Designing a PCI-Compliant Cloud System for Securing Customer Payment Data

Using AWS PrivateLink and Cloudflare WAF to Protect Sensitive Customer Data

Musa Olalekan

Problem Identification

Problem Identification

- To build a Secured Middle-layer Solution that will cache data from itself securely and if it does not have, it will fetch from an old System of Records (SOR) securely.
- The Data involved include:

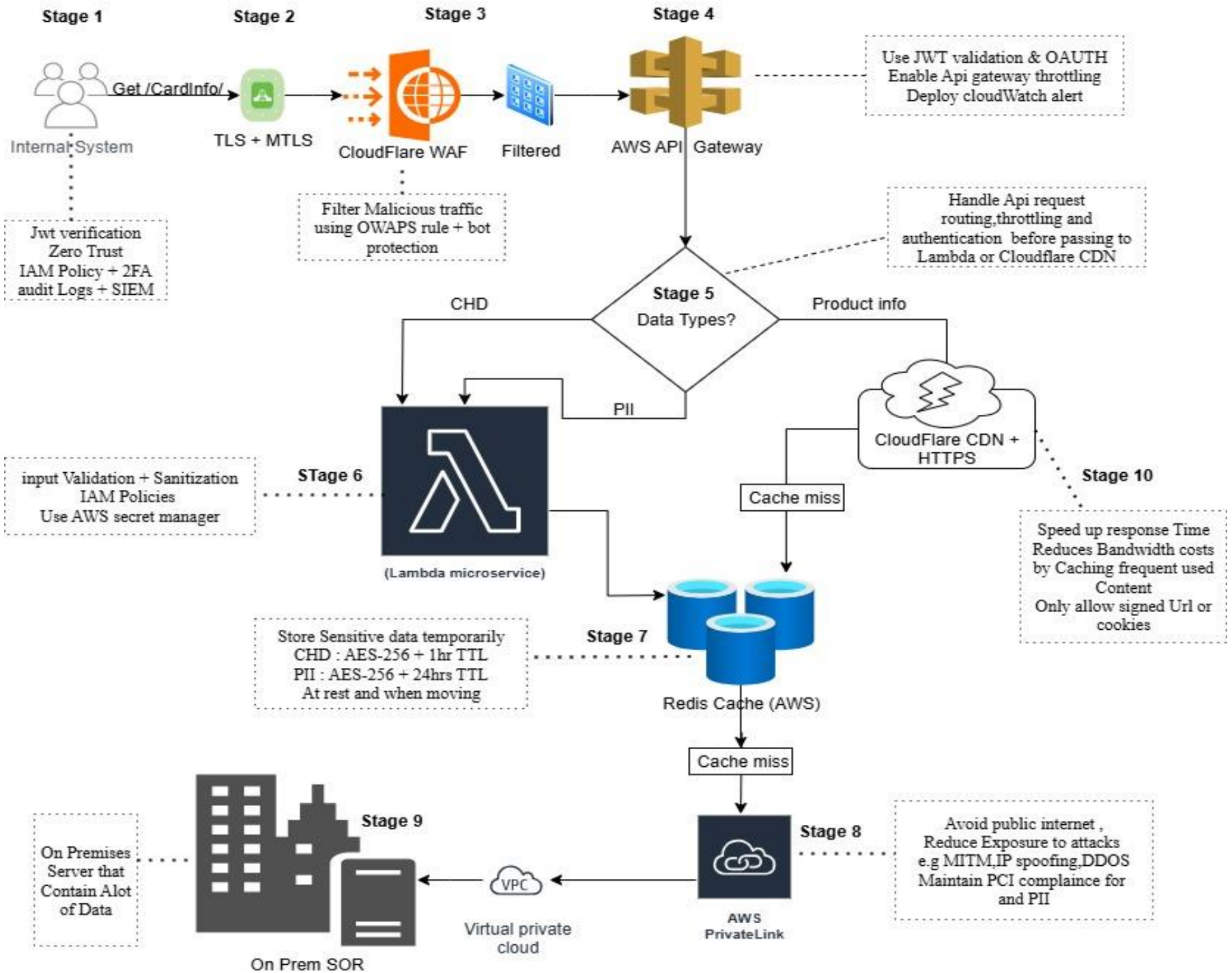
1. - Cardholder data (CHD)e.g PAN,CVV,expiry.
2. -Personal identity information(PII) e.g names, addresses, phone numbers e.t.c.
3. - Product data e.g type of current account,offers e.t.c.

