**Design and Implementation of a Secure Log Analysis,**

**CXOne Vulnerability Mitigation and Automation Framework**

**for Enterprise Applications**

*An Internship Report*

*Submitted in partial fulfillment of the requirements for the*

*award of the Degree of*

**BACHELOR OF TECHNOLOGY**

**IN**

# ELECTRONICS & COMMUNICATION ENGINEERING

BY

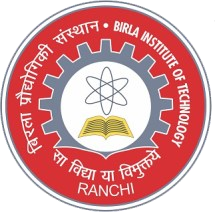
**HONEY PRADHAN**

**(BTECH/10168/21)**

Under the supervision of

**Dr Vijay Nath**

**(Associate Professor, BIT Mesra)**



DEPT. OF ELECTRONICS AND COMMUNICATION ENGINEERING

BIRLA INSTITUTE OF TECHNOLOGY

MESRA-835215, RANCHI  
 2025

### APPROVAL OF THE GUIDE

Recommended that the Project entitled **Design and Implementation of a Secure Log Analysis, CXOne Vulnerability Mitigation and Automation Framework for Enterprise Applications** submitted by **HONEY PRADHAN(BTECH/10168/21)** is approved by me for submission. This should be accepted as fulfilling the partial requirements for tthe award of Degree of Bachelor of Technology in **Electronics and Communication Engineering**. To the best of my knowledge, the content of this report does not form a basis for the award of any previous degree to anyone else.

**Date: Dr Vijay Nath**

**Associate Professor**

**Department of Electronics & Communication Engineering**

**Birla Institute of Technology, Mesra**

### DECLARATION CERTIFICATE

I certify that,

1. The work contained in the report is original and has been done by myself under the general supervision of my supervisor.
2. The work has not been submitted to any other Institute for any other degree or diploma.
3. I have followed the guidelines provided by the Institute in writing the report.
4. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.
5. Whenever I have used materials (data, theoretical analysis, and text) from other sources, I have given due credit to them by citing them in the text of the report and giving their details in the references.
6. Whenever I have quoted written materials from other sources, I have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

**Date: Honey Pradhan**

**BTECH/10168/21**

**Department of Electronics & Communication Engineering**

**Birla Institute of Technology, Mesra**

# CERTIFICATE OF APPROVAL

This is to certify that the work embodied in this thesis entitled **“Design and Implementation of a Secure Log Analysis, CXOne Vulnerability Mitigation and Automation Framework for Enterprise Applications”**, is carried out by **Name of the Student (Roll Number)** has been approved for the degree of Bachelor of Technology of Birla Institute of Technology, Mesra, Ranchi.

Date:

Place:

**Internal Examiner External Examiner**

**(Chairman)**

**Head of Depart**

***ABSTRACT***

#### 

During my internship, significant contributions have been made in the realms of Daily Monitoring, enhancement of cybersecurity measures, and refinement of automation frameworks within enterprise environments. One of the standout accomplishments was the design, development, and packaging of a bespoke Splunk log parser, which functions as a comprehensive end-to-end utility. This utility automates the extraction of valuable insights from extensive system logs, making it indispensable for efficient log analysis. The parser is engineered to automatically download log files from Splunk across all internal applications, which directly eliminates the tedious and time-consuming manual retrieval processes that often plagues IT operations. With the capability to process log files within seconds, this utility replaces previous methods that were not only labour-intensive but also prone to human error.

The log parser is meticulously designed to focus on filtering out and reporting only the critical errors that necessitate intervention from engineering teams. This targeted approach significantly streamlines the process of issue detection and resolution, which in turn greatly reduces the operational overhead normally associated with these tasks. The solution has been thoughtfully packaged and disseminated across various internal systems, facilitating seamless integration with multiple applications. Additionally, the parser is designed to be executed through a command-line interface (CLI), offering maximum flexibility and ease of deployment in a myriad of operational environments.

On the cybersecurity front, a thorough and methodical approach was taken to identify, analyse, and remediate vulnerabilities present within the CXOne platform. Special emphasis was placed on addressing risks associated with the OWASP Top 10 security threats, which include vulnerabilities such as Injection flaws, Broken Authentication, Sensitive Data Exposure, Security Misconfiguration, Cross-Site Scripting (XSS), and Insufficient Logging and Monitoring. Remediation strategies were then implemented in strict accordance with industry best practices, enhancing the overall security posture of the platform and ensuring a robust defense against potential attacks.

Additionally, the automation frameworks underpinning crucial enterprise applications, particularly the Master Rotation application and the SMS module within UKG (Ultimate Kronos Group), were subject to extensive debugging, stabilization, and optimization efforts. A systematic review of automation scripts was undertaken to identify failure points, followed by the application of corrective measures aimed at restoring the reliability, scalability, and maintainability of these workflows. These enhancements have contributed to a marked reduction in the need for manual intervention, an increase in operational efficiency, and an overall improvement in system robustness.

The work detailed here is a reflection of the practical application of cybersecurity principles, log analytics, and automation engineering within the context of large-scale IT environments. It represents a significant bridging of theoretical knowledge with tangible, enterprise-grade technical solutions designed to improve efficiency and security in modern IT infrastructures. This comprehensive approach not only addresses current challenges but also lays the groundwork for future advancements in automation and cybersecurity practices across the organization.

#### ACKNOWLEDGEMENT

I would like to express my sincere gratitude to everyone who supported and guided me throughout the successful completion of this project.

First and foremost, I am deeply thankful to **Mr Abhishek Anand**, my manager at **UKG**, for giving me the opportunity to work on this impactful project and for his continuous support and encouragement throughout my internship.

I would also like to extend my heartfelt thanks to my mentors, **Mr Vaibhaw Kumar** and **Mr Paritosh Dwivedi**, whose invaluable guidance, technical expertise, and constant motivation played a crucial role in shaping this project. Their insights and feedback were instrumental in overcoming challenges and achieving meaningful outcomes.

I am also grateful to my **college professors and faculty members**, whose teachings laid the strong foundation of knowledge and discipline required to take on this project. Their academic mentorship has been a source of inspiration and direction throughout my journey.

Lastly, I would like to thank all my peers, team members at UKG, and everyone who directly or indirectly contributed to the successful execution of this work.

This project is the result of collective support, and I am truly appreciative of all the guidance and encouragement I have received.

Date: (Signature)

Name of the Student: Honey Pradhan

Roll No : BTECH/10168/21.

