Security Crash Course – Mohamed Ashraf

Date: 2025-10-05

This crash course distills the secure ML chatbot platform built on Google Cloud, highlighting how its architecture satisfies network security, data protection, and governance requirements. It maps hands-on experience across multiple security domains and showcases Python-driven automation used to enforce policy and monitor the stack.

# 1. Reference Architecture Overview

The platform provisions a hardened GCP landing zone: Shared VPC hub, Cloud Armor-protected HTTPS load balancer, Cloud Run gateway, GKE Autopilot services, Vertex AI endpoints, and CMEK-backed data stores. Observability flows into Cloud Logging, Monitoring, Security Command Center, and Chronicle SIEM. Terraform modules separate networking, IAM, security, logging, and workload lifecycles, with environment-specific stacks for dev and prod.

# 2. Network Security Experience

* Designed segmented Shared VPC topology with private service access, Cloud NAT, and firewall policy modules enforcing least privilege.
* Implemented Cloud Armor WAF/bot management and reCAPTCHA Enterprise at the global HTTPS entry to absorb volumetric and Layer-7 attacks.
* Enabled mutual TLS and Binary Authorization on GKE, restricting egress via NAT and service perimeter boundaries (VPC Service Controls).
* Automated network compliance with Terraform + OPA/Config Validator checks in CI (denying open CIDRs, enforcing TLS 1.2+).

# 3. Data Security Experience

* CMEK-encrypted Cloud Storage, BigQuery, and Secret Manager assets enforced via organization policy and Terraform.
* Vertex AI pipelines restricted to private service endpoints; model artifacts stored in Artifact Registry with signed images.
* Secret rotation driven by Python automation that fetches short-lived tokens via Workload Identity Federation.
* Log sinks stream to immutable storage (Cloud Storage + BigQuery) for forensic readiness; Chronicle consumes findings for long-term analytics.

# 4. Governance & Policy-as-Code

* Terraform modules are wrapped with Conftest/OPA policies to block unapproved services, enforce CMEK, and mandate logging.
* GitHub Actions workflows require manual approvals for protected branches; plan files are signed and archived for audit.
* Security Command Center and Org Policy Service provide continuous posture management tied into incident runbooks.
* Devcontainer image standardizes tooling (Terraform, gcloud, opa, trivy) ensuring governance checks run identically on every workstation.

# 5. Domain Expertise Highlights

## Web application security

* Hardened FastAPI chatbot endpoints behind Cloud Run + IAP; CSP headers, JWT validation, and rate limiting with Cloud Armor rules.
* Automated DAST smoke tests triggered after deployment using Python-based Locust scenarios.

## Security assessments & penetration testing

* CI pipeline integrates Trivy, Checkov, Bandit, kube-bench, and custom OPA policies to block risky merges.
* Periodic manual assessments leverage Burp and gcloud IAP tunneling; findings drive Terraform remediations.

## Authentication & access control

* Adopted Workload Identity Federation (GitHub → GCP) eliminating long-lived keys; granular IAM Conditions for least privilege.
* Vertex AI, GKE, and Cloud Run rely on service accounts scoped per module with Secret Manager delivering short-lived tokens.

## Security monitoring & intrusion detection

* Security Command Center Event Threat Detection + Chronicle SIEM monitor network, workload, and IAM anomalies.
* Pub/Sub triggers call Cloud Functions to quarantine compromised namespaces or revoke credentials automatically.

## Incident response & forensics

* Immutable log sinks and snapshot policies enable rapid evidence collection; automation raises PagerDuty incidents with context.
* Documented runbooks stored with the repo align with NIST-style triage, containment, eradication, and recovery steps.

## Automation / frameworks

* Python automation orchestrates terraform fmt/validate, OPA, Trivy, and Checkov locally and in CI.
* GitHub Actions uses reusable workflows and matrix jobs to test modules across environments.

# 6. Python Automation Example

#!/usr/bin/env python3  
import subprocess  
import json  
  
steps = [  
 ['terraform', 'fmt', '-check'],  
 ['terraform', 'validate'],  
 ['checkov', '-d', '.'],  
 ['opa', 'eval', '--data', 'policies', '--input', 'plan.json', 'data.security.allow'],  
 ['trivy', 'fs', '--exit-code', '1', '--severity', 'HIGH,CRITICAL', '.']  
]  
  
results = {}  
for cmd in steps:  
 completed = subprocess.run(cmd, capture\_output=True, text=True)  
 results[' '.join(cmd)] = {  
 'returncode': completed.returncode,  
 'stdout': completed.stdout,  
 'stderr': completed.stderr,  
 }  
 if completed.returncode != 0:  
 break  
  
print(json.dumps(results, indent=2))

# 7. Common Issues & Mitigations

Key architectural gaps we address proactively:

* Ingress bypass or misconfiguration: Enforce Cloud Armor policies and Terraform guardrails to block direct backend exposure.
* Lateral movement inside GKE: Apply workload identity, network policies, mutual TLS, Binary Authorization, and automated namespace quarantine.
* Secret sprawl: Centralize secrets in CMEK-protected Secret Manager; rotate via Python automation using short-lived tokens.
* Environment drift: Use Terraform CI/CD with approvals and devcontainer-standard tooling to avoid manual drift.
* Delayed incident response: Chronicle + Pub/Sub automation trigger runbooks; maintain evidence sinks and snapshot policies for forensics.

Prepared for discussions with an AI Security Lead Engineer—focus areas: network security, data security, governance, and applied automation.