Contents

RADIANT Security Audit Checklist

Version: 5.42.0

Last Audit: January 2026

This document provides a security audit checklist for the RADIANT platform covering authentication, authorization, data isolation, and security best practices.

---

1. Row-Level Security (RLS) Policies

1.1 Core Tables - RLS Status

Table	RLS Enabled	Policy Type	Verified
tenants		tenant_id match	
users		tenant_id + user_id	
ai_reports		tenant_id match	
ai_report_templates		tenant_id OR is_system	
ai_report_brand_kits		tenant_id match	
ai_report_insights		tenant_id match	
billing_credits		tenant_id match	
usage_records		tenant_id match	
model_configs		tenant_id match	
agi_brain_plans		tenant_id + user_id	
conversations		tenant_id + user_id	

1.2 RLS Policy Template

```
-- Standard tenant isolation policy
ALTER TABLE table_name ENABLE ROW LEVEL SECURITY;

CREATE POLICY tenant_isolation ON table_name
  FOR ALL
  USING (tenant_id = current_setting('app.current_tenant_id', true)::text)
  WITH CHECK (tenant_id = current_setting('app.current_tenant_id', true)::text);

-- User-level isolation (for user-specific data)
CREATE POLICY user_isolation ON table_name
  FOR ALL
  USING (
    tenant_id = current_setting('app.current_tenant_id', true)::text
    AND user_id = current_setting('app.current_user_id', true)::text
  );
```

1.3 RLS Audit Queries

```
-- Check which tables have RLS enabled
SELECT schemaname, tablename, rowsecurity
```

```sql
FROM pg_tables
WHERE schemaname = 'public' AND rowsecurity = true;

-- Check RLS policies
SELECT tablename, policyname, permissive, roles, cmd, qual
FROM pg_policies
WHERE schemaname = 'public';

-- Verify tenant context is set
SELECT current_setting('app.current_tenant_id', true);
```

---

2. Authentication Flow

2.1 Token Validation

```typescript
// Required checks for every authenticated request:
interface AuthChecks {
  tokenSignature: boolean;      // JWT signature valid
  tokenExpiry: boolean;         // Not expired
  tokenIssuer: boolean;         // Correct issuer
  tenantExists: boolean;        // Tenant is valid
  tenantActive: boolean;        // Tenant not suspended
  userExists: boolean;          // User exists
  userActive: boolean;          // User not disabled
  permissionsValid: boolean;    // Has required permissions
}
```

2.2 Auth Middleware Checklist

- ☒ JWT signature verification
- ☒ Token expiration check
- ☒ Issuer validation
- ☒ Audience validation
- ☒ Tenant context injection
- ☒ User context injection
- ☒ Role/permission extraction
- ☒ Rate limiting per user
- ☒ Session validation (if applicable)

2.3 Admin Authentication

```typescript
// Admin endpoints require additional checks:
const adminAuthChecks = {
  isAdmin: user.roles.includes('admin') || user.isSuperAdmin,
  hasPermission: (permission: string) => user.permissions.includes(permission),
  isSuperAdmin: user.isSuperAdmin === true,
};
```

---

3. Authorization Patterns

3.1 Permission Hierarchy

```
SuperAdmin
     Full platform access
     Can access any tenant
     Can modify system settings

TenantAdmin
     Full tenant access
     Can manage users
     Can configure models
     Can view billing

TenantUser
     Limited to own data
     Can use AI features
     Cannot modify settings
```

3.2 Resource Authorization

```typescript
// Every resource access must verify:
async function authorizeResourceAccess(
  userId: string,
  tenantId: string,
  resourceType: string,
  resourceId: string,
  action: 'read' | 'write' | 'delete'
): Promise<boolean> {
  // 1. Verify tenant access
  if (!await canAccessTenant(userId, tenantId)) return false;

  // 2. Verify resource belongs to tenant
  if (!await resourceBelongsToTenant(resourceType, resourceId, tenantId)) return false;

  // 3. Verify user has permission for action
  if (!await userHasPermission(userId, resourceType, action)) return false;

  return true;
}
```

---

4. Input Validation & Sanitization

4.1 Required Validations

Input Type	Validation	Sanitization
UUID	Format check	None needed
Email	RFC 5322 regex	Lowercase, trim
Tenant ID	Alphanumeric + hyphen	Lowercase
User input text	Length limits	HTML escape
JSON payloads	Schema validation	Type coercion
File uploads	MIME type, size	Virus scan
SQL parameters	Parameterized queries	Never concatenate

4.2 SQL Injection Prevention

```
//  CORRECT: Parameterized queries
await executeStatement(
  'SELECT * FROM users WHERE tenant_id = $1 AND id = $2',
  [stringParam('tenantId', tenantId), stringParam('id', userId)]
);

//  WRONG: String concatenation
await executeStatement(
  `SELECT * FROM users WHERE tenant_id = '${tenantId}'` // NEVER DO THIS
);
```

4.3 XSS Prevention

```
// Sanitize user-generated content before rendering
import DOMPurify from 'dompurify';

const sanitizedHtml = DOMPurify.sanitize(userContent, {
  ALLOWED_TAGS: ['b', 'i', 'em', 'strong', 'a', 'p', 'br'],
  ALLOWED_ATTR: ['href', 'title'],
});
```

---

5. API Security

5.1 Rate Limiting

Endpoint Type	Rate Limit	Window
Auth endpoints	10 req	1 min
Admin API	100 req	1 min
AI generation	20 req	1 min
Export endpoints	5 req	1 min
Public API	1000 req	1 min

