2025

Cybersecurity Incident Response Playbook

Maceer.com

Table of Contents

- **About this Playbook**
- **Potential Threats**

  - SQL Injection Attacks

  - Cross-Site Scripting (XSS)

  - Denial of Service (DoS/DDoS)

  - Unauthorized Access / Credential Stuffing

  - Data Corruption or Data Loss

  - Server Downtime / High Load

  - Misuse of Admin Privileges

  - Broken Access Controls

- **Backup & Recovery Plan**
  1. Overview
  2. Backup Scope
  3. Backup Frequency
  4. Backup Storage & Security
  5. Recovery Procedures
     a. Detection & Assessment
     b. Containment
     c. Restoration
     d. Post-Recovery Actions
  6. Testing & Validation
  7. Responsibilities
  8. Continuous Improvement

About this Playbook

This Cybersecurity Incident Response Playbook is designed specifically for the college web portal project developed as a college graduation project. Its purpose is to help the development team, administrators, and security leads respond quickly and efficiently to cybersecurity incidents that may affect the platform's availability, integrity, and confidentiality.

The playbook outlines:

- Key risks and threats relevant to a student-focused academic web system
- Actionable steps for identifying, containing, and resolving incidents
- Roles and responsibilities during security events
- Recommended tools and practices for prevention and monitoring

By following this playbook, the project team can ensure that the platform remains secure, responsive, and reliable across all user groups (students, instructors, and administrators). It is a living document and should be reviewed and updated after every major incident or infrastructure change.

SQL Injection Attack

🔍 Detection – How to detect SQL Injection:

1. Unusual Query Patterns:
   - Look for SQL keywords like 'SELECT', 'DROP', '--', 'OR 1=1' inside input fields (e.g., search, login).
   - Monitor logs for repeated failed queries or sudden errors that reveal table names or SQL syntax.

2. Web Application Logs:
   - Laravel logs (usually in storage/logs) may show malformed queries or suspicious inputs.
   - Use a WAF (Web Application Firewall) or security tools like OWASP ModSecurity to detect injection patterns.

3. Database Behavior:
   - Unexpected data changes or unexplained query slowdowns could indicate tampering.

4. Security Scanners:
   - Use automated tools like OWASP ZAP or SQLMap to simulate attacks and identify vulnerabilities.

🛡️ Response – How to respond to an SQL Injection incident:

1. Containment:
   - Immediately block the affected endpoint or user account (IP-based rate limiting/firewall).
   - Switch the app to maintenance mode temporarily if needed to prevent further damage.

2. Investigation:
   - Review logs to identify the origin (IP address, endpoint used, input sent).
   - Check if any data was leaked, altered, or deleted.
   - Identify the vulnerable code that allowed the injection.

3. Eradication:
   - Fix the vulnerable code: always use Laravel's Eloquent ORM or query builder (which uses prepared statements).
   - Sanitize and validate all user input.
   - Disable error messages that reveal SQL syntax or table structure in production.

4. Recovery:
   - Restore affected data from backups if needed.
   - Run full database and application integrity checks.

5. Post-Incident Actions:
   - Log the incident and update the Playbook.
   - Conduct a security audit of all input fields and queries.
   - Train developers on secure coding practices (OWASP Top 10).

Cross-Site Scripting (XSS)

🔍 Detection – How to detect XSS:

1. Unexpected Script Execution:
   - Users report strange behaviors like unexpected pop-ups, redirections, or auto-filled forms.
   - Suspicious content appears in comment boxes, feedback forms, or URL parameters.

2. Security Logs:
   - Laravel logs may include suspicious input such as <script>, <img src=...>, onerror=, javascript:, etc.
   - Look for encoded characters like <, >, ', or URL-encoded payloads (%3Cscript%3E).

3. Security Tools:
   - Use tools like OWASP ZAP, Burp Suite, or Acunetix to scan for XSS vulnerabilities.
   - Browser-based content security policy (CSP) violation reports can also be configured to alert on suspicious inline scripts.

