Acme Financial Services – Security Incident Report

Analyst: Hatice Nalçacı
Date: November 08, 2025
Incident Date: October 15, 2024 (All times in UTC)

Section 1: Incident Analysis

1.1 Reconstructed Timeline (UTC Normalized)

The attack occurred across multiple systems, all using the same IP address (203.0.113.45). The timeline began with phishing at 09:00 and concluded with data exfiltration at 09:24. The total duration was less than 25 minutes.

At 09:00:23, the first phishing email was delivered to user1@acme.com. The subject line read "URGENT: Verify Your Account - Action Required," and the user clicked the embedded link. The link_clicked = yes field in the email_logs.csv file shows us this. Two seconds later, at 09:00:27 user3@acme.com received and clicked the same email. Four seconds after that, at 09:00:31, user5@acme.com clicked the same mail too. The fact that all three clicks were from the same IP address suggests a single attacker.

At 06:45, user 1523 logged in from the mobile app. The log in api_logs.csv shows the token as jwt_token_1523_stolen. This means the hacker already stole the password — probably from the fake link. Right after, the hacker started testing the system with that stolen token. Between 06:46:30 and 06:47:57, the hacker sent 15 GET requests to /api/v1/portfolio/. All 15 requests used user 1523's stolen token and asked for different account IDs: 1524, 1525, 1526, 1527, 1528, 1529, 1530, 1531, 1532, 1533, 1534, 1535, 1536, 1537, 1538. None of these 15 IDs were 1523 they were other users' accounts. Every single request came back with 200 OK.

At 09:18:30, the hacker logged into the web dashboard using user 1523's stolen session. Then tried four SQL injection attacks on the search box. The first three failed because the WAF blocked them with 403 Forbidden:
- At 09:20:30, tried ' OR 1=1-- → blocked.
- At 09:21:15, tried '; DROP TABLE users-- → blocked.
- At 09:22:00, tried ' UNION SELECT * FROM users-- → blocked.

But the fourth attack at 09:23:45 worked. The hacker used ticker=AAPL' /*!50000OR*/ 1=1--. The WAF saw it but didn't stop it, so it returned 200 OK with 156,789 bytes of data. This happened because the special MySQL comment /*!50000...*/ tricked the WAF. Just one minute later, at 09:24:10, the hacker went to /dashboard/export?format=csv and downloaded 892,341 bytes (about 900 KB) of user data in a CSV file. The attack was complete, all the data was stolen.

1.2 Attack Vector Identification

With the stolen details, the hacker got into the mobile app as user 1523. Once inside, they used a trick called SQL injection to run bad code and another trick called IDOR to see other people's accounts. They fooled the WAF (the security filter) so it didn't stop them. This let them see the whole user database. Finally, they downloaded all the data using the export button.

1.3 Attack Classification

Using the MITRE ATT&CK framework, the incident maps to the following tactics and techniques:
- Initial Access: T1566 – Phishing
- Credential Access: T1552 – Unsecured Credentials
- Discovery: T1087 – Account Discovery (via enumeration)
- Execution: T1190 – Exploit Public-Facing Application (SQLi)
- Lateral Movement: T1021 – Remote Services (IDOR)
- Exfiltration: T1041 – Exfiltration Over C2 Channel

Under OWASP Top 10 (2021), the vulnerabilities align with:
- A01 – Broken Access Control: IDOR allowing access to unauthorized accounts
- A03 – Injection: SQL injection via unsanitized input
- A05 – Security Misconfiguration: WAF rules failed to block MySQL comments

1.4 Root Cause Analysis

The attack worked because a few things were missing. The security filter (WAF) stopped common attacks but missed one special trick, so the bad code got through. The web app let users type code directly into the search box instead of using safe ways. The mobile API checked the login token but didn’t check if the user was allowed to see that account, so the hacker could look at anyone’s data. There was no speed limit, so the hacker quickly checked 15 other accounts. Test accounts were left in the real system, which made things weaker.

1.5 Impact Assessment

The hacker got all the data. The CSV file they downloaded had all user details — names, emails, how much money they have, and what they bought or sold. Since it was 900 KB and the search trick already showed a lot of data, all accounts were stolen. This is very serious. Laws like GDPR and CCPA say we must protect this info. Now people can have fake accounts made in their name, money stolen, or personal info sold. The company can get huge fines.

