Advanced Security Enhancement Plan - Part 1: Optimization & Database Integration

1. Memory Usage Optimization with Database Persistence

Phase 1.1: Database Schema Design (Week 1)

// server/security/advanced/threat/schema.ts

import { pgTable, serial, text, timestamp, integer, boolean, jsonb } from 'drizzle-orm/pg-core';

// Threat table to store detected security threats

export const securityThreats = pgTable('security\_threats', {

id: serial('id').primaryKey(),

threatId: text('threat\_id').notNull().unique(),

timestamp: timestamp('timestamp').notNull().defaultNow(),

threatType: text('threat\_type').notNull(),

severity: text('severity').notNull(),

description: text('description').notNull(),

sourceIp: text('source\_ip').notNull(),

userId: text('user\_id'),

requestPath: text('request\_path'),

requestMethod: text('request\_method'),

evidence: jsonb('evidence'),

ruleId: text('rule\_id').notNull(),

actionTaken: jsonb('action\_taken'),

resolved: boolean('resolved').default(false),

resolvedBy: text('resolved\_by'),

resolvedAt: timestamp('resolved\_at'),

isArchived: boolean('is\_archived').default(false)

});

// Rules table to store threat detection rules

export const detectionRules = pgTable('detection\_rules', {

id: serial('id').primaryKey(),

ruleId: text('rule\_id').notNull().unique(),

name: text('name').notNull(),

description: text('description').notNull(),

threatType: text('threat\_type').notNull(),

severity: text('severity').notNull(),

pattern: text('pattern'),

threshold: integer('threshold'),

timeWindow: integer('time\_window'),

autoBlock: boolean('auto\_block').default(false),

autoNotify: boolean('auto\_notify').default(false),

enabled: boolean('enabled').default(true),

createdAt: timestamp('created\_at').notNull().defaultNow(),

updatedAt: timestamp('updated\_at').notNull().defaultNow()

});

// Blocked IPs table

export const blockedIps = pgTable('blocked\_ips', {

id: serial('id').primaryKey(),

ip: text('ip').notNull().unique(),

blockedAt: timestamp('blocked\_at').notNull().defaultNow(),

blockedBy: text('blocked\_by'),

reason: text('reason'),

expiresAt: timestamp('expires\_at'),

isActive: boolean('is\_active').default(true)

});

Phase 1.2: Database Service Implementation (Week 2)

// server/security/advanced/threat/ThreatDatabaseService.ts

import { db } from '../../../db';

import {

securityThreats, detectionRules, blockedIps

} from './schema';

import {

DetectedThreat, DetectionRule, ThreatType, ThreatSeverity

} from './ThreatDetectionService';

import { eq, and, desc, gte, lte } from 'drizzle-orm';

export class ThreatDatabaseService {

/\*\*

\* Save a detected threat to the database

\*/

async saveThreat(threat: DetectedThreat): Promise<void> {

await db.insert(securityThreats).values({

threatId: threat.id,

timestamp: new Date(threat.timestamp),

threatType: threat.threatType,

severity: threat.severity,

description: threat.description,

sourceIp: threat.sourceIp,

userId: threat.userId?.toString(),

requestPath: threat.requestPath,

requestMethod: threat.requestMethod,

evidence: threat.evidence,

ruleId: threat.ruleId,

actionTaken: threat.actionTaken,

resolved: threat.resolved || false,

resolvedBy: threat.resolvedBy,

resolvedAt: threat.resolvedAt ? new Date(threat.resolvedAt) : null

});

}

/\*\*

\* Get threats from the database with pagination

\*/

async getThreats(

limit?: number,

offset?: number,

includeArchived: boolean = false

): Promise<DetectedThreat[]> {

const query = db

.select()

.from(securityThreats)

.orderBy(desc(securityThreats.timestamp));

if (!includeArchived) {

query.where(eq(securityThreats.isArchived, false));

}

if (offset !== undefined) {

query.offset(offset);

}

if (limit !== undefined) {

query.limit(limit);

}

const threats = await query;

return threats.map(threat => ({

id: threat.threatId,

timestamp: new Date(threat.timestamp).getTime(),

threatType: threat.threatType as ThreatType,

severity: threat.severity as ThreatSeverity,

description: threat.description,

sourceIp: threat.sourceIp,

userId: threat.userId,

requestPath: threat.requestPath || undefined,

requestMethod: threat.requestMethod || undefined,

evidence: threat.evidence,

ruleId: threat.ruleId,

actionTaken: threat.actionTaken as string[] || undefined,

resolved: threat.resolved,

resolvedBy: threat.resolvedBy || undefined,

resolvedAt: threat.resolvedAt ? new Date(threat.resolvedAt).getTime() : undefined

}));

}

/\*\*

\* Save a detection rule to the database

\*/

async saveRule(rule: DetectionRule): Promise<void> {

// Convert RegExp to string for storage

const patternStr = rule.pattern ? rule.pattern.toString() : null;

await db.insert(detectionRules).values({

ruleId: rule.id,

name: rule.name,

description: rule.description,

threatType: rule.threatType,

severity: rule.severity,

pattern: patternStr,

threshold: rule.threshold || null,

timeWindow: rule.timeWindow || null,

autoBlock: rule.autoBlock || false,

autoNotify: rule.autoNotify || false,

enabled: rule.enabled

});