4. Monitoring User Reports:
   - XSS is often discovered by user reports when UI behavior is abnormal or pages are defaced.

🛡️ Response – How to respond to an XSS incident:

1. Containment:
   - Disable or restrict the affected page/feature.
   - Revoke any sessions potentially hijacked by a malicious script (especially if cookies were stolen).
   - Invalidate user tokens if there's suspicion of session compromise.
2. Investigation:
   - Identify the input field or endpoint that allowed script injection.
   - Review logs to find the attacker's IP, payload, and targeted users.
   - Check if any user data was exposed or if scripts executed account takeovers.
3. Eradication:
   - Sanitize and escape all dynamic content rendered in HTML using Laravel's Blade templating (which escapes output by default).
   - Implement proper input validation on all user-submitted data.
   - Set a strong Content Security Policy (CSP) to block inline scripts.
   - Disable unnecessary HTML tags (like <script>, <iframe>, etc.) in user input fields (e.g., through a WYSIWYG editor with sanitization).
4. Recovery:
   - Inform affected users if account compromise is suspected.
   - Restore any defaced or corrupted content.
   - Run a full code and database audit to ensure no other vectors exist.
5. Post-Incident Actions:
   - Update secure coding standards for the team.
   - Conduct a full pentest of all input fields and API endpoints.
   - Add security headers: Content-Security-Policy, X-XSS-Protection (deprecated, but still relevant for legacy browsers), and X-Content-Type-Options.

Denial of Service (DoS/DDoS)

🔍 Detection – How to detect a DoS/DDoS Attack:

1. Sudden Spike in Traffic:
   - Sharp increase in the number of requests per second, especially from a single IP or region.
   - Unusual load on server resources (CPU, RAM, bandwidth usage).

2. Service Unavailability:
   - Website becomes slow or unresponsive.
   - Certain endpoints fail or timeout repeatedly.

3. Log Patterns:
   - Repetitive requests to the same route (e.g., /login, /register, or /api).
   - Requests from a botnet-like pattern (hundreds or thousands of different IPs in a short time).

4. Alerts from Hosting Provider or Monitoring Tools:
   - Azure and UptimeRobot (or Pingdom) can notify when downtime or performance drops are detected.
   - WAF (Web Application Firewall) tools may detect excessive or malicious request patterns.

🛡️ Response – How to respond to a DoS/DDoS Attack:

1. Immediate Containment:
   - Enable rate-limiting or throttling using Laravel middleware (e.g., Laravel Throttle).
   - Block offending IPs temporarily using Azure Network Security Group (NSG) rules or Fail2Ban.
   - If under a major attack, put the site into "maintenance mode" to preserve backend stability.
2. Scale Up Temporarily (if possible):
   - Use Azure autoscaling features to temporarily increase server resources (RAM, CPU, multiple instances).
3. Enable a CDN or DDoS Protection Layer:
   - Services like Cloudflare or Azure Front Door can absorb and filter DDoS traffic.
   - These services also offer Web Application Firewall (WAF) protection.
4. Investigate:
   - Check server and application logs for the nature of the requests.
   - Identify the main source(s) of attack and decide whether it’s targeted or random.
5. Restore Normal Operation:
   - Once traffic normalizes, slowly remove block rules and restore original server limits.
   - Confirm service availability using monitoring tools.
6. Post-Incident Actions:
   - Write a full incident report (date/time, nature of attack, response timeline, impact).
   - Plan and test an automated DDoS mitigation strategy.
   - Educate the team on red flags and quick responses to load spikes.

Unauthorized Access / Credential Stuffing

🔍 Detection – How to detect Unauthorized Access / Credential Stuffing:

1. Login Patterns:

- Multiple failed login attempts in a short time (especially from the same IP or for many accounts).
- Successful logins followed by unusual activity (e.g., accessing multiple user pages quickly).

2. IP & Geo Location Anomalies:

- Logins from suspicious IPs or countries outside expected regions.
- Same account logging in from different locations within minutes.