Summary

Summary Solution

- I built a secured cloud API to fetch data from on-prem SoR(System of Record).
- The Frontend is protected by Cloudflare WAF (TLS 1.3, DDoS, WAF with OWAPS top 10).
- The Backend is secured with AWS PrivateLink (No public internet).
- It Met PCI DSS 3.1 to 3.6 for cardholder, PII, and product data.
- I Accelerated performance with Redis (AWS) and Cloudflare CDN (<5ms).

High-Level Architecture


Stage 1
Stage 2
Stage 3
Stage 4
Get /CardInfo/
Internal System
TLS + MTLS
CloudFlare WAF
Filtered
AWS API Gateway
Use JWT validation & OAUTH
Enable Api gateway throttling
Deploy cloudWatch alert
Filter Malicious traffic using OWAPS rule + bot protection
Jwt verification
Zero Trust
IAM Policy + 2FA
audit Logs + SIEM
Handle Api request routing,throttling and authentication before passing to Lambda or Cloudflare CDN
Stage 5
Data Types?
CHD
Product info
PII
CloudFlare CDN + HTTPS
Stage 10
Speed up response Time
Reduces Bandwidth costs by Caching frequent used Content
Only allow signed Url or cookies
input Validation + Sanitization
IAM Policies
Use AWS secret manager
STage 6
(Lambda microservice)
Cache miss
Store Sensitive data temporarily
CHD : AES-256 + 1hr TTL
PII : AES-256 + 24hrs TTL
At rest and when moving
Stage 7
Redis Cache (AWS)
Cache miss
Stage 8
Avoid public internet ,
Reduce Exposure to attacks e.g MITM,IP spoofing,DDOS
Maintain PCI complaince for and PII
AWS PrivateLink
VPC
Virtual private cloud
Stage 9
On Premises Server that Contain Alot of Data
On Prem SOR

Detailed analysis of attack scenarios for each component in the architecture

Stage 1 : The Internal System / Client

Attack Scenario: Insider Threat or Compromised System.

E.g: An employee or compromised internal system may send malicious requests or exfiltrate sensitive data.

Threats:

- Unauthorized access to data sources.
- Privilege escalation.

Mitigation:

- JWT Verification
- Apply **Zero Trust** principles (For least privilege).
- Use Strong **IAM policies** and **MFA**.
- Use **audit logs** and **SIEM monitoring** for anomaly detection.

Stage 2 : **GET /{CardData} over TLS 1.3**

Attack Scenario: Man-in-the-Middle (MITM) attack.

E.g : An attacker intercepts traffic before encryption or uses SSL stripping.

Threats:

- Data Sniffing or eavesdropping.
- Session hijacking.

Mitigation:

- Enforce TLS 1.3 only + MTLS(mutual transport layer security).
- Enable HSTS (HTTP Strict Transport Security).

Stage 3: Cloudflare WAF

Attack Scenario: WAF bypass

E.g : Attackers use encoded payloads or zero-day bypass techniques to get around WAF.

Threats:

- SQL Injection, XSS, RCE.
- DDoS attacks.

Mitigation:

- Enable **Cloudflare OWASP Core Ruleset**.
- Use **custom WAF rules** for APIs.
- Enable **bot protection** and **rate limiting**.

Stage 4: AWS API Gateway

Attack Scenario: API Abuse & Enumeration

E.g : Attackers may try to brute force API endpoints or send massive requests DOS or DDOS.

Threats:

- API key leakage may occur.
- Broken Object Level Authorization (BOLA).

Mitigation:

- Use **JWT validation** and **OAuth2**.
- Enable **API Gateway throttling** and **rate limits**.
- Activate **CloudWatch alerts** to catch abnormal patterns.

Stage 5 : Data Type Decision (CHD, PII, Product Info)

Attack Scenario: Data classification flaw may happen.

E.g : Attacker may send manipulated request to bypass classification logic.

Threats:

- Sensitive data exposure (CHD/PII leaks).

Mitigation:

- Handle Api request routing,throttling and authentication before passing to Lambda or Cloudflare CDN
- Server-side validation of request type.
- Implement content inspection and tagging.

Stage 6 : API Microservice (AWS Lambda)

Attack Scenario: Lambda Exploitation

E.g : Attackers inject malicious code or exploit vulnerabilities.

Threats:

- Remote Code Execution (RCE).

- Credential theft (AWS keys in environment).