}

// Additional methods would be implemented with similar patterns

}

export const threatDbService = new ThreatDatabaseService();

Phase 1.3: Integrate Database with Threat Detection Service (Week 3)

// server/security/advanced/threat/ThreatDetectionService.ts (updated)

import { threatDbService } from './ThreatDatabaseService';

import { nanoid } from 'nanoid';

// Modified ThreatDetectionService to use database persistence

export class ThreatDetectionService {

private enabled: boolean = true;

private mlDetectionEnabled: boolean = false;

private memoryCache = {

activeThreats: new Map<string, DetectedThreat>(),

recentThreats: new Map<string, DetectedThreat>(),

blockedIps: new Set<string>()

};

// Memory-only rules for performance

private activeRules: DetectionRule[] = [];

// Cache TTL in milliseconds

private CACHE\_TTL = 5 \* 60 \* 1000; // 5 minutes

constructor() {

// Initialize with rules from the database

this.loadRulesFromDatabase();

// Initialize cache with recent threats

this.loadRecentThreatsToCache();

// Initialize blocked IPs

this.loadBlockedIpsToCache();

// Set up periodic cache refresh

setInterval(() => this.refreshCache(), this.CACHE\_TTL / 2);

console.log('Threat Detection Service initialized with database persistence');

}

/\*\*

\* Load rules from database to memory

\*/

private async loadRulesFromDatabase(): Promise<void> {

try {

const dbRules = await threatDbService.getRules();

this.activeRules = dbRules.map(rule => {

// Convert stored pattern string back to RegExp if present

let pattern: RegExp | undefined = undefined;

if (rule.pattern) {

// Extract pattern between /pattern/flags format

const patternMatch = /\/(.\*)\/([gimuy]\*)/.exec(rule.pattern);

if (patternMatch) {

try {

pattern = new RegExp(patternMatch[1], patternMatch[2]);

} catch (e) {

console.error(`Failed to parse RegExp pattern: ${rule.pattern}`, e);

}

}

}

return {

...rule,

pattern

};

});

console.log(`Loaded ${this.activeRules.length} detection rules from database`);

} catch (error) {

console.error('Failed to load rules from database:', error);

// Fall back to default rules

this.registerDefaultRules();

}

}

// Implement additional methods for database integration

/\*\*

\* Process a request for threats

\* - First checks in-memory cache for performance

\* - Falls back to database for historical data

\*/

public async processRequest(req: any): Promise<DetectedThreat | null> {

if (!this.enabled) {

return null;

}

// Implementation logic

return null;

}

}

2. Caching Strategy Enhancement

Phase 1.4: Implement LRU Cache for Rules and Configurations (Week 4)

// server/security/advanced/cache/SecurityCache.ts

export class LRUCache<K, V> {

private cache = new Map<K, V>();

private maxSize: number;

private ttl: number;

private timestamps = new Map<K, number>();

constructor(maxSize: number = 1000, ttlMs: number = 5 \* 60 \* 1000) {

this.maxSize = maxSize;

this.ttl = ttlMs;

}

set(key: K, value: V): void {

if (this.cache.size >= this.maxSize && !this.cache.has(key)) {

// Evict least recently used entry

this.evictOldest();

}

this.cache.set(key, value);

this.timestamps.set(key, Date.now());

}

get(key: K): V | undefined {

const value = this.cache.get(key);

if (value !== undefined) {

// Check if entry has expired

const timestamp = this.timestamps.get(key) || 0;

if (Date.now() - timestamp > this.ttl) {

this.cache.delete(key);

this.timestamps.delete(key);

return undefined;

}

// Update timestamp on access to mark as recently used

this.timestamps.set(key, Date.now());

}

return value;

}

private evictOldest(): void {

if (this.cache.size === 0) return;

let oldestKey: K | null = null;

let oldestTime = Infinity;

for (const [key, timestamp] of this.timestamps.entries()) {

if (timestamp < oldestTime) {

oldestTime = timestamp;

oldestKey = key;

}

}

if (oldestKey !== null) {

this.cache.delete(oldestKey);

this.timestamps.delete(oldestKey);

}

}

clear(): void {

this.cache.clear();

this.timestamps.clear();

}

size(): number {

return this.cache.size;

}

}

// SecurityCache implementation for rules and configurations

export class SecurityCache {

private ruleCache = new LRUCache<string, any>(200, 10 \* 60 \* 1000); // 10 minutes

private configCache = new LRUCache<string, any>(50, 5 \* 60 \* 1000); // 5 minutes

private ipReputationCache = new LRUCache<string, any>(1000, 30 \* 60 \* 1000); // 30 minutes

// Cache methods for rules, configs, and IP reputations

}

3. Rate Limiting Enhancement

Phase 1.5: Implement Token Bucket Algorithm (Week 5)

