Security Analysis and Implementation Report

Executive Summary

This document provides a comprehensive security analysis of the Hotel Shift Log Management System and details the security measures implemented to protect against common vulnerabilities and attacks when deployed on Google Cloud Platform.

---

1. Threat Model

1.1 Assets to Protect

- User credentials and personal information
- Shift reports containing sensitive hotel operations data
- File attachments (potentially containing PII)
- Manager comments and private notes
- System availability and integrity

1.2 Threat Actors

- **External Attackers**: Attempting unauthorized access, data theft, or service disruption
- **Malicious Insiders**: Employees attempting privilege escalation or data exfiltration
- **Automated Bots**: Scanning for vulnerabilities, attempting credential stuffing

1.3 Attack Vectors

- Authentication bypass
- SQL injection
- Cross-site scripting (XSS)
- Path traversal / file inclusion
- Resource exhaustion (DoS)
- Malicious file uploads
- Session hijacking
- CSRF attacks

---

2. Security Measures Implemented

2.1 Authentication and Authorization

✅ Password Security

Implementation:
- BCrypt hashing with cost factor 12 (2^12 = 4096 iterations)
- Passwords never stored in plain text
- Secure password validation on login

Code Location: `/lib/auth.ts`, `/app/api/users/route.ts`

Risk Mitigated: Password compromise, rainbow table attacks

✅ Session Management

Implementation:
- NextAuth.js with JWT strategy
- Secure session tokens
- HTTP-only cookies (managed by NextAuth)
- Session expiration

Code Location: `/lib/auth.ts`, `/middleware.ts`

Risk Mitigated: Session hijacking, token theft

✅ Role-Based Access Control (RBAC)

Implementation:
- Three distinct roles: SUPER_ADMIN, MANAGER, EMPLOYEE
- Granular permissions enforced at API level
- Middleware protection for sensitive routes
- Database-level user archiving (prevents archived users from logging in)

Code Location: All `/app/api/*` routes, `/middleware.ts`

Risk Mitigated: Unauthorized access, privilege escalation

2.2 Input Validation and Sanitization

✅ Filename Sanitization

Implementation:

```
// Remove path components and dangerous characters
sanitizeFilename(filename: string) {
  const basename = path.basename(filename)
  const sanitized = basename.replace(/[^a-zA-Z0-9._-]/g, '_')
  // Enforce length limits
  return sanitized.substring(0, 255)
}
```

Code Location: `/lib/security.ts`

Risk Mitigated: Path traversal, file injection, command injection

✅ Path Traversal Protection

Implementation:

```
validateFilePath(filepath: string, allowedDirectory: string) {
  const resolvedPath = path.resolve(filepath)
  const resolvedAllowedDir = path.resolve(allowedDirectory)
  return resolvedPath.startsWith(resolvedAllowedDir)
}
```

Code Location: `/lib/security.ts`, `/app/api/files/[...filename]/route.ts`

Risk Mitigated: Directory traversal attacks, unauthorized file access

✅ SQL Injection Protection

Implementation:
- Prisma ORM with parameterized queries
- No raw SQL concatenation
- Type-safe database operations

Code Location: All database operations via Prisma

Risk Mitigated: SQL injection, database compromise

✅ File Type Validation

Implementation:

```
const ALLOWED_EXTENSIONS = ['.jpg', '.jpeg', '.png', '.pdf', '.doc', '.docx',
                            '.xls', '.xlsx', '.txt', '.csv', '.zip']
// Validate by file extension
// Additional MIME type checking
```

Code Location: `/app/api/reports/create/route.ts` , `/app/api/comments/route.ts`

Risk Mitigated: Malicious file uploads, code execution

2.3 Resource Exhaustion Protection

✅ Daily Post Limits

Implementation:
- Maximum 25 reports per user per day
- Tracked via `DailyPostTracker` database table
- Reset at midnight (server timezone)

Code Location: `/app/api/reports/create/route.ts`

Risk Mitigated: Spam, DoS via excessive posting

✅ File Size Limits

Implementation:
- Individual file: 30MB maximum
- Total per report: 90MB maximum (3 files × 30MB)
- Enforced before file processing

Code Location: `/app/api/reports/create/route.ts` , `/app/api/comments/route.ts`

Risk Mitigated: Storage exhaustion, bandwidth abuse, zip bombs

✅ Comment Limits

Implementation:
- Maximum 30 comments per manager per report
- Prevents comment spam
- Database counter validation

Code Location: `/app/api/comments/route.ts`

Risk Mitigated: Database bloat, spam

✅ Rate Limiting

Implementation:

```
// Login: 5 attempts per 15 minutes per IP
loginRateLimiter.check(request, 5, clientIp)

// API: 100 requests per minute per user
apiRateLimiter.check(request, 100, userId)
```