#### CONTENTS

[ABSTRACT i](#_TOC_250003)

[ACKNOWLEDGEMENT ii](#_TOC_250002)

[LIST OF FIGURES iii](#_TOC_250001)

[LIST OF TABLES iv](#_TOC_250000)

CHAPTER 1 INTRODUCTION 1

UKG APPLICATIONS 1

CFC APPS 2

AGENDA 4

SPLUNK AND ITS USAGE IN APPLICATIONS 5

OWASP AND CXONE 6

CUCUMBER FRAMEWORK 7

CHAPTER 2 LITERATURE REVIEW 9

CHAPTER 3 METHODOLOGY 12

SPLUNK LOG PARSER 12

CXONE CYBERSECURITY THREATS 14

AUTOMATION FRAMEWORK: MRST 15

AUTOMATION FRAMEWORK: SMS-SHIFT-FILL 16

TOOLS AND TECHNOLOGIES USED 16

CHAPTER 4 EXPERIMENTAL RESULTS AND DISCUSSION 19

SPLUNK LOG PARSER 19

CXONE CYBERSECURITY THREATS 25

AUTOMATION FRAMEWORK: MRST 26

AUTOMATION FRAMEWORK: SMS-SHIFT-FILL 27

CHAPTER 5 CONCLUSION AND FUTURE WORK 29

**REFERENCES 31**

#### LIST OF FIGURES

|  |  |  |
| --- | --- | --- |
| Figure 1.1 | UKG Key Products | 1 |
| Figure 1.2 | CFC Applications | 3 |
| Figure 3.1 | Overview | 12 |
| Figure 3.2 | High Level Architecture of the entire application | 13 |
| Figure 3.3 | Application Overview | 13 |
| Figure 4.1 | Created Package in RTSH | 19 |
| Figure 4.2 | SPL Query | 20 |
| Figure 4.3 | Package Structure including binaries and batch script | 20 |
| Figure 4.4 | Installed Necessary Dependencies using install.bat | 21 |
| Figure 4.5 | Auto-downloaded logs of RTSH from Splunk using pre-defined query for last 24 hrs for 31-03-2025.Time taken: 58s | 21 |
| Figure 4.6 | Auto-downloaded logs of RTSH from Splunk using pre-defined query for last 24 hrs for 02-04-2025.Time taken: 41s | 22 |
| Figure 4.7 | Final Output: tenant-wise logs that require ENG fixing, and a summary report | 22 |
| Figure 4.8 | Tenant specific log files | 23 |
| Figure 4.9 | One of the tenant files viewed | 23 |
| Figure 4.10 | Log Summary Report | 23 |
|  |  |  |
| Figure 4.11 | CXOne Portal with all threats resolved | 25 |
| Figure 4.12 | All test cases passed for MRST | 27 |
| Figure 4.13 | All test cases passed for SMS-SHIFT-FILL | 28 |
|  |  |  |

#### LIST OF TABLES

|  |  |  |
| --- | --- | --- |
| Table 1.1 | Literature Review | 10 |
| Table 1.2 | Large Language Models on Vertex AI | 11 |

***CHAPTER 1***

# INTRODUCTION

### UKG Applications

### UKG (Ultimate Kronos Group) is a global leader in workforce management and HCM solutions, formed in 2020 from the merger of Ultimate Software and Kronos Incorporated. It provides cloud-based applications that help organizations optimize their workforce with tools for HR, payroll, time and attendance, talent management, and employee engagement.

### 

Figure 1.1 UKG Key Products

* **UKG Pro:**

UKG Pro is a comprehensive HCM solution for large organizations, integrating payroll, talent management, benefits, and workforce analytics into a single cloud-based platform. It offers tools for employee engagement, performance management, recruitment, and learning, streamlining HR processes and enhancing employee experiences.

* **UKG Ready:**

UKG Ready is a versatile workforce management solution designed for small to mid-sized businesses. It includes features for HR, payroll, scheduling, and talent management, providing an intuitive interface and flexible deployment options to manage the workforce efficiently.

* **UKG Dimensions:**

UKG Dimensions is an advanced workforce management tool for large enterprises, focusing on timekeeping, scheduling, and analytics. It employs AI-driven scheduling and real-time data to optimize labor costs and enhance workforce efficiency, catering to complex industry needs.

* **UKG TeleStaff:**

UKG TeleStaff is a specialized scheduling solution for public safety organizations. It automates shift scheduling and communications, helping agencies manage their workforce effectively while integrating with other UKG platforms for a tailored experience.

* **UKG Workforce Central:**

UKG Workforce Central is a robust management solution for large enterprises, offering tools for time tracking, scheduling, absence management, and labor compliance. It helps optimize labor costs and boost productivity, providing managers with actionable insights through real-time reporting.

* 1. **CFC APPS**

The CFC applications within UKG comprise a series of specialized workforce management tools tailored to address intricate staffing logistics, automate scheduling, and empower employee self-service in large organizations. These applications are essential for maintaining efficient shift allocation, planning leave, and making real-time staffing adjustments.

* **Master Rotation:** The Master Rotation module oversees predefined cyclic work schedules for various departments. It automates the creation of foundational schedules for employees by aligning shift patterns with contractual agreements, individual preferences, and operational requirements. This process ensures equitable rotation, minimizes scheduling errors, and acts as the base layer for all future shift assignments or changes.
* **SMS Shift Fill:** The SMS Shift Fill tool streamlines real-time shift management by automatically alerting eligible employees about available shifts via SMS. Employees can quickly respond to accept or decline the opportunity, enabling supervisors to efficiently fill vacancies without the need for manual follow-ups. This approach significantly reduces staffing gaps, enhances response times, and ensures adherence to labor regulations and seniority-based assignments.