// server/security/advanced/ratelimit/TokenBucketRateLimiter.ts

export interface RateLimitConfig {

tokensPerInterval: number; // Number of tokens added per interval

interval: number; // Interval in milliseconds

burstCapacity: number; // Maximum tokens that can be accumulated (burst capacity)

}

export interface RateLimitState {

tokens: number; // Current number of tokens in the bucket

lastRefill: number; // Timestamp of last token refill

}

export class TokenBucketRateLimiter {

private buckets = new Map<string, RateLimitState>();

private defaultConfig: RateLimitConfig;

private customConfigs = new Map<string, RateLimitConfig>();

constructor(defaultConfig: RateLimitConfig = {

tokensPerInterval: 60,

interval: 60000, // 1 minute

burstCapacity: 120

}) {

this.defaultConfig = defaultConfig;

}

/\*\*

\* Set a custom rate limit configuration for a specific key

\*/

setCustomConfig(key: string, config: RateLimitConfig): void {

this.customConfigs.set(key, config);

}

/\*\*

\* Try to consume a token for the given key

\* Returns true if successful, false if rate limit exceeded

\*/

consume(key: string, tokens: number = 1): boolean {

const config = this.customConfigs.get(key) || this.defaultConfig;

let state = this.buckets.get(key);

const now = Date.now();

if (!state) {

// Create new bucket with full tokens minus consumed

state = {

tokens: config.burstCapacity - tokens,

lastRefill: now

};

this.buckets.set(key, state);

return true;

}

// Refill tokens based on time elapsed

const elapsedMs = now - state.lastRefill;

const tokensToAdd = (elapsedMs / config.interval) \* config.tokensPerInterval;

if (tokensToAdd > 0) {

state.tokens = Math.min(config.burstCapacity, state.tokens + tokensToAdd);

state.lastRefill = now;

}

// Check if enough tokens are available

if (state.tokens >= tokens) {

state.tokens -= tokens;

return true;

}

return false;

}

/\*\*

\* Get remaining tokens for a key

\*/

getRemainingTokens(key: string): number {

const state = this.buckets.get(key);

if (!state) {

return this.customConfigs.get(key)?.burstCapacity || this.defaultConfig.burstCapacity;

}

// Refill tokens based on time elapsed

const config = this.customConfigs.get(key) || this.defaultConfig;

const now = Date.now();

const elapsedMs = now - state.lastRefill;

const tokensToAdd = (elapsedMs / config.interval) \* config.tokensPerInterval;

return Math.min(config.burstCapacity, state.tokens + tokensToAdd);

}

/\*\*

\* Reset rate limit state for a key

\*/

reset(key: string): void {

this.buckets.delete(key);

}

}

Phase 1.6: Integrate Token Bucket with Rate Limiting Middleware (Week 6)

// server/middleware/RateLimitMiddleware.ts

import { Request, Response, NextFunction } from 'express';

import { TokenBucketRateLimiter } from '../security/advanced/ratelimit/TokenBucketRateLimiter';

export interface RateLimitOptions {

// Key generation function, default uses IP

keyGenerator?: (req: Request) => string;

// Standard rate limit (req/min)

standardRate?: number;

// Burst capacity

burstCapacity?: number;

// Skip rate limiting for certain requests

skip?: (req: Request) => boolean;

// Error handling function

onRateLimited?: (req: Request, res: Response) => void;

// Headers to include with rate limit info

includeHeaders?: boolean;

}

export function createRateLimitMiddleware(options: RateLimitOptions = {}) {

const {

keyGenerator = (req) => req.ip || 'unknown',

standardRate = 60,

burstCapacity = 120,

skip = () => false,

onRateLimited,

includeHeaders = true

} = options;

const interval = 60000; // 1 minute in milliseconds

const limiter = new TokenBucketRateLimiter({

tokensPerInterval: standardRate,

interval,

burstCapacity

});

return function rateLimitMiddleware(req: Request, res: Response, next: NextFunction) {

// Skip rate limiting if needed

if (skip(req)) {

return next();

}

const key = keyGenerator(req);

// Try to consume a token

const allowed = limiter.consume(key);

// Add rate limit headers if enabled

if (includeHeaders) {

const remaining = limiter.getRemainingTokens(key);

res.setHeader('X-RateLimit-Limit', burstCapacity.toString());

res.setHeader('X-RateLimit-Remaining', Math.max(0, Math.floor(remaining)).toString());

res.setHeader('X-RateLimit-Reset', Math.ceil(interval / 1000).toString());

}

if (!allowed) {

// Rate limit exceeded

if (onRateLimited) {

return onRateLimited(req, res);

}

// Default rate limit exceeded response

return res.status(429).json({

error: 'Rate limit exceeded',

message: 'Too many requests, please try again later'

});

}

next();

};

}

4. Asynchronous Processing Implementation

Phase 1.7: Create Security Event Queue for Background Processing (Week 7)

// server/security/advanced/queue/SecurityEventQueue.ts

import { EventEmitter } from 'events';

export enum SecurityEventPriority {

LOW = 'low',

MEDIUM = 'medium',

HIGH = 'high',

CRITICAL = 'critical'