3. Logs & Monitoring:

- Laravel logs (or any custom log middleware) can show spikes in login activity.
- Use Azure Security Center or intrusion detection tools to identify patterns of brute-force attacks.

4. User Reports:

- Users may report being locked out, unexpected password change notifications, or seeing activity they didn't perform.

🛡️ Response – How to respond to Unauthorized Access / Credential Stuffing:

1. Immediate Containment:
   - Temporarily lock accounts with excessive failed logins (Laravel throttle middleware can help).
   - Block or throttle IPs with suspicious behavior using a firewall or services like Fail2Ban or Azure WAF.
   - Force logout for any session suspected of being compromised.
2. Investigation:
   - Identify affected user accounts and source IP addresses.
   - Check login timestamps, user-agent strings, and activity logs.
   - Compare with leaked credential patterns (have users reused passwords from past breaches?).
3. Eradication:
   - Reset passwords for affected users.
   - Invalidate all sessions if breach is significant.
   - Implement or enforce strong password policies (e.g., minimum length, complexity).
4. Recovery:
   - Enable Two-Factor Authentication (2FA) if not already enabled.
   - Add CAPTCHA or invisible bot protection (e.g., reCAPTCHA) on login page to slow down bots.
   - Monitor accounts for continued unusual behavior.
5. Post-Incident Actions:
   - Notify affected users and advise them to change passwords on other services too.
   - Conduct a full audit of authentication and session management.
   - Update your playbook and perform a penetration test focusing on auth endpoints.

Data Corruption or Data Loss

🔍 Detection – How to detect Data Corruption or Data Loss:

1. Application Errors:
   - Users report missing or incorrect data (e.g., grades, schedules not showing, corrupted text).
   - Laravel error logs show issues with database reads/writes or null/undefined values.

2. Database Integrity Checks:
   - Unexpected NULL values, broken foreign keys, or rows missing required fields.
   - Scheduled checksum comparisons between production and backup datasets.

3. Unusual Activity Logs:
   - Unexpected deletions or edits in audit logs.
   - Bulk update/delete commands recorded without proper user activity.

4. Backup Monitoring:
   - Alerts from backup tools (e.g., Azure Backup) indicating failed or incomplete backups.

🛡️ Response – How to respond to Data Corruption or Loss:

1. Containment:
   - Switch the app to read-only mode if possible to prevent further data damage
   - Identify and isolate the affected tables or modules (e.g., registration, grading).
2. Investigation:
   - Check who made the last changes (using database logs or Laravel activity log packages).
   - Identify if the issue is from a system bug, malicious activity, or external failure (e.g., hardware/disk error).
3. Recovery:
   - Restore data from the most recent clean backup (full or incremental).
   - Validate restored data using integrity checks and user testing.
   - If using SQLite/MySQL, make sure to flush cache and restart services to prevent rollback issues.
4. Communication:
   - Inform stakeholders (students, faculty, admin) of affected services.
   - Notify users whose data may have been lost or rolled back.
5. Post-Incident Actions:
   - Implement better backup policies (daily full backups + real-time incremental if possible).
   - Perform regular backup tests (restore drills).
   - Introduce database-level audit logging to track all critical data changes.
   - Consider enabling versioning for sensitive data (e.g., using soft deletes or revision tables).

Server Downtime / High Load

🔍 Detection – How to detect Server Downtime / High Load:

1. Monitoring Tools:
   - Use tools like UptimeRobot, Azure Monitor, or Pingdom to get real-time alerts for downtime or high response times.
   - Laravel Telescope or custom logging can track long-running queries or slow responses.

2. Server Metrics:
   - Sudden spikes in CPU, RAM, or disk I/O usage on Azure.
   - High number of active PHP processes or slow database queries (check Laravel debug logs and MySQL slow query logs).

3. User Reports:
   - Multiple users report the site being slow, timing out, or completely unreachable.

4. Logs:
   - Look for error 500s, database timeouts, or Laravel queue delays.
   - Check for out-of-memory (OOM) errors or PHP max execution time exceeded.