5.2 CORS Configuration

```
// Strict CORS for production
const corsConfig = {
  origin: [
    'https://admin.radiant.ai',
    'https://app.radiant.ai',
    process.env.NODE_ENV === 'development' && 'http://localhost:3000',
  ].filter(Boolean),
  methods: ['GET', 'POST', 'PUT', 'DELETE', 'PATCH'],
  allowedHeaders: ['Content-Type', 'Authorization', 'X-Tenant-ID'],
  credentials: true,
  maxAge: 86400,
};
```

5.3 Security Headers

```
const securityHeaders = {
  'Strict-Transport-Security': 'max-age=31536000; includeSubDomains',
  'X-Content-Type-Options': 'nosniff',
  'X-Frame-Options': 'DENY',
  'X-XSS-Protection': '1; mode=block',
  'Content-Security-Policy': "default-src 'self'; script-src 'self'",
  'Referrer-Policy': 'strict-origin-when-cross-origin',
};
```

---

6. Data Protection

6.1 Encryption at Rest

Data Type	Encryption	Key Management
Aurora PostgreSQL	AES-256	AWS KMS
S3 Buckets	AES-256	AWS KMS
Secrets	AES-256	Secrets Manager
Local storage	N/A	Not stored

6.2 Encryption in Transit

- All API traffic: TLS 1.2+
- Database connections: SSL required
- Internal AWS: VPC endpoints

6.3 PII Handling

```
// PII fields requiring special handling
const piiFields = [
  'email',
```

```
  'name',
  'phone',
  'address',
  'ip_address',
  'api_key',
];

// Log redaction
function redactPII(obj: Record<string, unknown>): Record<string, unknown> {
  const redacted = { ...obj };
  for (const field of piiFields) {
    if (field in redacted) {
      redacted[field] = '[REDACTED]';
    }
  }
  return redacted;
}
```

---

7. Secret Management

7.1 Secret Storage

Secret Type	Storage	Rotation
API Keys	Secrets Manager	90 days
Database credentials	Secrets Manager	30 days
JWT signing keys	Secrets Manager	365 days
Encryption keys	KMS	Auto

7.2 Secret Access

```
// Secrets should never be:
// - Logged
// - Returned in API responses
// - Stored in environment variables (use Secrets Manager)
// - Committed to code

// Correct pattern:
const secret = await secretsManager.getSecretValue({
  SecretId: 'prod/radiant/api-key',
});
```

---

8. Audit Logging

8.1 Events to Log

Event Category	Events
Authentication	Login, logout, failed attempts
Authorization	Permission denied, role changes
Data access	Read sensitive data, exports
Data modification	Create, update, delete
Admin actions	User management, settings changes
Security events	Rate limit hits, suspicious activity

8.2 Log Format

```typescript
interface AuditLog {
  timestamp: string;
  eventType: string;
  tenantId: string;
  userId: string;
  resourceType: string;
  resourceId: string;
  action: string;
  outcome: 'success' | 'failure';
  ipAddress: string;
  userAgent: string;
  details: Record<string, unknown>;
}
```

---

9. Vulnerability Checklist

9.1 OWASP Top 10 Coverage

Vulnerability	Mitigation	Status
Injection	Parameterized queries	
Broken Auth	JWT + MFA support	
Sensitive Data Exposure	Encryption + redaction	
XML External Entities	Not applicable (JSON only)	
Broken Access Control	RLS + auth middleware	
Security Misconfiguration	IaC + security headers	
XSS	Content sanitization	
Insecure Deserialization	Schema validation	
Known Vulnerabilities	Dependabot + npm audit	
Insufficient Logging	Audit logging	

9.2 Dependency Security

```bash
# Run regularly
npm audit
npm audit fix
```

```
# Check for outdated packages
npm outdated

# Review Dependabot alerts in GitHub
```

---

10. Incident Response

10.1 Security Incident Procedure

1. **Detect** - Automated alerts or manual report
2. **Contain** - Isolate affected systems
3. **Investigate** - Review logs, identify scope
4. **Eradicate** - Remove threat, patch vulnerability
5. **Recover** - Restore services
6. **Document** - Post-incident report

10.2 Emergency Contacts

```
Security Team: security@radiant.ai
On-Call: PagerDuty integration
AWS Support: Enterprise support case
```

---

11. Compliance

11.1 Compliance Requirements

Standard	Status	Notes
SOC 2 Type II	In progress	Annual audit
GDPR		Data processing agreements
HIPAA	Optional	PHI sanitization available
CCPA		Privacy controls implemented

11.2 Data Retention

Data Type	Retention	Deletion
Audit logs	2 years	Auto-archive
User data	Account lifetime + 30 days	On request
AI conversations	90 days default	Configurable
Reports	1 year	Auto-delete

---

12. Security Testing

12.1 Testing Schedule

Test Type	Frequency	Last Run
Dependency audit	Weekly	Auto
SAST (static analysis)	On PR	Auto
DAST (dynamic)	Monthly	Manual
Penetration test	Annual	External
Security review	Quarterly	Internal

12.2 Security Test Commands

```
# Static analysis
npm run lint
npm run typecheck

# Dependency audit
npm audit

# Secret scanning
git secrets --scan

# Infrastructure security
cdk synth && cfn-nag
```

Approval

Role	Name	Date
Security Lead		
Engineering Lead		
CTO		