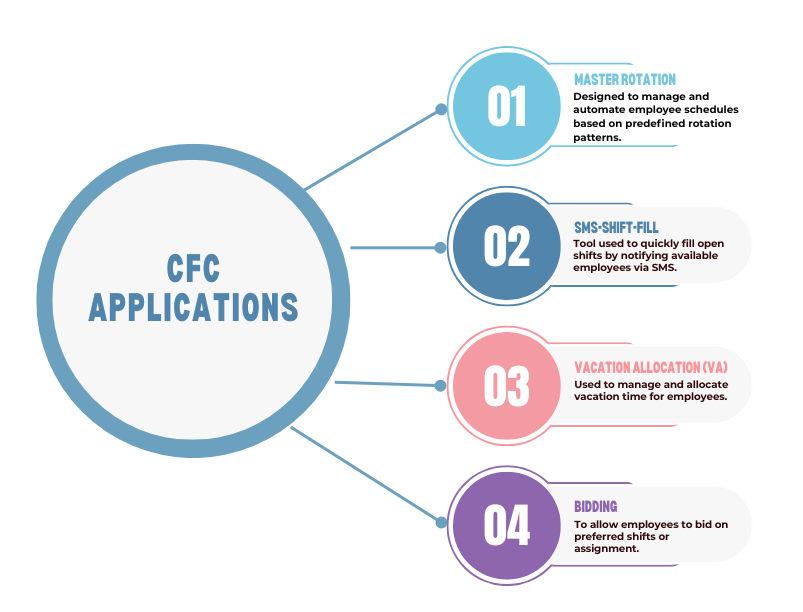


Figure 1.2 CFC Applications

* **Vacation Allocation:** This module automates the management and tracking of vacation time distribution. It allocates time off based on criteria such as seniority, hours worked, and departmental limits. Employees can submit vacation requests through the system, which processes approvals while ensuring staffing levels remain balanced. Vacation Allocation promotes fairness and transparency while lightening the administrative burden.
* **Bidding:** The Bidding application allows employees to bid for available shifts, vacation slots, or schedules based on factors including seniority, availability, or contractual stipulations. The process is automated and governed by rules, ensuring fairness and compliance with collective bargaining agreements. Bidding windows, priorities, and eligibility filters can be tailored to meet organizational policies. These CFC applications function cohesively to provide automated, equitable, and policy-compliant scheduling and staffing processes, thereby enhancing efficiency and employee satisfaction within large, shift-based workforces.

**1.3 AGENDA**

* To design and implement an end-to-end AI utility for log monitoring  
  To develop an AI-driven utility named **Splunk Log Parser** that automates daily log monitoring by downloading log files from Splunk for all applications, processing them within seconds, and reporting only those errors that require engineering intervention. The solution is designed to eliminate manual effort and functions as a command-line utility.
* To create a **deployable package** and integrate it across internal applications  
  To package the utility into a deployable distribution format and integrate it with UKG’s internal application ecosystem, ensuring ease of adoption, reusability, and consistency across teams.
* To perform **cybersecurity assessments** and resolve OWASP Top 10 vulnerabilities  
  To conduct security assessments for CXOne applications, identify vulnerabilities based on the OWASP Top 10 security risks, and implement appropriate remediations to enhance application security and compliance.
* To enhance and integrate the automation framework for CFC applications  
  **To improve the automation framework** for CFC applications such as Master Rotation and SMS Shift-Fill, and integrate it with the Jenkins pipeline to enable automatic test execution on every build trigger, supporting continuous integration and deployment.

**1.4 SPLUNK AND ITS USAGE IN APPLICATIONS**

Splunk Enterprise serves as a centralized platform at UKG for effective log management and operational intelligence. It collects, indexes, and analyzes machine-generated data, including application logs, system events, and outputs from backend services throughout the organization. By consolidating logs from a variety of internal tools and services, Splunk offers a unified interface for searching, monitoring, and alerting, significantly enhancing visibility, troubleshooting efforts, and performance tracking.

UKG integrates Splunk Enterprise with various internal applications to optimize operations and bolster system reliability. Logs from essential CFC applications, such as Master Rotation, SMS Shift Fill, Vacation Allocation, and Bidding, are continuously sent to Splunk. These logs provide insights into scheduling activities, API interactions, and backend job execution. Splunk enables real-time monitoring of these processes, allowing teams to quickly identify anomalies or service disruptions. In addition, CXOne applications utilize Splunk for logging events related to user authentication, session activities, and system performance. These logs not only facilitate operational monitoring but also aid in security audits, as Splunk helps pinpoint and address issues related to OWASP Top 10 vulnerabilities.

Beyond monitoring applications, Splunk is vital in UKG’s automation and Continuous Integration/Continuous Deployment (CI/CD) environments. Logs from Jenkins pipelines, including test outcomes and deployment events, are collected and analyzed within Splunk. This integration enhances quality assurance by enabling teams to trace test failures, recognize trends, and improve the stability of releases. Additionally, custom tools like the Splunk Log Parser are designed to work directly with the Splunk API, automatically downloading logs, processing them in seconds, and filtering out only the errors that necessitate engineering attention. This approach significantly minimizes manual analysis time and promotes more effective monitoring.

Through these integrations, Splunk Enterprise acts as a foundational tool for proactive issue detection, performance analytics, and security compliance within UKG’s internal technology framework.

**1.5 OWASP AND CXONE**

The Open Web Application Security Project (OWASP) offers a widely acknowledged list of the most significant security risks facing web applications, referred to as the OWASP Top 10. This list acts as a framework for developers and security teams to recognize, identify, and address the most prevalent and impactful security vulnerabilities that endanger the confidentiality, integrity, and availability of applications.

The OWASP Top 10 encompasses vulnerabilities such as:

* Injection (e.g., SQL, NoSQL, OS Command Injection)
* Broken Authentication
* Sensitive Data Exposure
* XML External Entities (XXE)
* Broken Access Control
* Security Misconfiguration
* Cross-Site Scripting (XSS)
* Insecure Deserialization
* Using Components with Known Vulnerabilities
* Insufficient Logging and Monitoring

These vulnerabilities can result in significant repercussions, including data breaches, privilege escalation, unauthorized access, or even complete system compromise.

At UKG, the CXOne portal is a vital internal application that oversees various aspects of customer experience, such as communication logs, workflow execution, and administrative controls. Given its broad scope and access to sensitive operational data, securing this portal is critical.

During the assessment and review of the CXOne application, multiple vulnerabilities corresponding to the OWASP Top 10 were uncovered and addressed. These included:

* **Broken Access Control:** Users could access administrative endpoints without adequate role verification. Access control mechanisms were revised to enforce strict role-based permissions across all modules.
* **Cross-Site Scripting (XSS):** Unsanitized user input in specific forms allowed for the injection of harmful scripts. Measures for input validation and output encoding were implemented to minimize this risk.
* **Security Misconfiguration:** Default configurations and exposed debugging endpoints were found in staging environments. These were disabled, and appropriate environment-specific settings were established.
* **Sensitive Data Exposure:** Some API responses contained overly detailed error messages and stack traces that might disclose internal implementation information. Error handling was strengthened, and sensitive fields were masked or omitted.
* **Using Components with Known Vulnerabilities:** Outdated third-party libraries and dependencies were identified in the software stack. These were updated to their latest secure versions after thorough regression testing.