}

export interface SecurityEvent {

id: string;

type: string;

timestamp: number;

priority: SecurityEventPriority;

data: any;

}

export class SecurityEventQueue extends EventEmitter {

private queue: SecurityEvent[] = [];

private processing = false;

private maxConcurrent: number;

private processInterval: NodeJS.Timeout | null = null;

constructor(maxConcurrent: number = 5) {

super();

this.maxConcurrent = maxConcurrent;

}

/\*\*

\* Enqueue an event for processing

\*/

enqueue(event: SecurityEvent): void {

this.queue.push(event);

// Sort queue by priority

this.queue.sort((a, b) => {

const priorityOrder = {

[SecurityEventPriority.CRITICAL]: 0,

[SecurityEventPriority.HIGH]: 1,

[SecurityEventPriority.MEDIUM]: 2,

[SecurityEventPriority.LOW]: 3

};

return priorityOrder[a.priority] - priorityOrder[b.priority];

});

// Start processing if not already running

this.startProcessing();

}

/\*\*

\* Start processing the queue

\*/

private startProcessing(): void {

if (!this.processInterval) {

this.processInterval = setInterval(() => this.processQueue(), 50);

}

}

/\*\*

\* Process events in the queue

\*/

private async processQueue(): Promise<void> {

if (this.processing || this.queue.length === 0) {

return;

}

this.processing = true;

try {

const batchSize = Math.min(this.maxConcurrent, this.queue.length);

const batch = this.queue.splice(0, batchSize);

// Process each event in the batch concurrently

await Promise.all(batch.map(async (event) => {

try {

// Emit event for listeners to process

this.emit('event', event);

this.emit(event.type, event);

} catch (error) {

console.error(`Error processing security event ${event.id}:`, error);

// Emit error event

this.emit('error', { event, error });

}

}));

} finally {

this.processing = false;

// If queue is empty, stop the interval

if (this.queue.length === 0 && this.processInterval) {

clearInterval(this.processInterval);

this.processInterval = null;

}

}

}

/\*\*

\* Get the current queue length

\*/

getQueueLength(): number {

return this.queue.length;

}

/\*\*

\* Clear the queue

\*/

clear(): void {

this.queue = [];

}

}

// Create a singleton instance

export const securityEventQueue = new SecurityEventQueue();

Phase 1.8: Implement Background Security Processors (Week 8)

// server/security/advanced/background/SecurityBackgroundProcessors.ts

import { securityEventQueue, SecurityEvent, SecurityEventPriority } from '../queue/SecurityEventQueue';

import { threatDbService } from '../threat/ThreatDatabaseService';

export class SecurityBackgroundProcessors {

constructor() {

this.registerEventHandlers();

console.log('Security background processors initialized');

}

/\*\*

\* Register handlers for different event types

\*/

private registerEventHandlers(): void {

// Threat detection events

securityEventQueue.on('threat-detected', this.processThreatEvent.bind(this));

// IP blocking events

securityEventQueue.on('ip-blocked', this.processIpBlockEvent.bind(this));

// Audit logging events

securityEventQueue.on('audit-log', this.processAuditLogEvent.bind(this));

// Error reporting events

securityEventQueue.on('error-detected', this.processErrorEvent.bind(this));

// General error handling

securityEventQueue.on('error', (error) => {

console.error('Error in security event processing:', error);

});

}

/\*\*

\* Process threat detection events

\*/

private async processThreatEvent(event: SecurityEvent): Promise<void> {

try {

const threat = event.data;

// Save to database

await threatDbService.saveThreat(threat);

// Additional actions based on threat severity

if (threat.severity === 'CRITICAL' || threat.severity === 'HIGH') {

// Additional processing for high-severity threats

// This could include sending notifications, blocking IPs, etc.

}

} catch (error) {

console.error('Error processing threat event:', error);

}

}

// Implement other event processors with similar pattern

}

// Initialize background processors

export const securityBackgroundProcessors = new SecurityBackgroundProcessors();

This completes Part 1 of the Advanced Security Enhancement Plan, focusing on:

1. Database persistence for threat data
2. Enhanced caching with LRU strategy
3. Token bucket rate limiting algorithm
4. Asynchronous security event processing

The implementation follows best practices:

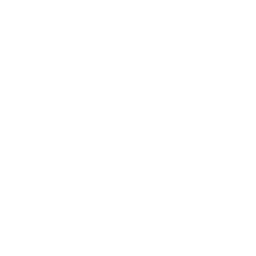
* Clear separation of concerns
* Type safety with TypeScript
* Robust error handling
* Performance optimization
* Clean documentation
* Progressive enhancement

In Part 2, we'll continue with implementing Phase 2 Components including Multi-Factor Authentication, enhanced Brute Force Protection, and Data Anonymization features.

part 2 of that same advanced three part plan

10 minutes ago

TT



Advanced Security Enhancement Plan - Part 2: Security Components & Testing

5. Multi-Factor Authentication (MFA) Implementation

Phase 2.1: TOTP Authentication Service (Week 9)