🛡️ Response – How to respond to Server Downtime / High Load:

1. Immediate Containment:
   - Enable maintenance mode using php artisan down to reduce load during investigation.
   - Restart affected services (e.g., PHP, MySQL) if they're unresponsive or hung.
   - Scale up temporarily using Azure (increase instance size or number of instances).
2. Investigation:
   - Identify the root cause: Is it a traffic spike, infinite loop in code, database lock, or external API failure?
   - Use Laravel logs and Azure Application Insights to trace where the bottleneck is happening.
   - Check queue workers, database indexes, and slow queries.
3. Recovery:
   - Fix the root problem (e.g., optimize query, fix infinite loop, increase memory limit).
   - Deploy hotfix if needed and monitor metrics for stability.
   - If downtime lasted long, notify users and provide updates on what happened.
4. Post-Incident Actions:
   - Set up autoscaling and alerts in Azure for CPU/memory/disk/network thresholds.
   - Optimize Laravel performance (caching config/routes/views, using queue workers, limiting N+1 queries).
   - Use a load balancer and CDN (e.g., Cloudflare or Azure Front Door) to reduce backend pressure.
   - Regularly stress-test your app and DB (e.g., with tools like Apache Bench or Laravel's built-in test framework).

Misuse of Admin Privileges

🔍 Detection – How to detect Misuse of Admin Privileges:

1. Audit Logs & Access Trails:
   - Review admin actions such as user deletions, grade modifications, or unauthorized data exports.
   - Use Laravel activity logging packages (e.g., Spatie Activity Log, Laravel Telescope) to track all admin actions.

2. Behavioral Anomalies:
   - Unusual patterns like repeated access to sensitive student records or course data outside of working hours.
   - Frequent use of admin-only features that aren't part of the user's normal workflow.

3. Role Escalation Alerts:
   - Logs showing a regular user being granted admin access without proper request/ticket flow.
   - Unexpected changes to user roles, permissions, or access scopes.

4. External Monitoring:
   - Azure Security Center or identity protection services can detect privilege abuse or insider threats.

🛡️ Response – How to respond to Misuse of Admin Privileges:

1. Immediate Containment:
   - Revoke admin privileges for the suspected account.
   - If necessary, disable the account entirely while an investigation is ongoing.
   - Lock down any critical actions the user may have initiated (e.g., suspend deletion tasks, pause workflows).
2. Investigation:
   - Review logs to identify what the admin did and when.
   - Interview involved users (if needed) and assess whether it was malicious, accidental, or due to poor training.
   - Check if any unauthorized data was exported or altered (grades, registration records, etc.).
3. Recovery:
   - Restore any changed or deleted data from backups if possible.
   - Notify affected users if their data was impacted.
   - Reaudit permissions and make sure least-privilege access is enforced.
4. Post-Incident Actions:
   - Implement role-based access control (RBAC) strictly — only give permissions users absolutely need.
   - Enable approval workflows for sensitive admin actions (e.g., grade changes, account deletions).
   - Require 2FA for all admin accounts.
   - Regularly audit admin activity logs and create alerting rules for suspicious actions.
   - Educate admin users on acceptable use and legal/data protection responsibilities.

Broken Access Controls

🔍 Detection – How to detect Broken Access Controls:

1. Role Bypass Attempts:
   - Requests from lower-privileged users (e.g., students) trying to access admin or faculty URLs (like /admin/dashboard or /faculty/grades).
   - Laravel logs showing 403/404 errors or unauthorized access attempts.

2. Insecure Direct Object References (IDOR):
   - Users manually changing URLs or IDs (e.g., /student/marks/1002 instead of /student/marks/1001) to access data not meant for them.
   - Unusual log entries where a user accesses another user's data or performs unauthorized actions.

3. Penetration Testing:
   - Automated tools (like OWASP ZAP or Burp Suite) can help detect broken access rules.
   - Manual tests by QA/security team changing session roles or manipulating request parameters.