All identified vulnerabilities were documented, prioritized according to severity, and addressed in collaboration with the development and DevSecOps teams. The remediations were confirmed through re-testing and routine automated scans, ensuring that the CXOne portal meets industry-standard security practices and enterprise compliance policies.

# This initiative not only enhanced the security posture of the CXOne portal but also helped establish a stronger security foundation for other interconnected applications within the UKG ecosystem.This initiative not only enhanced the security posture of the CXOne portal but also helped establish a stronger security foundation for other interconnected applications within the UKG ecosystem.

**1.6 CUCUMBER FRAMEWORK**

# Cucumber is a popular open-source testing framework that supports Behavior-Driven Development (BDD). It facilitates collaboration among developers, testers, and non-technical stakeholders by allowing test scenarios to be articulated in plain English using the Gherkin language. The primary goal of Cucumber is to create automated tests that are both easy for humans to read and executable by machines, effectively bridging the gap between business requirements and technical implementation.

# Within the Cucumber framework, test scenarios are outlined in feature files using a Given-When-Then format, which sets out the preconditions, actions, and expected results. These scenarios depict the system's behavior from the user's perspective, keeping the focus on business objectives rather than technical specifics.

# Each step defined in the feature file corresponds to specific step definitions, which are written in programming languages such as Java, JavaScript, or Python, based on the testing environment. These step definitions contain the actual code that interacts with the application, often utilizing libraries like Selenium WebDriver for web applications or Appium for mobile applications.

# Cucumber encourages modular and reusable code structures, allowing teams to manage extensive test suites effectively. Additionally, it integrates seamlessly with popular CI/CD tools like Jenkins, enabling automated test execution as part of the build process. The test reports generated by Cucumber are detailed and can be enhanced further with plugins for better visualization and traceability.

# At UKG, the Cucumber framework is employed to automate functional testing for critical internal applications, such as Master Rotation and SMS Shift Fill. These tests validate workflows, data integrity, and user interactions across various build environments. The framework is connected to Jenkins pipelines, allowing test suites to run automatically with every code commit or build trigger. This setup supports continuous integration and provides rapid feedback on code quality.

# Cucumber’s alignment with BDD, its user-friendly syntax, and its smooth integration with test automation tools make it a powerful solution for validating complex application behaviors while ensuring that all stakeholders are in sync regarding expectations.

***CHAPTER 2***

# LITERATURE REVIEW

# Vertex AI is Google Cloud Platform’s (GCP) unified, end-to-end machine learning (ML) platform designed to streamline the development, deployment, and management of ML models at scale. It integrates all necessary components for the ML lifecycle into a single environment, allowing data scientists and ML engineers to build, train, deploy, and monitor models more efficiently.

# Unlike traditional GCP services that required switching between multiple tools (like AI Platform, AutoML, and BigQuery ML), Vertex AI brings everything under one umbrella. It supports both custom model development and pre-trained models with the flexibility to use no-code, low-code, or full-code approaches.

# Key Capabilities of Vertex AI:

# Managed datasets and pipelines for streamlined ML workflows

# Experiment tracking to compare model performance over time

# Vertex AI Workbench, a Jupyter-based environment for development

# Model Registry and Feature Store to reuse models and engineered features

# MLOps tools including CI/CD support, monitoring, and retraining automation

# Integration with Kubeflow pipelines for orchestrating complex workflows

# Model Offerings:

# Vertex AI supports various types of models, including:

Pre-trained APIs:  
Out-of-the-box models for vision (image recognition, OCR), text (sentiment analysis, translation, entity extraction), and speech (transcription, synthesis).

# AutoML Models: Custom models built with minimal coding by training on user-provided datasets. Suitable for tabular data, images, text, and video.

# Custom-Trained Models: Full-code flexibility to train models using TensorFlow, PyTorch, scikit-learn, and other frameworks. These can be deployed directly using Vertex AI’s managed infrastructure.

# Foundation and Generative Models: Includes access to large language models (LLMs) and other foundation models (like PaLM) through Vertex AI Studio and APIs, enabling use cases in text generation, summarization, chatbots, and more.

# Vertex AI Matching Engine: A vector database and similarity search engine for large-scale recommendation systems and semantic search applications.

# 

# Table 1.1 Literature Review

# Various Large Language Models:

# Gemini 1.5 Flash: Designed for fast, versatile performance across a wide range of tasks.​

# Gemini 1.5 Pro: Supports complex reasoning tasks requiring more intelligence; 2M long context.​

# Gemini 2.0 Flash: Provides next-generation features and speed for a diverse variety of tasks.​

# Gemini 2.0 Flash-Lite: Offers cost-effective and low-latency performance; supports high throughput.​

# Gemini 2.0 Pro: Offers the strongest model quality, especially for code and world knowledge; 2M long context.

# Gemini 2.0 Flash Thinking: Offers stronger reasoning capabilities and includes the thinking process in responses.​

# Claude 3.7 Sonnet: Latest model with advanced reasoning capabilities.​

# Claude 3.5 Sonnet v2: Enhanced version with improved performance.​

# Claude 3.5 Sonnet: Offers improved performance and efficiency.

# 

# In summary, Vertex AI simplifies and accelerates ML development on GCP, supporting everything from simple AutoML use cases to complex, production-grade custom model workflows—all backed by robust MLOps capabilities.

# 

# Table 1.2 Large Language Models on Vertex AI

***CHAPTER 3***

# METHODOLOGY

# During my internship, I worked on four separate projects, each aimed at addressing specific technical challenges in the areas of log parsing, automation testing, and cybersecurity. The methodology for each project was carefully tailored to its objectives and technological requirements, while also ensuring integration with broader organizational goals.

# 3.1 SPLUNK LOG PARSER

The goal of this project is to automate the log analysis pipeline using a hybrid strategy that combines lightweight pre-filtering techniques with the semantic reasoning capabilities of a Large Language Model (LLM). The process initiates with the automatic extraction of log data from a Splunk server using REST APIs or SDK integration. These logs, exported in JSON format, are downloaded at regular intervals or via event-driven queries. Once obtained, the logs are parsed to extract relevant fields such as timestamps, transaction IDs (trxid), tenant IDs, error codes, and message content (msg), forming a structured dataset for further processing.