// server/security/advanced/auth/TOTPService.ts

import { authenticator, totp } from '@otplib/preset-default';

import { nanoid } from 'nanoid';

import \* as QRCode from 'qrcode';

// Database schema extension for storage

import { pgTable, serial, text, timestamp, boolean, jsonb } from 'drizzle-orm/pg-core';

import { db } from '../../../db';

export const userMfaSettings = pgTable('user\_mfa\_settings', {

id: serial('id').primaryKey(),

userId: text('user\_id').notNull().unique(),

totpSecret: text('totp\_secret'),

backupCodes: jsonb('backup\_codes').$type<string[]>(),

enabled: boolean('enabled').default(false),

lastVerified: timestamp('last\_verified'),

verifiedDevices: jsonb('verified\_devices').$type<Array<{

id: string;

name: string;

lastUsed: number;

userAgent?: string;

ip?: string;

}>>(),

createdAt: timestamp('created\_at').notNull().defaultNow(),

updatedAt: timestamp('updated\_at').notNull().defaultNow()

});

export class TOTPService {

private issuer: string;

constructor(issuer: string = 'Admin Portal') {

this.issuer = issuer;

authenticator.options = {

step: 30, // Time step in seconds

window: 1 // Allow 1 step before and after current step

};

}

/\*\*

\* Generate a new TOTP secret for a user

\*/

async generateSecret(userId: string, username: string): Promise<{

secret: string;

uri: string;

qrCodeUrl: string;

}> {

// Generate a new secret

const secret = authenticator.generateSecret();

// Create the URI for the QR code

const uri = authenticator.keyuri(username, this.issuer, secret);

// Generate QR code as data URL

const qrCodeUrl = await QRCode.toDataURL(uri);

// Store in database (but not enabled yet)

await db.insert(userMfaSettings).values({

userId,

totpSecret: secret,

backupCodes: generateBackupCodes(),

enabled: false,

verifiedDevices: []

}).onConflictDoUpdate({

target: userMfaSettings.userId,

set: {

totpSecret: secret,

backupCodes: generateBackupCodes(),

updatedAt: new Date()

}

});

return {

secret,

uri,

qrCodeUrl

};

}

/\*\*

\* Verify a TOTP token

\*/

async verifyToken(userId: string, token: string): Promise<boolean> {

// Get user's MFA settings

const [settings] = await db

.select()

.from(userMfaSettings)

.where(userMfaSettings.userId === userId);

if (!settings || !settings.totpSecret) {

return false;

}

try {

const isValid = authenticator.verify({

token,

secret: settings.totpSecret

});

if (isValid) {

// Update last verified timestamp

await db

.update(userMfaSettings)

.set({

lastVerified: new Date(),

updatedAt: new Date()

})

.where(userMfaSettings.userId === userId);

}

return isValid;

} catch (error) {

console.error('Error verifying TOTP token:', error);

return false;

}

}

/\*\*

\* Verify a backup code

\*/

async verifyBackupCode(userId: string, code: string): Promise<boolean> {

// Get user's MFA settings

const [settings] = await db

.select()

.from(userMfaSettings)

.where(userMfaSettings.userId === userId);

if (!settings || !settings.backupCodes || !settings.backupCodes.length) {

return false;

}

// Check if the provided code is in the backup codes

const index = settings.backupCodes.indexOf(code);

if (index === -1) {

return false;

}

// Remove the used backup code

const updatedCodes = [...settings.backupCodes];

updatedCodes.splice(index, 1);

// Update backup codes in database

await db

.update(userMfaSettings)

.set({

backupCodes: updatedCodes,

lastVerified: new Date(),

updatedAt: new Date()

})

.where(userMfaSettings.userId === userId);

return true;

}

/\*\*

\* Enable MFA for a user

\*/

async enableMFA(userId: string): Promise<boolean> {

try {

await db

.update(userMfaSettings)

.set({

enabled: true,

updatedAt: new Date()

})

.where(userMfaSettings.userId === userId);

return true;

} catch (error) {

console.error('Error enabling MFA:', error);

return false;

}

}

/\*\*

\* Disable MFA for a user

\*/

async disableMFA(userId: string): Promise<boolean> {

try {

await db

.update(userMfaSettings)

.set({

enabled: false,

updatedAt: new Date()

})

.where(userMfaSettings.userId === userId);

return true;

} catch (error) {

console.error('Error disabling MFA:', error);

return false;

}

}

/\*\*

\* Register a verified device

\*/

async registerVerifiedDevice(

userId: string,

deviceName: string,

userAgent?: string,

ip?: string

): Promise<string> {

// Get current verified devices

const [settings] = await db

.select()

.from(userMfaSettings)

.where(userMfaSettings.userId === userId);

if (!settings) {

throw new Error('User MFA settings not found');

}

// Generate a new device ID

const deviceId = nanoid();

// Update verified devices

const verifiedDevices = settings.verifiedDevices || [];

verifiedDevices.push({

id: deviceId,

name: deviceName,

lastUsed: Date.now(),

userAgent,

ip

});