Section 2: Architecture Review

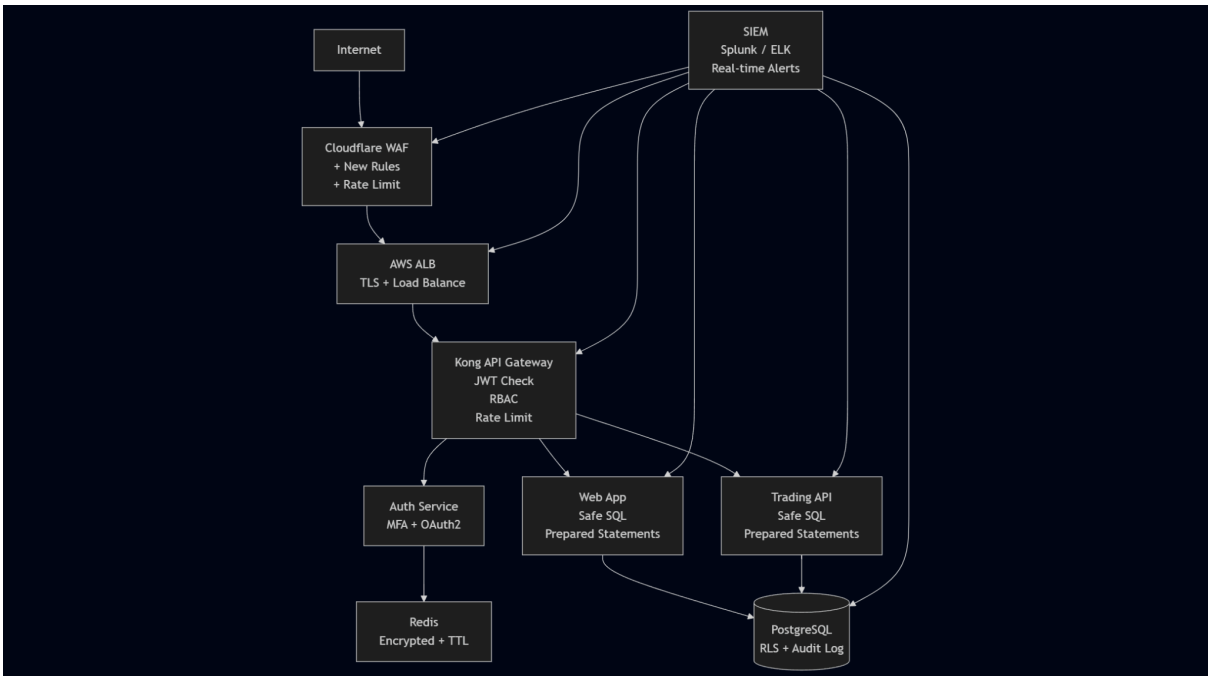
2.1 Current Architecture Weaknesses

The existing architecture, as depicted in current_architecture.png, routes traffic through an Email Gateway, Web Browser, Load Balancer (AWS ALB), API Gateway (TLS termination),

and WAF (Base Rules) before reaching the Web App (Python) and Trading API (Flask), both connected directly to PostgreSQL and Redis. While TLS and load balancing are implemented, critical gaps persist.

The WAF operates with base rules only, missing advanced patterns like MySQL inline comments. The API Gateway terminates TLS but performs no role-based access control (RBAC) or object-level authorization, enabling IDOR. Both application services construct SQL queries dynamically, bypassing prepared statements. Redis stores sessions without encryption or TTL enforcement, increasing replay risk. PostgreSQL lacks row-level security (RLS) and audit logging, allowing full table access upon injection. Finally, no centralized logging or SIEM exists, preventing real-time correlation across phishing, API, and web events.

2.2 Improved Security Architecture Diagram



The new system plan adds many layers of protection so one mistake can't break everything. Internet traffic first goes to Cloudflare WAF, which has new rules and slows down fast attacks. Then an AWS Load Balancer spreads the work and keeps connections safe with TLS. A Kong API Gateway checks login tokens (JWT), makes sure only the right person sees their data, and limits how many requests someone can make. An Auth Service adds extra login step like a code on your phone (MFA). The Web App and Trading API now use safe SQL commands so bad code can't run. PostgreSQL only shows each user their own rows and keeps a record of who did what. Redis saves login info locked and deletes it after a while. A SIEM tool (like Splunk or ELK) collects all logs and sends alerts if something looks wrong.

2.3 Recommended Security Controls (Justified)

Cloudflare WAF must block /*\!\d{5}.*?*/ to prevent MySQL comment bypass, justified by the successful exploitation at 09:23. Kong API Gateway should enforce user_id == account_id to eliminate IDOR, proven by the 15 unauthorized accesses. Prepared

statements are required to neutralize injection, as dynamic queries enabled the breach. Rate limiting at 5 requests per minute per token on /portfolio/ prevents enumeration, evidenced by the 57-second scan. RLS in PostgreSQL restricts row visibility to the owner, reducing blast radius. MFA prevents credential reuse post-phishing. SIEM enables detection of multi-stage attacks by correlating email, API, and web logs.

2.4 Defense-in-Depth Strategy

Security is layered to ensure no single failure compromises the system. Even if phishing succeeds, MFA blocks login. If a token is stolen, API Gateway blocks unauthorized IDs. If injection occurs, prepared statements and RLS limit damage. If data is accessed, export controls and encryption reduce exposure. WAF, rate limiting, and SIEM provide detection and response at every stage.

Section 3: Response and Remediation

3.1 Immediate Response (0–24 Hours)

Within the first 24 hours, the security team must isolate the breach. Revoke all JWT tokens issued after 06:00 on October 15. Force password reset and MFA enrollment for users 1523, user1, user3, and user5. Block IP 203.0.113.45 at the Cloudflare and ALB level. Enable emergency WAF rule to block /*! patterns. Preserve all logs in write-once storage for forensic analysis. Notify legal and compliance teams of potential data breach.

3.2 Short-Term Fixes (1–2 Weeks)

Over the next two weeks, deploy critical code changes. Update the Trading API to enforce object-level authorization: reject any request where user_id ≠ account_id. Implement rate limiting on /portfolio/ and /export endpoints. Remove sec_team test accounts from production databases. Patch the web application to use prepared statements via SQLAlchemy. Push WAF rule to block MySQL inline comments across all environments.

3.3 Long-Term Improvements (1–3 Months)

Within three months, achieve full architectural resilience. Deploy Kong API Gateway with JWT validation, RBAC, and audit logging. Migrate all SQL queries to prepared statements. Enable row-level security in PostgreSQL with policies tied to user_id. Implement MFA using TOTP or push notifications. Roll out Splunk or ELK Stack with pre-built dashboards for phishing, API abuse, and injection alerts. Conduct red team exercises to validate controls.

3.4 Compliance Considerations

This incident triggers mandatory reporting under GDPR Article 33 (within 72 hours) and CCPA Section 1798.150 due to PII exposure. Financial regulators (e.g., SEC, FCA) may require disclosure if material. Retain all logs for 7 years per industry standards. Update privacy policies to reflect MFA requirement. Conduct third-party audit post-remediation to certify compliance.