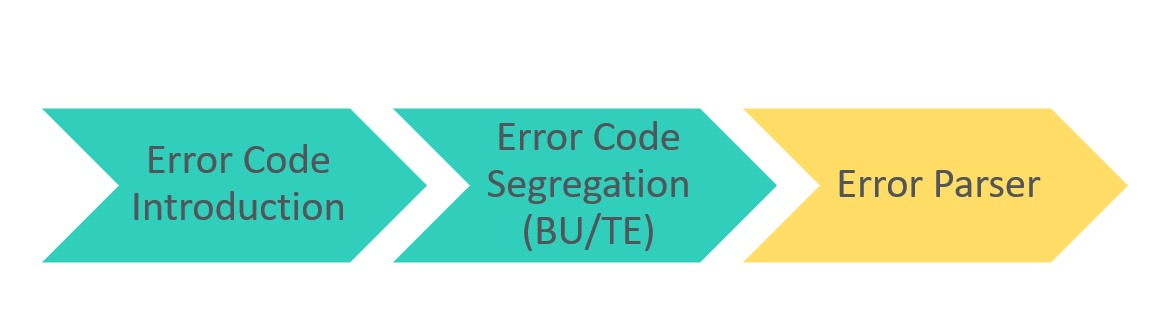
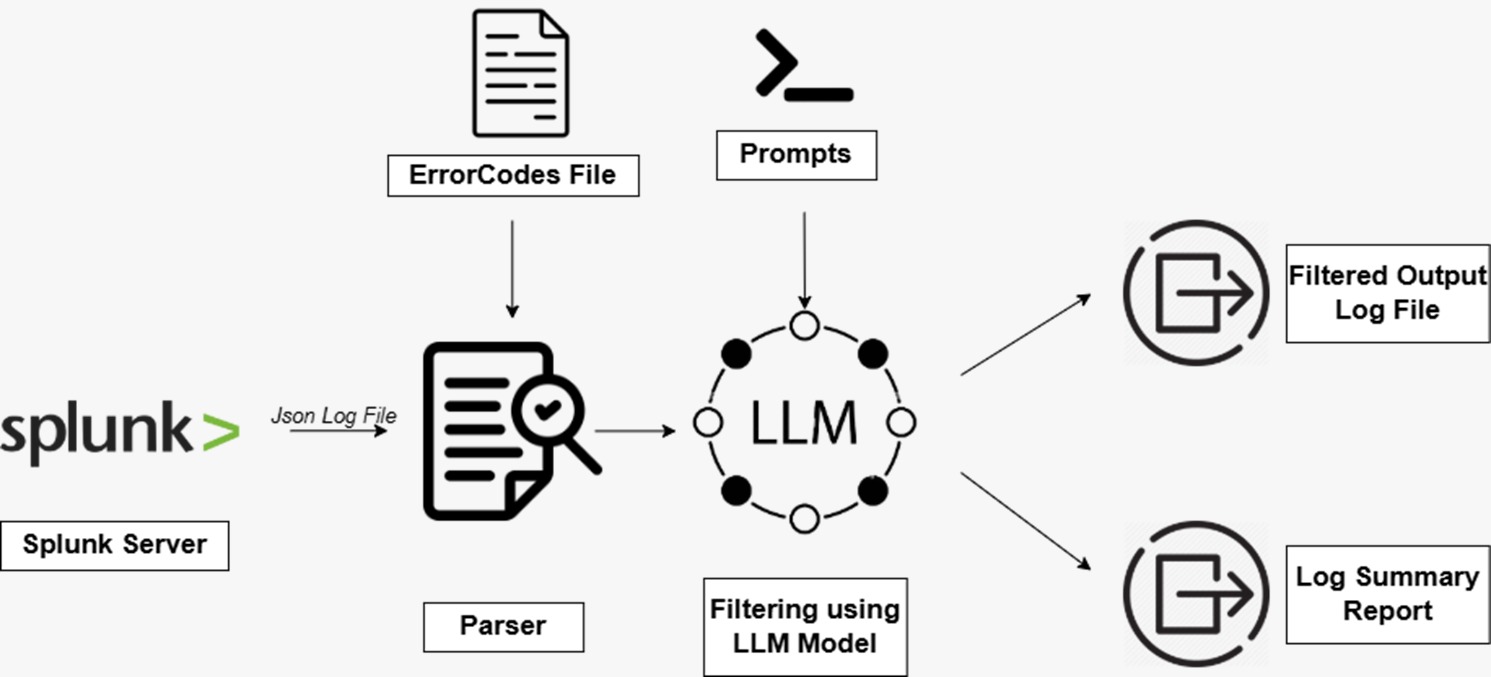


Figure 3.1 Overview

A pre-processing stage is applied to eliminate noise and reduce the input size for the LLM. This stage uses a configurable error code file that includes known benign or ignorable error patterns. Logs containing these error codes are discarded early, improving processing efficiency. Further filtering is optionally performed on the basis of transaction IDs or tenant-specific heuristics, helping to localize log content to a finer granularity.

The cleaned logs are then fed into a Large Language Model for intelligent classification. This is where the system leverages domain-informed prompt engineering to guide the model’s behaviour. The prompt is constructed to help the LLM distinguish between ignorable and severe errors based on contextual clues and expected root causes.

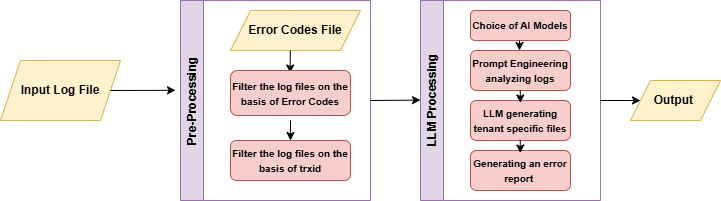
Figure 3.2 High Level Architecture of the entire application

Figure 3.3 Application Overview

For instance, prompts direct the model to flag logs that indicate issues like service misconfiguration, authentication failure, or database connectivity problems, while filtering out routine operational messages. The design of the prompt was informed by expert understanding of the application domain, ensuring that the model can effectively capture the semantics of logs beyond simple keyword matching.

Factors considered in the selection included latency, cost, ability to handle high-throughput data, and most importantly the model's performance in accurately identifying and contextualizing error patterns in log data. Gemini-2-flash-001, part of Google’s Gemini family, demonstrated superior inference speed and contextual awareness during evaluation.

The model was accessed through **Google Cloud’s Vertex AI platform**, which offers a seamless interface for model selection, deployment, and API interaction. Integration involved setting up a Vertex AI project, enabling the Gemini APIs, and authenticating via service accounts. Requests were constructed using RESTful endpoints, where filtered logs and the domain-specific prompt were passed as payloads. The responses from the model were parsed and interpreted to determine which logs should be retained.