Code Location: `/lib/rate-limit.ts` (ready for integration in auth routes)

Risk Mitigated: Brute force attacks, credential stuffing, API abuse

Note: Rate limiting implementation is prepared but not yet integrated into all routes. To activate, import and call rate limiters in:
- `/app/api/auth/[...nextauth]/route.ts` - for login protection
- Other high-traffic API endpoints

2.4 Security Headers

✅ Implemented Headers

Implementation:

```
// Middleware adds headers to all authenticated routes
'X-Content-Type-Options': 'nosniff'          // Prevent MIME sniffing
'X-Frame-Options': 'DENY'                    // Prevent clickjacking
'X-XSS-Protection': '1; mode=block'          // XSS filter
'Referrer-Policy': 'strict-origin-when-cross-origin'  // Privacy
'Permissions-Policy': 'camera=(), microphone=(), geolocation=()'  // Restrict APIs
```

Code Location: `/middleware.ts`

Risk Mitigated: XSS, clickjacking, MIME confusion, privacy leaks

⚠️ Additional Headers (Recommended for GCP Load Balancer)

Configure in Cloud Armor or Load Balancer:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-inline'; style-
src 'self' 'unsafe-inline'
```

2.5 File Handling Security

✅ Secure File Serving

Implementation:
- Authentication required for all file access
- Path validation before serving
- Sanitized Content-Disposition headers
- MIME type validation
- `X-Content-Type-Options: nosniff` header

Code Location: `/app/api/files/[...filename]/route.ts`

Risk Mitigated: Unauthorized file access, XSS via file upload

✅ File Storage Isolation

Implementation:
- Uploads stored outside public directory
- Served via authenticated API route
- No direct web server access to uploads

Code Location: `/uploads/` directory, `/app/api/files/` routes

Risk Mitigated: Unauthorized file access, directory listing

2.6 Environment Security

✅ Environment Variable Validation

Implementation:

```
validateEnvironment() {
  const required = ['DATABASE_URL', 'NEXTAUTH_SECRET']
  const missing = required.filter(key => !process.env[key])
  if (missing.length > 0) throw new Error(...)
  // Validate secret strength
}
```

Code Location: `/lib/security.ts`

Risk Mitigated: Misconfiguration, weak secrets

✅ Secrets Management

Implementation:
- `.env` file excluded from version control
- Use GCP Secret Manager for production
- No secrets in code or logs

Risk Mitigated: Credential exposure, unauthorized access

3. Security Risks Assessed and Status

3.1 High-Risk Threats

Threat	Status	Mitigation
SQL Injection	✅ Protected	Prisma ORM with parameterized queries
Authentication Bypass	✅ Protected	NextAuth.js, bcrypt, session management
Unauthorized File Access	✅ Protected	Path validation, authentication required
Malicious File Upload	✅ Protected	File type/size validation, sanitization
Password Compromise	✅ Protected	BCrypt with cost 12, secure storage
Session Hijacking	✅ Protected	Secure cookies, JWT, session expiration
Privilege Escalation	✅ Protected	RBAC enforcement at API level

3.2 Medium-Risk Threats

Threat	Status	Mitigation
XSS Attacks	✅ Protected	Security headers, input sanitization, React escaping
CSRF	✅ Protected	NextAuth CSRF tokens, SameSite cookies
Path Traversal	✅ Protected	Path validation, sanitized filenames
Resource Exhaustion	✅ Protected	Post limits, file size limits, comment limits
Brute Force	⚠️ Partial	Rate limiting prepared (not fully integrated)
Clickjacking	✅ Protected	X-Frame-Options header

3.3 Low-Risk Threats

Threat	Status	Mitigation
Information Disclosure	✅ Protected	Generic error messages, no stack traces in prod
MIME Confusion	✅ Protected	X-Content-Type-Options header
Referrer Leakage	✅ Protected	Referrer-Policy header
Directory Listing	✅ Protected	Uploads outside webroot, API-served files

---

4. Risks NOT Addressed (Out of Scope for Current Implementation)

4.1 Advanced Threats Requiring Infrastructure Changes

❌ **DDoS Protection** - Requires GCP Cloud Armor with rate limiting rules
❌ **Advanced Bot Detection** - Requires reCAPTCHA or similar service
❌ **Web Application Firewall (WAF)** - Requires GCP Cloud Armor
❌ **Certificate Pinning** - Mobile app specific, not applicable
❌ **Database Encryption at Rest** - Enabled by default in Cloud SQL

4.2 Threats Mitigated by GCP Infrastructure

✅ **Network-Level DDoS** - GCP’s global load balancing
✅ **Database Encryption** - Cloud SQL encryption at rest/in transit
✅ **Infrastructure Security** - GCP’s security measures
✅ **Backup Security** - Cloud Storage encryption

---

5. Security Testing Performed

5.1 Manual Testing

- [x] SQL injection attempts (Prisma parameterization validated)
- [x] Path traversal attempts (blocked by validation)
- [x] File upload with dangerous extensions (rejected)
- [x] Oversized file uploads (rejected at size limit)
- [x] Authentication bypass attempts (failed)
- [x] CSRF token validation (NextAuth protection verified)

5.2 Code Review

- [x] All API routes require authentication
- [x] RBAC enforced at API level

- [x] Input validation on all user inputs
- [x] No raw SQL queries
- [x] Secure file handling
- [x] No sensitive data in logs

5.3 Dependency Security