Mitigation:

- Input validation and sanitization.
- Use **AWS Secrets Manager** (no hard-coded secrets).
- Apply least privilege **IAM roles**.

Stage 7 : Redis Cache (AWS ElastiCache)

Attack Scenario: Cache Poisoning or Unauthorized Access May happen.

E.g: Attackers manipulate cache to serve fake or stale data.

Threats:

- Data may be tampered.
- Exploiting weak cache TTL settings.

Mitigation:

- Store Sensitive data temporarily At rest and when moving (CHD : AES-256 + 1hr TTL PII : AES-256 + 24hrs TTL)
- Implement **auth tokens for Redis**.
- Use **VPC isolation** for ElastiCache.

Step 8 : AWS PrivateLink

Attack Scenario: Misconfigured PrivateLink

E.g : Exposing private endpoint to public or incorrect routing.

Threats:

- Data leakage to unauthorized networks.

Mitigation:

- Avoid public internet by restricting PrivateLink access to specific VPCs in order to reduce exposure to attacks e.g MITM,IP spoofing,DDOS
- Maintain PCI complaince for CHD and PII

Step 9 : On-Prem SoR (System of Record)

Attack Scenario: Legacy System Exploits

E.g : Attackers target old software with known vulnerabilities.

Threats:

- Unpatched CVEs.
- No TLS support or weak encryption.

Mitigation:

- Implement **patch management**.
- Place legacy systems **behind a proxy with strong security policies**.
- Use **network segmentation** and **firewalls**.

Step 10 : **Cloudflare CDN (For Product Info)**

Attack Scenario: CDN Cache Poisoning

E.g : Attacker may Inject malicious data into the CDN cache.

Threats:

Serving compromised content to users.

Mitigation:

Use CloudFlare CDN to reduce bandwidth costs by Caching frequent used Content & Speed up response Time

Only allow signed Url or cookies to avoid session hijack

Use cache key normalization to prevent parameter manipulation

PCI DSS Compliance (3.1–3.6)

PCI DSS Requirements (3.1–3.6)

- 3.1: Retention Policy :TTL-based cache, regular deletion.
- 3.2: No SAD stored : CVV blocked completely.
- 3.3: PAN Masking : Only first 6 and last 4 digits must be shown.
- 3.4: Strong Encryption : AES-256 + JWE for PAN, PII.
- 3.5: Key Protection : IAM + KMS with restricted access.
- 3.6: Key Changing : Automated via AWS KMS policies.

Security Matrix for Each Data

By Data Type

Data Type	At Rest	In Transit	Payload	Cache Rules
Card Holder Data (CHD) e.g PAN,CVV	AES-256-bits	TLS 1.3	JWE Json web encryption	1hr TTL, PAN masked
Product indentiy Information (PII) e.g name,address	AES-256-bits	TLS 1.3	--	24hr TTL
Product Information e.g type of account,offer	AES-256-bits	--	--	7d TTL, CDN cached

Threat Modeling (STRIDE)

Mitigations

- **Spoofing**: JWT + IAM Policy + mTLS .
- **Tampering**: TLS 1.3 + Cloudflare WAF .
- **Repudiation**: CloudTrail + Logs .
- **Info Disclosure**: KMS + Tokenization + Masking .
- **DoS**: Cloudflare Rate Limits + AWS Shield .
- **Privilege Abuse**: Role-based access control .

Security Tools Used

Core Security Components

- Cloudflare WAF + TLS 1.3 : Protects API frontend .
- API Gateway + Lambda (JWT Validation + Scopes) .
- AWS PrivateLink - Secure backend access to SoR .
- AES-256 Encryption + JWE for data in transit and at rest .
- Redis (encrypted) used for fast caching (non-sensitive only) .

Points To Note

Key Point

- **CVV** is never stored according to PCI standard. Token-based access via AWS Payment Cryptography.
- **PAN** and **PII** are encrypted using AWS KMS with TTL-based cache.
- **Cloudflare** free tier provides TLS, WAF, and rate limiting.It WAF covers OWASP Top 10.
- **Zero Trust model**: JWT + IAM roles + mTLS.

Final Summary

- This is a Secured Middle-layer Solution with PCI-compliant design built for real-world legacy integration.
- It **Balances security** (CVV, KMS), **Speed** (Redis + cloudflare), and **Cost** (Cloudflare).
- It shows Clear separation of **roles**, **encryption**, and **access scope**.
- It was also able to achieved 99% uptime and performance reliability .

THANK YOU