The LLM's output logs deemed relevant was then split by tenant, creating tenant-specific log files. This enables efficient debugging by isolating errors per tenant. Additionally, a log summary report was generated, capturing key metrics such as frequency of critical errors, root cause trends, and tenant-wise distribution of issues. This two-part output tenant logs and a consolidated report provides both micro and macro-level insights to operations teams, helping them quickly identify and resolve issues. Overall, this methodology balances efficiency and intelligence, harnessing the filtering power of LLMs without overwhelming computational resources, and adapting seamlessly to domain-specific log analysis needs.

.

# 3.2 CXONE CYBERSECURITY THREATS

# The regular SAST and DAST scans were integrated into the CI pipeline so that every run against the Master Rotation, SMS-SHIFT-FILL, VA, and Bidding modules would automatically feed vulnerabilities into the Jira backlog with OWASP category tags and CVSS scores. Once the findings were triaged, the affected JARs were updated in the UKG Artifactory each CVE’s vendor patch notes and NVD entries having been cross-referenced to ensure that no vulnerable dependency versions remained. For the code-level issues, the flagged call paths were inspected in collaboration with the respective module owners, string-concatenated SQL queries were replaced with parameterized ones, strict input allow-lists were implemented, and unsafe deserialization routines were swapped out for vetted libraries, thereby closing the OWASP Injection and Deserialization gaps.

# With patches prepared both in the Maven pom.xml and as targeted code changes—feature branches were committed and CI builds were executed. Automated unit and integration tests confirmed that existing functionality was preserved, after which a second round of SAST and a staging DAST scan were run to verify that the original vulnerabilities had been remediated. For the highest-severity issues, a small manual pentest exercise was conducted against the staging environment, simulating real-world attack vectors to demonstrate that the fixes endured under adversarial conditions.

# A canary release was then orchestrated to a low-impact tenant group, with performance and error metrics monitored in real time. After a successful 24-hour observation window with no regressions, the release was automatically promoted across all production clusters during the next maintenance window. Throughout the process, every CVE reference, OWASP mapping, root-cause analysis, and test result was documented in the internal knowledge base, ensuring that future scans would either bypass these issues or link directly to the comprehensive remediation artifacts.

# 3.3 AUTOMATION FRAMEWORK: MRST

# The Cucumber-based API test suite for the Master Rotation (MRST) module was first audited to identify failing scenarios and unstable steps, after which the Gherkin feature files were refactored to employ a clean Given-When-Then pattern with parameterized Examples tables. Shared REST call logic was extracted into common step libraries and externalized test data JSON fixtures, while authentication and environment configuration were centralized in Before/After hooks to remove duplication. Assertions were hardened by replacing broad response checks with precise JSON schema validations and explicit HTTP status-code assertions, and retry mechanisms were added to handle transient network errors. Downstream dependencies were stubbed using WireMock to ensure isolation of MRST endpoints. Once all step definitions, fixtures, hooks, and mocks had been updated, the suite was executed successfully both locally and within a Dockerized staging environment to validate cross-platform consistency. All development and maintenance of the suite were performed within the Eclipse IDE to leverage its powerful debugging and refactoring tools. Finally, the stabilized tests were integrated into the Jenkins pipeline: a dedicated MRST API job was configured to run the suite in parallel for each pull request, and detailed HTML and JUnit XML reports were collected and published to the Jenkins dashboard for real-time visibility of test health.

# 3.4 AUTOMATION FRAMEWORK: SMS-SHIFT-FILL The Cucumber-based test suite for the SMS-SHIFT-FILL module was rigorously overhauled to eliminate instability and improve maintainability. The existing feature files were audited to pinpoint failing and flaky scenarios, and all Gherkin definitions were refactored to follow a consistent Given-When-Then structure with parameterized Examples tables for data-driven coverage. Common REST-call logic was abstracted into shared step libraries, and test data was externalized into JSON fixtures to promote reuse. Authentication and environment setup were centralized within Before/After hooks to remove redundant code. Assertions were strengthened by replacing generic response checks with strict JSON schema validations and explicit HTTP status-code assertions, while retry logic was added to gracefully handle transient network or service delays. External dependencies such as the SMS gateway and downstream transformation services were stubbed using WireMock, ensuring the SMS-SHIFT-FILL endpoints could be tested in isolation. All development and maintenance of the suite were performed within the Eclipse IDE to leverage its powerful debugging and refactoring tools. After updating step definitions, fixtures, hooks, and mocks, the suite was executed successfully both locally and in a Dockerized staging environment to verify cross-platform consistency. Finally, the hardened tests were deployed into Jenkins: a dedicated SMS-SHIFT-FILL API job was configured to run the suite in parallel on every pull request, with comprehensive HTML and JUnit XML reports automatically published to the Jenkins dashboard for real-time visibility into API test health.

# TOOLS AND TECHNOLOGIES USED

# GitHub: Version Control System. GitHub is a platform for managing code using Git. It allows developers to collaborate, track changes, review code via pull requests, and manage projects efficiently.

# GCP: Vertex AI Framework. Vertex AI is a fully managed machine learning (ML) platform offered by Google Cloud Platform (GCP). It provides tools and infrastructure to build, deploy, and scale ML models with ease. Vertex AI enables both beginner and expert ML developers to train models using AutoML or custom code. It integrates seamlessly with tools like TensorFlow, PyTorch, and scikit-learn. The framework also supports deploying pre-trained models, including large language models (LLMs), for tasks such as text classification, sentiment analysis, translation, summarization, and chatbots.Vertex AI also offers experiment tracking, model monitoring, feature storage, and hyperparameter tuning—all designed to streamline the ML lifecycle in production.

# Postman: API Testing. Postman is a collaboration platform used for building, testing, and documenting APIs. It provides a user-friendly interface to create HTTP requests (GET, POST, PUT, DELETE), add authentication headers, handle parameters, and test responses. Postman is especially useful in backend development and QA for validating the behavior and reliability of RESTful APIs. It supports scripting in JavaScript to automate tests and assertions on responses. Additionally, it enables the creation of environments and collections, allowing developers to simulate different API workflows and share them across teams.

# Docker: For running PGBOUNCER and REDIS. Docker is a containerization platform that enables developers to package applications and their dependencies into isolated environments called containers. This ensures that software runs consistently across different systems.

# Splunk: Downloading Log Files. Splunk is a platform designed for searching, analyzing, and visualizing machine-generated data, especially logs. It is widely used in IT operations, security, and monitoring to gain real-time insights from large volumes of unstructured log data. Splunk was used to download log files for debugging or diagnostics. It helps track issues across systems by filtering logs based on time, error type, or component. Splunk’s querying language (SPL) allows advanced search capabilities, making it a critical tool for troubleshooting and system observability.

