOps / Platform	AIOps or Platform
SV (SmartVision)	SV / Platform	SV or Platform
Core Platform	Platform / Infra / Common	Platform, Infra, or Common
XD_EXCLUDED	—	Contains /xd/ substring

Excluded /xd/ containers are removed from all environment totals and counted separately for audit transparency.

Full API endpoint specifications, executor input/output schemas, and email connector configuration are provided in the Technical Appendix.

Email connector

The Email Connector is implemented as the final stage of the Automation Story. A JSON Builder block assembles the complete payload — comprising email_subject, email_body (HTML), and the Base64-encoded Excel attachment — and publishes it to the configured email queue via a Queue Publish block. The email subject line is generated programmatically from the first_monday_timestamp, guaranteeing date alignment between the build selection, the Excel report filename, and the distributed notification.

Error handling and operational resilience

Production-readiness in an automated reporting pipeline is defined not just by what it does when everything works, but by how it behaves when something fails. This solution is designed for fault-tolerant execution across a range of real-world failure scenarios:

Failure Scenario	Behavior
Non-200 HTTP response from Jenkins API	The response is skipped; execution continues with remaining items. No termination occurs.
Artifact download failure for one environment	The failure does not terminate execution. Processing continues with remaining artifact URLs.
No qualifying First Monday builds found	artifact_urls array is empty. A structurally complete empty report is generated with zero counts.
No CRITICAL or HIGH vulnerabilities detected	All summary blocks are populated with zero counts. Structural consistency is fully maintained.
JSON parsing failure for individual file	The file is skipped with appropriate logging. Other files in the archive continue to be processed.

Access and environment prerequisites

Successful execution of the automation requires the following prerequisites to be satisfied within the execution environment:

- Jenkins API access fully enabled for the target instance.
- Artifact access permissions configured for the Jenkins service account.
- Python executor environment includes the requests, openpyxl, and pandas libraries.
- Formal operational ownership defined for execution failure management.
- Container classification keywords maintained and updated as naming conventions evolve.
- Environment mapping updated whenever environment naming conventions change.

Financial outcomes and business impact

The implemented automation delivers five measurable operational outcomes that directly address each of the deficiencies identified in the current manual process:

Outcome	Description
Automatic build selection	The designated First Monday reporting build is identified and selected deterministically on every execution cycle — no manual date verification or artifact validation required.
Standardized report structure	The Excel vulnerability report is generated with an identical structure, layout, and formula logic on every execution, enabling reliable cross-period trend analysis.
Error-free aggregation	Programmatic deduplication and summation eliminate the calculation errors inherent in manual aggregation of thousands of vulnerability records.
Eliminated human dependency	The reporting function is no longer dependent on the availability of a specific subject matter expert. Any authorized trigger can initiate the complete pipeline.
Automatic stakeholder distribution	Vulnerability intelligence is delivered automatically to all configured stakeholders without manual email composition or distribution.

Conclusion

Vulnerability reporting is a foundational pillar of enterprise security governance. It is also, in most organizations, a process that hasn't kept pace with the scale of modern container-based delivery.

The Trivy Vulnerability Report Automation solution presented in this white paper demonstrates what UST SmartOps makes possible: a seven-step manual reporting cycle, dependent on a single expert, prone to calculation error, and impossible to audit end-to-end, replaced by a fully autonomous, deterministic pipeline that runs on schedule, delivers standardized intelligence, and scales without constraint.

For security and engineering leaders managing container workloads at scale, the question is no longer whether vulnerability reporting can be automated. It's how quickly the manual process can be retired.

UST SmartOps can help your organization get there. To see this solution in action or discuss how it applies to your environment, contact the UST SmartOps team at ust.com/smartops.

AUTHOR

Kasinathan V

UST SmartOps | UST

Since 1999, UST has worked side by side with the world's best companies to make a powerful impact through transformation. Powered by technology, driven by AI, inspired by people, and led by our purpose, we partner with our clients from design to operation. Our AI-driven digital solutions, proprietary platforms, engineering, R&D, products, and innovation ecosystem turn core challenges into impactful, disruptive business outcomes. With deep industry knowledge and a future-ready mindset, we infuse expertise, innovation, and agility into our clients' organizations—delivering measurable value and positive lasting change for them, their customers, and communities around the world. Together, with 30,000+ employees in 35+ countries, we build for boundless impact—touching billions of lives in the process.

ust.com

Version 0101-20260619