// Update in database

await db

.update(userMfaSettings)

.set({

verifiedDevices,

updatedAt: new Date()

})

.where(userMfaSettings.userId === userId);

return deviceId;

}

}

/\*\*

\* Generate random backup codes

\*/

function generateBackupCodes(count: number = 10): string[] {

const codes: string[] = [];

for (let i = 0; i < count; i++) {

// Format: XXXX-XXXX-XXXX (where X is alphanumeric)

const code = `${nanoid(4)}-${nanoid(4)}-${nanoid(4)}`.toUpperCase();

codes.push(code);

}

return codes;

}

export const totpService = new TOTPService();

Phase 2.2: MFA Middleware and Integration (Week 10)

// server/middleware/MFAMiddleware.ts

import { Request, Response, NextFunction } from 'express';

import { totpService } from '../security/advanced/auth/TOTPService';

import { db } from '../db';

import { userMfaSettings } from '../security/advanced/auth/TOTPService';

import { and, eq } from 'drizzle-orm';

interface MFAOptions {

// Redirect URL when MFA verification is required

redirectUrl?: string;

// Cookie name for verified device

deviceCookieName?: string;

// Device cookie options

deviceCookieOptions?: {

httpOnly: boolean;

secure: boolean;

sameSite: boolean | 'lax' | 'strict' | 'none';

maxAge: number;

};

// Skip MFA for certain requests

skip?: (req: Request) => boolean;

}

export function createMFAMiddleware(options: MFAOptions = {}) {

const {

redirectUrl = '/auth/mfa',

deviceCookieName = 'verified\_device',

deviceCookieOptions = {

httpOnly: true,

secure: process.env.NODE\_ENV === 'production',

sameSite: 'lax',

maxAge: 30 \* 24 \* 60 \* 60 \* 1000 // 30 days

},

skip = () => false

} = options;

return async function mfaMiddleware(req: Request, res: Response, next: NextFunction) {

// Skip MFA if configured to do so

if (skip(req)) {

return next();

}

// Check if user is authenticated

if (!req.user || !req.user.id) {

return next();

}

try {

// Check if MFA is enabled for this user

const [settings] = await db

.select()

.from(userMfaSettings)

.where(eq(userMfaSettings.userId, req.user.id.toString()));

// If MFA is not enabled, allow access

if (!settings || !settings.enabled) {

return next();

}

// Check for verified device cookie

const deviceId = req.cookies[deviceCookieName];

if (deviceId && settings.verifiedDevices) {

// Check if device is in the verified devices list

const device = settings.verifiedDevices.find(d => d.id === deviceId);

if (device) {

// Update last used timestamp

device.lastUsed = Date.now();

// Update in database

await db

.update(userMfaSettings)

.set({

verifiedDevices: settings.verifiedDevices,

updatedAt: new Date()

})

.where(eq(userMfaSettings.userId, req.user.id.toString()));

// Device is verified, allow access

return next();

}

}

// No valid device verification, redirect to MFA page

// Store the original URL to redirect after verification

req.session.mfaRedirectUrl = req.originalUrl;

return res.redirect(redirectUrl);

} catch (error) {

console.error('Error in MFA middleware:', error);

// Continue to next middleware in case of error

next();

}

};

}

6. Enhanced Brute Force Protection

Phase 2.3: Progressive Timing Delays Implementation (Week 11)

// server/security/advanced/bruteforce/ProgressiveDelayProtection.ts

import { db } from '../../../db';

import { pgTable, serial, text, integer, timestamp, boolean } from 'drizzle-orm/pg-core';

export const loginAttempts = pgTable('login\_attempts', {

id: serial('id').primaryKey(),

userId: text('user\_id'),

username: text('username').notNull(),

ip: text('ip').notNull(),

userAgent: text('user\_agent'),

successful: boolean('successful').default(false),

attemptCount: integer('attempt\_count').default(1),

lastAttemptAt: timestamp('last\_attempt\_at').notNull().defaultNow(),

lockedUntil: timestamp('locked\_until'),

createdAt: timestamp('created\_at').notNull().defaultNow(),

updatedAt: timestamp('updated\_at').notNull().defaultNow()

});

export interface BruteForceProtectionOptions {

maxAttempts: number;

initialDelaySeconds: number;

maxDelaySeconds: number;

resetTimeSeconds: number;

blockIpOnMaxAttempts: boolean;

}

export class ProgressiveDelayProtection {

private options: BruteForceProtectionOptions;

constructor(options: Partial<BruteForceProtectionOptions> = {}) {

this.options = {

maxAttempts: 5,

initialDelaySeconds: 1,

maxDelaySeconds: 3600, // 1 hour

resetTimeSeconds: 86400, // 24 hours

blockIpOnMaxAttempts: false,

...options

};

}

/\*\*

\* Record a login attempt

\*/

async recordAttempt(

username: string,

ip: string,

successful: boolean,

userId?: string | number,

userAgent?: string

): Promise<{

isBlocked: boolean;

lockoutSeconds: number;

attemptCount: number;