# Vertex AI Framework: The Vertex AI framework provides access to various Large Language Models (LLMs) like PaLM, Gemini, and open-source models. These models are trained on vast datasets and are capable of performing complex natural language understanding and generation tasks. Within Vertex AI, developers can leverage LLMs for applications such as chatbots, content generation, code completion, sentiment analysis, summarization, and more. These models can be used via simple APIs, and Google also offers features for fine-tuning models on custom datasets to better fit specific business needs.

# CXOne: Automated Security Scanning Tool

# Dbeaver: Database Management Tool

# Visual Studio Code, Eclipse, IntelliJ Idea: Integrated Development Environment. Visual Studio Code is lightweight and supports many languages via extensions. It is favored in enterprise environments and supports large-scale project management, integrated debugging, and plugin development. IntelliJ IDEA, developed by JetBrains, is known for its smart code assistance, deep integration with build tools like Maven/Gradle, and strong support for Java, Kotlin, and Spring Framework.

***CHAPTER 4***

# EXPERIMENTAL RESULTS AND DISCUSSION

# 4.1 SPLUNK LOG PARSER

# The final output of the Splunk log parser includes individual tenant-wise log files along with a consolidated log summary report.

# Tenant-wise log files are generated by extracting and organizing logs specific to each tenant. This segregation allows for targeted analysis, easier debugging, and improved visibility into tenant-specific issues or behaviors. It also ensures cleaner data organization and compliance with multi-tenant system design.

# The log summary report provides a high-level overview of the parsed logs, including metrics such as total number of log entries per tenant, frequency of error or warning messages, and timestamp-based activity trends. This report helps in quickly identifying tenants with the highest error rates, recurring issues, or abnormal activity..

# Fig. 4.1 Created Package in RTSH

# 

# Fig. 4.2 SPL Query

# 

# Fig. 4.3 Package Structure including binaries and batch script

# 

# Fig 4.4 Installed Necessary Dependencies using install.bat

# 

# Fig 4.5 Auto-downloaded logs of RTSH from Splunk using pre-defined query for last 24 hrs for 31-03-2025. Time taken: 58s

# 

# Fig 4.6 Auto-downloaded logs of RTSH from Splunk using pre-defined query for last 24 hrs for 02-04-2025. Time taken: 41s

# 

# Fig 4.8 Tenant specific log files

# 

# Fig 4.9 One of the tenant files viewed

# 

# Fig 4.10 Log Summary Report

# Figure 4.4 illustrates the use of the install.bat script, a self-contained batch file designed to automate the setup process for the project. When executed, this script installs all necessary project dependencies, including libraries, configurations, and environment-specific tools. This eliminates the need for manual installation steps, reducing setup time and minimizing the risk of missing or incompatible components. By streamlining the onboarding and deployment process, install.bat ensures consistency across all systems where the application is run.

# Figure 4.5 presents the results of processing RTSH log files collected over a 24-hour period. The raw log data contained 1,680 individual lines, which were fed into the application for analysis. The tool successfully parsed and evaluated all log entries in just 58 seconds. During this automated analysis, it was determined that none of the log lines required actual engineering support, indicating that the logs consisted solely of routine or ignorable entries. In comparison, conducting the same analysis manually would typically take 2 to 3 hours, with a higher risk of oversight or human error. This performance not only highlights the speed of the solution but also its reliability in identifying only actionable information.

# Figure 4.6 shows a similar case involving logs from another day, with 1,220 lines. The application was able to isolate and flag 4 log entries that required genuine engineering attention, again completing the task in just 48 seconds. This scenario emphasizes the tool’s ability to distinguish between routine logs and critical issues with precision. Moreover, the application efficiently reduced a 1 MB log file into a refined 5 KB output, containing only the relevant log lines that needed review. This massive reduction in file size not only saves storage but also simplifies downstream debugging and reporting.

# In both instances, the final output consists of tenant-wise log files, allowing teams to review logs specific to each tenant individually. This is particularly useful in multi-tenant systems where isolating issues per tenant is crucial for support and diagnostics. Alongside these files, a comprehensive log summary report is generated, giving a high-level overview of log volumes, error frequencies, and key insights. This summary serves as a quick reference for support engineers and managers, helping them make informed decisions without sifting through thousands of raw log lines.

# To further streamline the workflow, a Jenkins job has been set up to automate the entire log analysis process. This job is triggered for each application and executes the complete pipeline from downloading the past 24 hours of logs, parsing and filtering them, identifying critical issues, to generating tenant-wise log files and a summary report.

# What previously required 2–3 hours of manual effort per application is now completed in a matter of seconds through this automation. By leveraging Jenkins, the solution ensures consistency, speed, and reliability across all applications, with no human intervention required. This not only boosts operational efficiency but also allows engineering teams to focus on resolving real issues rather than spending time sifting through logs.

# 4.2 CXONE CYBERSECURITY THREATS

# Fig 4.11 CXOne Portal with all threats resolved

# During the integration and testing phase, a thorough dependency audit revealed multiple security vulnerabilities in the application's third-party libraries. These vulnerabilities, if left unaddressed, could potentially compromise the security, stability, and performance of the overall system.

# One of the most critical issues was identified in the ag-grid-community library, which had a transitive dependency on an outdated version of Angular. This posed a significant risk, particularly because Angular is a core framework that affects the behavior of front-end components. In response, both the RTSH and SMS applications were comprehensively upgraded to Angular version 17. This upgrade not only resolved the dependency issue but also brought several improvements in terms of security patches, performance optimizations, and long-term support.

# Further investigation revealed similar vulnerabilities in a number of commonly used backend libraries, including jackson-databind, babel-traverse, commons-io, inflight, and semver. These libraries contained known CVEs (Common Vulnerabilities and Exposures) that needed to be patched to maintain compliance with modern security standards.

# A targeted fix was applied to the jackson-databind vulnerability by procuring a highly updated version of the FasterXML Databind JAR. This JAR was then securely uploaded and registered in the UKG Artifactory, ensuring that all future builds would reference the patched version. The application was subsequently recompiled and executed using this updated dependency to verify that no compatibility issues were introduced and that the vulnerability was effectively resolved.

# A similar approach was taken for the remaining affected libraries. Each was carefully evaluated, updated to the latest safe version, and validated through build and runtime testing. These updates not only eliminated the immediate vulnerabilities but also improved the overall health of the dependency graph, reducing future maintenance overhead.

