Security Incident Analysis Report

Acme Financial Services Trading Platform Breach

Security Analyst Candidate

November 8, 2024

Abstract

This report covers my analysis of a security attack that happened at Acme Financial Services on October 15, 2024. I looked at the logs and found three main problems: phishing emails, SQL injection, and broken API security. In this report, I'll explain what happened, when it happened, and what we can do to fix it and stop it from happening again.

1 Incident Analysis

1.1 Executive Summary

After looking at the logs from October 15, 2024, I found that someone attacked Acme Financial Services and got into 15 customer accounts. What really got my attention was how they used three different tricks together: phishing emails, SQL injection, and API bugs.

Here's what I found:

- **Critical:** The API doesn't check if you're allowed to see someone else's account - it just lets you in
- **Critical:** The attacker got past the firewall using a special MySQL trick
- **High:** 3 out of 6 people clicked the phishing link (that's really high)
- **Medium:** The firewall needs better rules to catch new attack methods

1.2 Attack Timeline - What Happened and When

I looked at four different log files to figure out what happened. All times are in UTC.

09:00:23 UTC - Phase 1: The Phishing Emails

First, I saw some weird emails in the logs. The sender used `acme-finance.com` instead of the real `acme.com`. Pretty smart trick - the domains look almost the same. The subject said "URGENT: Verify Your Account - Action Required" to make people panic and click fast. And it worked - 3 out of 6 people clicked it. That's bad. All three clicks came from IP `203.0.113.45`. Remember this IP, it shows up again later.

09:18:30 UTC - Phase 2: Getting In

Only 18 minutes later, I saw user 1523 login to the system. But look at the IP - it's `203.0.113.45` again! Same as the phishing clicks. So now I know for sure: they got the password from the fake website.

09:20:30 - 09:23:45 UTC - Phase 3: The SQL Injection

Two minutes after getting in, the attacker started trying SQL injection. I found 4 attempts. First three got blocked - stuff like `' OR 1=1--` and `DROP TABLE`. The firewall caught those. But then - and this is the interesting part - they tried `/*!50000OR*/`. This is a MySQL trick that the firewall didn't know about. And it worked! Got 156KB of data (way more than normal). One minute later, they downloaded 892KB as a CSV file. So the firewall stopped the easy attacks but missed this one.

06:46:30 - 06:47:57 UTC - Phase 4: The API Problem

Here's something interesting - this attack actually happened earlier, before the SQL stuff. User 1523 logged in at 06:45, then started looking at other people's accounts: 1524, 1525, 1526, and so

on. Every 3 seconds. Obviously a script doing this automatically. And here's the worst part - every request said "200 OK". The API checked if the user was logged in, but it never checked "are you allowed to see THIS account?". So basically, if you're logged in, you can see anyone's data. This is called IDOR (broken access control). They looked at 15 different customer accounts this way.

1.3 Attack Vector Classification

MITRE ATT&CK Framework Mapping:

- **T1566.002** - Phishing: Spearphishing Link
- **T1078** - Valid Accounts: credential harvesting via phishing
- **T1190** - Exploit Public-Facing Application: SQL injection
- **T1213** - Data from Information Repositories: database exfiltration
- **T1087.004** - Account Discovery: API account enumeration
- **T1530** - Data from Cloud Storage Object: API data exfiltration

OWASP Top 10 Classification:

- **A01:2021 - Broken Access Control:** API IDOR vulnerability (primary)
- **A03:2021 - Injection:** SQL injection with WAF bypass
- **A07:2021 - Identification and Authentication Failures:** credential phishing

1.4 Why This Happened - Root Causes

1. The API Bug (Biggest Problem)

When I looked at the API code, I found it only checks "are you logged in?" but not "are you allowed to see this account?". The fix is actually simple - just add one line that says "if your user ID doesn't match the account ID, no access". That's it.

2. SQL Injection Problem

The web app puts SQL queries together using simple text joining (like "SELECT * FROM users WHERE id=" + userId). This is dangerous. It should use safe query methods instead. Also, the firewall doesn't know about MySQL special tricks like `/*!50000OR*/`, so it missed the attack.

3. Email Security Missing

We don't have SPF, DKIM, or DMARC set up. These are like ID checks for emails. Without them, anyone can pretend to be from our domain.

1.5 What Got Damaged - Impact

They got into 15 customer accounts and saw all their portfolios, transactions, and personal info. They also downloaded 892KB of database stuff.

For the business, this is bad news. We have to tell customers what happened (required by law). If any EU customers were affected, we might get fined up to €10 million under GDPR rules. But honestly, the bigger problem is trust. People trust us with their money - if they think we can't protect their data, they'll leave. We know for sure that at least 1 password was stolen (maybe 3 from the phishing emails).

2 Architecture Review

2.1 Current Security Problems

When I look at the architecture diagram, I see several issues: the apps talk directly to the database (that's why SQL injection works), the firewall only has basic rules, there's nothing checking emails, and the API doesn't verify permissions properly. Also, there's no way to detect when someone is making lots of requests really fast (which would have caught the API attack).

2.2 Better Security Design

I recommend building five layers of security. Think of it like a castle with multiple walls:

Layer 1 (Outside): Email checker for phishing, blocks DDoS attacks

Layer 2 (Gateway): Better firewall that knows modern attacks, limit how many requests per minute, API gateway

Layer 3 (Apps): Use safe SQL methods, check permissions before showing data, require two-factor authentication

Layer 4 (Database): Firewall for database, encrypt all stored data

Layer 5 (Watching): System that watches for weird behavior, security team available 24/7

This way, if one layer fails (like the firewall), the others still protect us.

2.3 What We Need to Do

Right Now (today): Block the attacker's IP (`203.0.113.45`), make everyone change passwords, cancel all login tokens, fix firewall to catch MySQL tricks, turn on more detailed logging.

This Month: Fix that API bug (add the permission check I mentioned), make SQL queries safe, limit API to 10 requests per minute, set up email security (SPF/DKIM/DMARC), turn on two-factor authentication, teach people how to spot phishing.

Next Few Months: Switch to safer SQL methods (ORM), add API gateway, set up monitoring system, do regular security tests, work on getting SOC 2 certification.

3 Response & Remediation

3.1 How to Respond

We need to follow standard steps: first, stop the damage (block the attacker, disable compromised accounts). Second, remove the vulnerabilities. Third, bring systems back online carefully with extra monitoring. Finally, have a meeting to learn what went wrong and update our procedures.

3.2 Legal Stuff

If any EU customers were affected, we have to tell regulators within 72 hours (GDPR law). If credit card data was touched, we need to tell the credit card companies too. I'd say get a lawyer involved now and write down everything we're doing to fix this. Even if we're not legally required to tell customers, I think we should anyway - being honest helps keep their trust.

4 Final Thoughts

What I learned from this investigation: one small bug can turn into a big problem when you have other security holes too. The firewall stopped some attacks but not all - and that's why we need multiple layers of protection.

If I had to pick the most important thing to fix, it's that API bug. Seriously, just one line of code would have stopped 15 accounts from getting hacked. The SQL injection needs more work to fix properly, but we need to do both.

This whole situation reminded me of something I've noticed in security - your defenses need to work as a team. If each one works alone, attackers will find the weak spots. That's why I suggested the five-layer setup, following what OWASP and NIST recommend.

Everything in this report comes from the log files they gave me. I used standard methods to analyze them.