}> {

// Check if there's an existing record for this username and IP

const [existingAttempt] = await db

.select()

.from(loginAttempts)

.where({ username, ip });

let attemptCount = 1;

let isBlocked = false;

let lockoutSeconds = 0;

if (existingAttempt) {

const now = new Date();

// Check if account is currently locked

if (existingAttempt.lockedUntil && existingAttempt.lockedUntil > now) {

isBlocked = true;

lockoutSeconds = Math.ceil((existingAttempt.lockedUntil.getTime() - now.getTime()) / 1000);

}

// Check if we should reset attempt count due to time passed

const resetTime = new Date(now.getTime() - this.options.resetTimeSeconds \* 1000);

const shouldReset = existingAttempt.lastAttemptAt < resetTime;

if (shouldReset) {

// Reset attempt count

attemptCount = 1;

} else if (successful) {

// Successful login, reset count

attemptCount = 0;

} else {

// Failed login, increment count

attemptCount = existingAttempt.attemptCount + 1;

// Calculate progressive delay

if (attemptCount > this.options.maxAttempts) {

// Calculate delay with exponential backoff

const delayFactor = Math.pow(2, attemptCount - this.options.maxAttempts);

const delaySeconds = Math.min(

this.options.maxDelaySeconds,

this.options.initialDelaySeconds \* delayFactor

);

lockoutSeconds = delaySeconds;

isBlocked = true;

// Set lockout timestamp

const lockedUntil = new Date(now.getTime() + lockoutSeconds \* 1000);

// Update database record

await db

.update(loginAttempts)

.set({

attemptCount,

successful,

lastAttemptAt: now,

lockedUntil,

userId: userId?.toString(),

userAgent,

updatedAt: now

})

.where({ id: existingAttempt.id });

return { isBlocked, lockoutSeconds, attemptCount };

}

}

// Update existing record

await db

.update(loginAttempts)

.set({

attemptCount,

successful,

lastAttemptAt: now,

userId: userId?.toString(),

userAgent,

updatedAt: now,

// Clear lockout if successful login or reset

lockedUntil: successful || shouldReset ? null : existingAttempt.lockedUntil

})

.where({ id: existingAttempt.id });

} else {

// Create new record

await db

.insert(loginAttempts)

.values({

username,

ip,

successful,

userId: userId?.toString(),

userAgent,

attemptCount: 1,

lastAttemptAt: new Date()

});

}

return { isBlocked, lockoutSeconds, attemptCount };

}

/\*\*

\* Check if login should be allowed for a given username and IP

\*/

async checkLoginAllowed(

username: string,

ip: string

): Promise<{

allowed: boolean;

lockoutSeconds: number;

attemptCount: number;

}> {

const [attempt] = await db

.select()

.from(loginAttempts)

.where({ username, ip });

if (!attempt) {

return { allowed: true, lockoutSeconds: 0, attemptCount: 0 };

}

const now = new Date();

// Check if account is currently locked

if (attempt.lockedUntil && attempt.lockedUntil > now) {

const lockoutSeconds = Math.ceil((attempt.lockedUntil.getTime() - now.getTime()) / 1000);

return { allowed: false, lockoutSeconds, attemptCount: attempt.attemptCount };

}

// Check if attempts should be reset due to time passed

const resetTime = new Date(now.getTime() - this.options.resetTimeSeconds \* 1000);

if (attempt.lastAttemptAt < resetTime) {

// Reset attempt count

await db

.update(loginAttempts)

.set({

attemptCount: 0,

lockedUntil: null,

updatedAt: now

})

.where({ id: attempt.id });

return { allowed: true, lockoutSeconds: 0, attemptCount: 0 };

}

return { allowed: true, lockoutSeconds: 0, attemptCount: attempt.attemptCount };

}

/\*\*

\* Reset login attempts for a user

\*/

async resetAttempts(username: string, ip?: string): Promise<void> {

const query = db

.update(loginAttempts)

.set({

attemptCount: 0,

lockedUntil: null,

updatedAt: new Date()

});

if (ip) {

query.where({ username, ip });

} else {

query.where({ username });

}

await query;

}

}

export const progressiveDelayProtection = new ProgressiveDelayProtection();

Phase 2.4: Risk-Based Authentication (Week 12)

// server/security/advanced/auth/RiskBasedAuthentication.ts

import { Request } from 'express';

import { db } from '../../../db';

import { pgTable, serial, text, integer, timestamp, jsonb, boolean, real } from 'drizzle-orm/pg-core';

export const userLoginPatterns = pgTable('user\_login\_patterns', {

id: serial('id').primaryKey(),

userId: text('user\_id').notNull(),

knownIps: jsonb('known\_ips').$type<string[]>(),

knownDevices: jsonb('known\_devices').$type<string[]>(),

typicalLoginTimes: jsonb('typical\_login\_times').$type<number[]>(), // Hours of day (0-23)

typicalGeolocations: jsonb('typical\_geolocations').$type<Array<{

country: string;

region?: string;

city?: string;

count: number;