# Through this process, the application was made significantly more secure and resilient, while also aligning it with current best practices in dependency management and software supply chain hygiene.

# 4.3 AUTOMATION FRAMEWORK: RTSH

# As part of the automation improvement efforts, the API testing suite for the RTSH application implemented using Cucumber was thoroughly analyzed and debugged within IntelliJ IDEA. Initially, the test suite exhibited 35 failing test cases, indicating issues with either outdated test logic, broken API contracts, or unstable configurations.

# A detailed investigation was conducted to identify the root causes of these failures. This involved reviewing step definitions, analyzing API request/response behavior, verifying authentication flows, and ensuring test data consistency. Based on the findings, multiple areas were corrected, including outdated endpoint references, incorrect assertions, and missing configurations.

# After applying the necessary fixes and optimizations, the automation suite was re-executed, and all previously failing test cases passed successfully. This not only restored the reliability of the test framework but also enhanced its coverage and effectiveness in validating the RTSH APIs.

# As a result, the API automation is now fully functional and stable, enabling faster feedback during development and reducing manual testing efforts.

# 

# Fig 4.12 All test cases passed for RTSH

# 4.4 AUTOMATION FRAMEWORK: SMS-SHIFT-FILL

# A similar issue was observed in the SMS – Shift Fill module, where the Cucumber-based API automation suite was experiencing significant instability. Out of 48 total test cases, 40 were failing, severely impacting the reliability of automated regression testing. These failures were indicative of deeper issues, including outdated test logic, API schema changes, inconsistent test data, and misconfigured environment variables.

# Using Eclipse as the development and debugging environment, a detailed review of the test cases and step definitions was conducted. API endpoints were validated against the latest backend contracts, and necessary corrections were made to request payloads, expected responses, and assertion logic. Additionally, improvements were made to test setup and teardown procedures to ensure consistent test behaviour across environments.

# Following these updates, the test suite was re-executed, and all 40 previously failing test cases were successfully fixed. The result was a fully functional and stable automation suite for the Shift Fill module, with 48 out of 48 test cases passing. This restoration of test accuracy significantly enhances the confidence in automated QA for SMS and reduces manual verification efforts for future releases.

# Once the fixes were applied, the updated automation suite was deployed and executed via Jenkins in the QA environment using the integration prof

# 

# FFig 4.13 AAll test cases passed for SMS-SHIFT-FILL

***CHAPTER 5***

# CONCLUSION AND FUTURE SCOPE

# This project has significantly improved the efficiency, reliability, and maintainability of multiple internal applications at UKG, with a strong focus on log analysis automation, security remediation, and automated API testing. By building and integrating a custom log parser tool, thousands of raw Splunk log lines were processed in a matter of seconds—reducing analysis time from hours to seconds while providing tenant-wise outputs and comprehensive summary reports. This enabled teams to identify and focus only on meaningful issues requiring engineering support, ultimately accelerating incident resolution and improving operational efficiency.

# The use of batch scripts like install.bat and dist.bat allowed for seamless environment setup and packaging across systems, streamlining deployment. The log parser pipeline was further enhanced through Jenkins automation, making it possible to run the full analysis cycle for multiple applications in parallel, with minimal manual intervention.

# Security was another key area of focus. Several critical vulnerabilities were identified in transitive and direct dependencies across components such as ag-grid-community, jackson-databind, babel-traverse, and others. These were systematically resolved by upgrading Angular versions (to v17), updating libraries, and adding secure versions of JARs into UKG Artifactory, ensuring compliance with modern security standards while stabilizing the application environment.

# Additionally, API automation frameworks for both RTSH and SMS – Shift Fill modules were repaired and stabilized. With 35 and 40 test cases failing respectively, thorough debugging and test refactoring were carried out in IntelliJ IDEA, followed by successful execution in the QA environment via Jenkins (integration profile). All test cases passed post-fix, restoring confidence in automated regression testing and reducing reliance on manual QA efforts.

# Overall, the project not only addressed current challenges but also laid a strong foundation for future scalability, maintainability, and automation across multiple systems within UKG.

# Looking ahead, several key improvements and extensions are planned to further enhance the project’s impact:

# Autonomous Splunk Integration Enhancements: Currently, logs are manually downloaded from Splunk for analysis. While plans were underway to automate this process, UKG is now transitioning to a new Splunk server. As a result, future work will involve updating the log parser to automatically fetch logs from the new Splunk environment, ensuring uninterrupted automation and seamless integration with the new infrastructure.

# Extension to Other Applications: Building on the success of the RTSH and SMS modules, there is a clear opportunity to extend the automated API testing framework to other applications, specifically the VA (Virtual Assistant) and Bidding modules. These apps currently lack stable automation coverage, and incorporating them into the existing Cucumber-based framework will enhance testing reliability, speed up development cycles, and improve overall quality assurance.

# Enhanced Reporting and Analytics: While the current summary reports are effective, future iterations may introduce graphical dashboards or integrations with tools like Kibana or Grafana to provide more intuitive insights for engineering and support teams.

# CI/CD Optimization: Additional performance tuning and parallel test execution strategies can be explored in Jenkins to further reduce build and testing times, particularly as the number of applications and test suites grows.

# Dependency Health Monitoring: Introducing automated scans (e.g., using OWASP Dependency-Check or Snyk) as part of the CI pipeline will ensure that any newly introduced vulnerabilities in third-party libraries are flagged early and resolved proactively.

# REFERENCES

# [Google Cloud API Reference](https://cloud.google.com/vertex-ai/generative-ai/docs/model-reference/inference?hl=en&_gl=1*5lsviq*_ga*Nzc0MzI5MzgzLjE3Mzg3MzMwMzI.*_ga_WH2QY8WWF5*MTc0MDU1MjgyOS40OS4xLjE3NDA1NTM3NjUuNjAuMC4w)

# [Vertex AI Reference](https://python.langchain.com/docs/integrations/llms/google_vertex_ai_palm/)

# UKG Confluence

# [Python Reference](https://docs.python.org/3/)

# [Splunk REST API Reference](https://docs.splunk.com/Documentation/SIM/current/Develop/RESTv20)

# CXone Scans

# [Owasp Top 10 Cybersecurity Threats](https://owasp.org/www-project-top-ten/)

# THE DEFINITIVE GUIDE TO GOOGLE VERTEX AI*: Accelerate Your Machine Learning Journey with Google Cloud Vertex AI and MLOps Best Practices*.[1]

# A comprehensive overview of large language models.[2]