REPORT

CyberSecurity log system

Mohammed Derkaoui &
Anass Afkir

Table of Contents

Problematic

Security incidents such as brute force attacks, unauthorized access attempts, and data breaches occur daily, generating massive amounts of log data. The main challenges organizations face include:

1 Overwhelming Log Volume

2 Incident Response Delays

without a way to detect threats instantly the response team gets delayed for quite some time which allow the intruders to install malware, modify the db, steal data...

3 Pattern Recognition Difficulty

4 Lack of Automation

Solution

Our Cybersecurity Log & Incident Management System offers a platform with multiple solutions such as:

1 **Automated Log Collection**

storage of security events with timestamps, IP addresses, and event types

2 **Intelligent Anomaly Detection**

Multiple algorithms detect brute force attacks, account compromise attempts, login spikes, and suspicious patterns

2 **Automated Incident Management**

Automatic incident creation when threats are detected, with severity levels and detailed descriptions

2 **Real-Time Monitoring**

Continuous analysis of incoming logs with trigger-based alerts

2 **Comprehensive Reporting**

Weekly summaries showing security trends and incident statistics

2 **Search & Filtering System**

System Architecture

Backend	Database Features	Python Libraries
• Python 3.x - Core application logic • MySQL 8.0 - Database and storage	• Foreign key constraints for data integrity • Indexed columns for query performance • Stored procedures for complex operations • Triggers for automated responses • Views for simplified queries	• mysql-connector-python - Database connectivity • Built-in libraries for data processing

Database Design

3.1 Database Schema

The system uses 2 main tables with proper relationships:

Table 1: logs

Stores all security events from monitored systems:

```
-- ------------------------------------------------------------------------------
-- TABLE 2 : logs
-- Purpose: record all activity logs (login attempts, suspicious events, etc.)
-- ------------------------------------------------------------------------------

create table logs (
  log_id int auto_increment primary key,                -- unique log entry
  event_time datetime not null,                          -- when event occurred
  username varchar(50) default null,                    -- user involved (optional)
  ip_address varchar(50) default null,                   -- supports IPv4 + IPv6
  event_type varchar(45) not null,                      -- e.g. 'login_failed'
  severity enum('low','medium','high') default 'low',   -- quick risk level
  message text,                                         -- full log content
  created_at datetime default current_timestamp,        -- when inserted
  foreign key (username) references users(username)
      on delete set null on update cascade
  );
```

- Stores all security events from monitored systems.
- event_time
- ip_address: Source IP
- event_type: Example: login_success, login_failed, access_denied...
- severity

Table 2: incidents

Tracks security incidents requiring investigation:

```
-- ------------------------------------------------------------------------------
-- TABLE 3 : incidents
-- Purpose: record security incidents, often linked to one or more logs
-- ------------------------------------------------------------------------------

create table incidents (
  incident_id int auto_increment primary key,                    -- unique incident
  log_id int default null,                                       -- reference to related log
  title varchar(200) not null,                                   -- short incident summary
  description text,                                              -- detailed context
  status enum('open','investigating','resolved') default 'open', -- workflow stage
  severity enum('low','medium','high','critical') default 'medium',
  reporter varchar(50) default 'system',                         -- who created the incident
  created_at datetime default current_timestamp,
  updated_at datetime default current_timestamp on update current_timestamp,
  foreign key (log_id) references logs(log_id)
      on delete set null on update cascade
  );
```

Purpose: Track security incidents from detection to resolution:

- status
- severity - Priority level
- reporter - Who/what created the incident (system vs analyst)

3.2 Database View

View: active_incidents

Provides quick access to unresolved security incidents.

```
-- ------------------------------------------------------------------------------
-- VIEW
-- Purpose: simplified view for active incidents (for dashboard)
-- ------------------------------------------------------------------------------

create or replace view active_incidents as
select
    incident_id, title, severity, status, created_at
    from incidents
    where status != 'resolved';

-- explanation:
-- * Makes dashboard queries cleaner.
-- * Demonstrates MySQL VIEW usage.
-- * Simplifies queries. every time, you just do: select * from active_incidents;
```

Purpose: can quickly see all incidents requiring attention.

3.3 Stored Procedure

Procedure: weekly_summary()

Generates weekly security statistics for reporting.

```
DELIMITER //
DROP PROCEDURE IF EXISTS weekly_summary;
CREATE PROCEDURE weekly_summary()
BEGIN
    SELECT
        DATE(l.event_time) AS date,
        COUNT(*) AS total_logs,
        SUM(l.event_type = 'login_failed') AS failed_logins,
        COUNT(DISTINCT i.log_id) AS total_incidents
    FROM logs l
    LEFT JOIN incidents i ON DATE(i.created_at) = DATE(l.event_time)
    GROUP BY DATE(l.event_time)
    ORDER BY date DESC
    LIMIT 7;
END;
//
DELIMITER ;
```

Purpose: Show logs for the last 7 days.