```
# Run audit before deployment
yarn audit
```

Recommendation: Set up automated dependency scanning in CI/CD pipeline.

---

6. Deployment Security Checklist

Pre-Deployment

- [ ] Change all default passwords
- [ ] Generate strong NEXTAUTH_SECRET (32+ characters)
- [ ] Configure SMTP credentials in Secret Manager
- [ ] Enable Cloud SQL backups
- [ ] Set up Cloud Storage for file backups
- [ ] Configure SSL/TLS certificates
- [ ] Review and update CORS settings if needed

Post-Deployment

- [ ] Test all authentication flows
- [ ] Verify email notifications work
- [ ] Test backup and restore procedures
- [ ] Configure monitoring and alerts
- [ ] Review audit logs
- [ ] Perform security scan (e.g., OWASP ZAP)
- [ ] Document incident response procedures

Ongoing Maintenance

- [ ] Weekly log reviews
- [ ] Monthly dependency updates
- [ ] Quarterly security audits
- [ ] Regular backup testing
- [ ] User access reviews

---

7. Incident Response Plan

7.1 Security Incident Severity Levels

Critical: Unauthorized access to database, mass data breach
High: Successful privilege escalation, single-user account compromise

Medium: Failed attack attempts in logs, suspicious activity
Low: Configuration issues, minor vulnerabilities

7.2 Response Procedures

Immediate Actions (Critical/High):
1. Isolate affected systems
2. Revoke compromised credentials
3. Enable Cloud SQL read-only mode if needed
4. Notify stakeholders
5. Preserve logs and evidence

Investigation:
1. Review Cloud Logging for attack vectors
2. Check database for unauthorized changes
3. Review user access logs
4. Identify scope of compromise

Remediation:
1. Patch vulnerabilities
2. Reset affected credentials
3. Restore from clean backups if needed
4. Implement additional controls
5. Document lessons learned

---

8. Recommendations for Future Enhancements

8.1 Short Term (Next Release)

1. **Fully integrate rate limiting** into auth routes
2. **Add reCAPTCHA** to login form
3. **Implement audit logging** for all sensitive operations
4. **Add IP allowlisting** option for admin access

8.2 Medium Term (3-6 Months)

1. **Two-factor authentication (2FA)** for admin accounts
2. **Password complexity requirements**
3. **Session timeout** with activity tracking
4. **Automated security scanning** in CI/CD

8.3 Long Term (6-12 Months)

1. **SOC 2 compliance** preparation
2. **Penetration testing** by third-party
3. **Security awareness training** for users
4. **Advanced threat detection** with ML

---

9. Compliance Considerations

9.1 Data Protection

- **GDPR/CCPA**: User data deletion implemented (archive + delete flow)
- **Data Residency**: Cloud SQL region can be configured to meet requirements
- **Right to Export**: Export functionality available (PDF, CSV)

9.2 Access Controls

- **Least Privilege**: Role-based access enforced
- **Audit Trail**: Database tracks authorship and timestamps
- **User Management**: Admins can manage user accounts

9.3 Data Retention

- **Backups**: Configurable retention (default 30 days)
- **Deleted Data**: Author names preserved after user deletion
- **Logs**: Cloud Logging retention configurable

---

10. Security Contact Information

For security concerns or to report vulnerabilities:

Internal Contact: System Administrator
Process:
1. Document the issue with steps to reproduce
2. Do NOT disclose publicly
3. Provide details: affected version, impact, screenshots
4. Allow 48 hours for initial response

Conclusion

This application implements defense-in-depth security with multiple layers of protection. The combination of secure coding practices, infrastructure security (GCP), and operational procedures provides strong protection against common web application vulnerabilities.

Security Posture: STRONG

The application is ready for production deployment with the understanding that security is an ongoing process requiring regular updates, monitoring, and improvements.

Document Version: 1.0
Last Updated: October 21, 2025
Next Review: January 21, 2026