4. User Reports:
   - Reports from users saying they can see other students' data, modify unauthorized fields, or access unintended sections of the site.

🛡️ Response – How to respond to Broken Access Controls:

1. Immediate Containment:
   - Disable or restrict the vulnerable endpoint or feature until a fix is applied.
   - Log out suspicious sessions and invalidate tokens for affected users.
2. Investigation:
   - Identify which access control checks are missing or failing.
   - Determine if any sensitive data was exposed or manipulated (e.g., grades, registration info).
   - Check access logs to find out how many users were affected.
3. Recovery:
   - Revert unauthorized changes if any were made (e.g., wrong grades or settings).
   - Restore affected data from backup if corruption or overwriting occurred.
   - Inform users if their data privacy was compromised.
4. Eradication:
   - Implement strict role-based access control (RBAC) in Laravel using middleware (e.g., Gate, Policy, or custom role middleware).
   - Make sure every route and controller checks for both authentication and proper authorization.
   - Never rely on frontend controls (like hiding buttons) for access control.
5. Post-Incident Actions:
   - Conduct a complete audit of access control across the system.
   - Include access control rules in automated testing.
   - Add logging and alerts for unauthorized access attempts.
   - Educate developers on secure coding and access patterns.

Backup & Recovery Plan

Overview:

To ensure the integrity, availability, and resilience of the college's web application and its critical data, a comprehensive Backup & Recovery Plan is in place. This plan defines how data is backed up, where it is stored, how restoration is carried out, and how the system is protected from data loss or corruption due to technical failure, human error, or malicious attacks.

Backup Scope:

- The following components are included in the backup process:
- Database: All MySQL data related to student registrations, course enrollments, schedules, grades, complaints, and user accounts.
- Application Codebase: All Laravel source code, including routes, controllers, models, and configuration files.
- Environment Configuration: Secure backup of environment (.env) settings stored separately and encrypted.
- Logs: Authentication logs, activity logs, error logs, and audit trails necessary for investigation and recovery.

Backup Frequency:

- Database Backups: Full backup every 24 hours (daily at 2:00 AM).
- Codebase Snapshots: Weekly backup and after every major deployment.
- Log Files: Backed up every 3 days with log rotation enabled.
- Configuration Files: Backed up whenever modified.

Backup Storage & Security:

- Storage Location: Backups are stored securely on Azure Blob Storage with Geo-Redundant Storage (GRS) enabled.
- Encryption: All backups are encrypted in transit (TLS 1.2+) and at rest using AES-256.
- Access Control: Access to backup storage is restricted using Azure Role-Based Access Control (RBAC). Only authorized personnel can view, modify, or restore backups.

Recovery Procedures:

In the event of data loss, system failure, or a cyberattack, the following recovery process is initiated:

a. Detection & Assessment

- The security team is alerted via logs, monitoring tools, or user reports.
- An initial assessment is conducted to determine the impact and scope.

b. Containment

- Affected systems or services are isolated.
- Maintenance mode is enabled to prevent further data changes.

c. Restoration

- The most recent stable backup is selected.
- Database is restored using Azure Database Recovery or MySQL tools.
- Application files and configurations are redeployed.
- Data and system integrity are verified post-restore.

d. Post-Recovery Actions

- Full system scan is performed for hidden threats.
- Affected users are notified.
- An incident report is generated and logged.

Testing & Validation:

- Backup restore tests are conducted every quarter in a sandbox environment.
- Random data integrity checks are performed monthly.
- Recovery Time Objective (RTO): 2 hours
- Recovery Point Objective (RPO): 24 hours

Responsibilities:

- Cybersecurity Lead: Oversees the security and encryption of backups.
- DevOps Engineer: Maintains and automates the backup and recovery process.
- IT Admin: Executes recovery steps and ensures successful restoration.

Continuous Improvement:

- The Backup & Recovery Plan is reviewed and updated every 6 months or following any major incident or infrastructure change to improve its reliability and effectiveness.