3.4 Database Trigger

Trigger: trg_auto_incident

Automatically creates incidents when attack patterns are detected.

```
-- ------------------------------------------------------------------------------
-- TRIGGER
-- Purpose: auto-create incident on repeated failed logins
-- ------------------------------------------------------------------------------

delimiter //
create trigger trg_auto_incident
after insert on logs
for each row
begin
    declare fail_count int;

    if new.event_type = 'login_failed' then
        select count(*) into fail_count
        from logs
        where ip_address = new.ip_address
            and event_type = 'login_failed'
            and event_time > now() - interval 10 minute;

        if fail_count >= 3 then
            insert into incidents (log_id, title, description, severity, reporter)
            values (new.log_id, 'Multiple login failures detected', concat('IP:', new.ip_address, ' had ', fail_count, ' failed
attempts'), 'high', 'system');
                end if;
        end if;
end;
//
delimiter ;
```

Application Features

4.1 CRUD Operations

CRUD = Create, Read, Update, Delete - the fundamental operations for data management.

Logs CRUD (models/log_model.py)

Stores all security events from monitored systems:

CREATE - Insert New Logs:

- def insert_log(ip_address, username, event_type, message, severity='low')

READ - Retrieve Logs:

- def get_all_logs()
- def get_log_by_id(log_id)
- def search_logs(ip_address, username, event_type, severity, start_date, end_date)

UPDATE - Modify Logs:

- def update_log(log_id, message)

DELETE - Remove Logs:

- def delete_log(log_id)

Incidents CRUD (models/incident_model.py)

CREATE:

- def create_incident(title, description, severity, log_id, reporter)

READ:

- def get_all_incidents()
- def get_incident_by_id(incident_id)
- def search_incidents(status, severity, title_keyword, start_date, end_date)

UPDATE:

- def update_incident_status(incident_id, new_status)
- def update_incident_severity(incident_id, new_severity)

DELETE:

- def delete_incident(incident_id)

Benefits of CRUD Architecture:

- Database logic isolated
- Functions used across all features
- Easy to update database operations
- Each function can be tested independently

4.2 Anomaly Detection System

The system implements **4 detection algorithms** that analyze logs.

Algorithm 1: Brute Force Detection

Purpose: Detect repeated failed login attempts from same IP address

- def detect_brute_force_attacks(time_window_minutes=60, threshold=5)

Algorithm 2: Account Compromise Detection

Purpose: Detect one IP trying multiple different user accounts

- def detect_account_compromise_attempts(time_window_minutes=30, threshold=3)

Why This Matters:

- Attacker doesn't know valid usernames
- Different pattern than brute force (one user, many attempts)

Algorithm 3: Login Spike Detection

Purpose: Detect unusual surge in login activity

- def detect_login_spikes(time_window_minutes=10, spike_threshold=20)

Why This Matters:

- Could indicate: Bot attack, system malfunction

Algorithm 4: High-Severity Event Clustering

Purpose: Detect multiple high-severity events occurring together

- def detect_high_severity_events(time_window_hours=24)

Automated Incident Creation

When anomalies are detected, the system automatically:

- Creates incident in database
- Generates descriptive title
- Adds detailed description with IPs, counts, timestamps
- Sets appropriate severity level
- Marks reporter as "anomaly_detection_system"

TESTING

The system includes testing:

Test Files	Purpose	Test Results
test_CRUD_incidents_logs.py	Tests all CRUD operations	✅ PASSED
test_incidents_table.py	**Tests database connectivity**	✅ **PASSED**
test_trigger_only.py	Validates automatic trigger functionality	✅ PASSED
test_anomaly_detection.py	**Validates all detection algorithms**	✅ **PASSED**

User Interface

Command-Line Interface (CLI)

The system provides an intuitive text-based menu interface:

```
============================================================
 CYBERSECURITY LOG & INCIDENT MANAGEMENT SYSTEM
============================================================

📋 MAIN MENU:
  1. Add New Log Entry
  2. View All Logs
  3. Search/Filter Logs
  4. View All Incidents
  5. Search/Filter Incidents
  6. Update Incident Status
  7. Generate Weekly Report
  8. Run Anomaly Detection
  0. Exit
------------------------------------------------------------
Enter your choice:
```

Features:

- Clean, organized menu structure
- Numbered options for easy navigation
- Real-time feedback and error messages
- Formatted table output for data display
- Input validation and error handling

Conclusion

This project delivers cybersecurity log and incident management system with:

- Automated Threat Detection
- anomaly detection algorithms
- Real-time Monitoring
- Database triggers for immediate response
- CRUD operations with search/filter
- Reporting
- test coverage for all components

7.2 Technical Skills Demonstrated

- Database design
- SQL stored procedures, triggers, and views
- Python backend development
- CRUD operations
- Anomaly detection algorithms
- Software testing methodologies
- Version control with Git
- Technical documentation

7.3 Future Enhancements

Potential improvements for production deployment:

- Web-based dashboard with visualizations
- Email/SMS alerts for critical incidents
- Machine learning for adaptive threat detection
- Multi-user authentication and authorization
- Conversion to csv files

8. REFERENCES

- MySQL 8.0 Documentation - https://dev.mysql.com/doc/
- Python mysql-connector Documentation

Project Repository: https://github.com/codebyderkaoui/CyberSecurity-log-system
Authors: [Mohammed Derkaoui, Anass Afkir]
Date: 2025
Course: [Python et DB avancees]