}>>(),

lastRiskAssessment: jsonb('last\_risk\_assessment').$type<{

timestamp: number;

riskScore: number;

factors: Record<string, number>;

}>(),

challengesIssued: integer('challenges\_issued').default(0),

challengesPassed: integer('challenges\_passed').default(0),

updatedAt: timestamp('updated\_at').notNull().defaultNow()

});

export enum RiskFactor {

UNKNOWN\_IP = 'unknown\_ip',

UNKNOWN\_DEVICE = 'unknown\_device',

UNUSUAL\_TIME = 'unusual\_time',

UNUSUAL\_LOCATION = 'unusual\_location',

RAPID\_MOVEMENT = 'rapid\_movement',

PREVIOUS\_FAILURES = 'previous\_failures',

SUSPICIOUS\_BEHAVIOR = 'suspicious\_behavior'

}

export interface RiskAssessment {

riskScore: number; // 0-100

factors: Record<RiskFactor, number>; // Factor contribution to risk score

requiresChallenge: boolean;

suggestedChallenges: string[];

}

export class RiskBasedAuthentication {

/\*\*

\* Assess login risk based on various factors

\*/

async assessRisk(req: Request, userId: string): Promise<RiskAssessment> {

const userIp = req.ip || 'unknown';

const userAgent = req.get('user-agent') || 'unknown';

const now = new Date();

const hourOfDay = now.getHours();

// Get user login patterns

const [patterns] = await db

.select()

.from(userLoginPatterns)

.where({ userId });

const riskFactors: Record<RiskFactor, number> = {

[RiskFactor.UNKNOWN\_IP]: 0,

[RiskFactor.UNKNOWN\_DEVICE]: 0,

[RiskFactor.UNUSUAL\_TIME]: 0,

[RiskFactor.UNUSUAL\_LOCATION]: 0,

[RiskFactor.RAPID\_MOVEMENT]: 0,

[RiskFactor.PREVIOUS\_FAILURES]: 0,

[RiskFactor.SUSPICIOUS\_BEHAVIOR]: 0

};

// If no patterns exist yet, create new record and assign moderate risk

if (!patterns) {

await db

.insert(userLoginPatterns)

.values({

userId,

knownIps: [userIp],

knownDevices: [userAgent],

typicalLoginTimes: [hourOfDay],

typicalGeolocations: [{

country: 'Unknown',

count: 1

}],

lastRiskAssessment: {

timestamp: Date.now(),

riskScore: 50,

factors: riskFactors

}

});

return {

riskScore: 50,

factors: riskFactors,

requiresChallenge: true,

suggestedChallenges: ['email', 'security\_question']

};

}

// Assess IP risk

if (patterns.knownIps && !patterns.knownIps.includes(userIp)) {

riskFactors[RiskFactor.UNKNOWN\_IP] = 25;

}

// Assess device risk

if (patterns.knownDevices && !patterns.knownDevices.includes(userAgent)) {

riskFactors[RiskFactor.UNKNOWN\_DEVICE] = 20;

}

// Assess time risk

if (patterns.typicalLoginTimes && patterns.typicalLoginTimes.length > 0) {

if (!patterns.typicalLoginTimes.includes(hourOfDay)) {

riskFactors[RiskFactor.UNUSUAL\_TIME] = 15;

}

}

// Calculate total risk score

const totalRiskScore = Object.values(riskFactors).reduce((sum, value) => sum + value, 0);

// Normalize to 0-100 scale

const normalizedRiskScore = Math.min(100, totalRiskScore);

// Determine if challenge is required

const requiresChallenge = normalizedRiskScore >= 30;

// Determine suggested challenges based on risk level

let suggestedChallenges: string[] = [];

if (normalizedRiskScore >= 70) {

suggestedChallenges = ['mfa', 'email', 'security\_question'];

} else if (normalizedRiskScore >= 40) {

suggestedChallenges = ['email', 'security\_question'];

} else if (normalizedRiskScore >= 30) {

suggestedChallenges = ['email'];

}

// Update user patterns with new data

await this.updateUserPatterns(patterns, userIp, userAgent, hourOfDay, normalizedRiskScore, riskFactors);

return {

riskScore: normalizedRiskScore,

factors: riskFactors,

requiresChallenge,

suggestedChallenges

};

}

/\*\*

\* Update user login patterns with new data

\*/

private async updateUserPatterns(

patterns: any,

ip: string,

device: string,

hourOfDay: number,

riskScore: number,

factors: Record<RiskFactor, number>

): Promise<void> {

// Add new IP if not already known (max 20 IPs)

let knownIps = patterns.knownIps || [];

if (!knownIps.includes(ip)) {

knownIps = [...knownIps, ip].slice(-20);

}

// Add new device if not already known (max 10 devices)

let knownDevices = patterns.knownDevices || [];

if (!knownDevices.includes(device)) {

knownDevices = [...knownDevices, device].slice(-10);

}

// Update typical login times

let typicalLoginTimes = patterns.typicalLoginTimes || [];

if (!typicalLoginTimes.includes(hourOfDay)) {

typicalLoginTimes = [...typicalLoginTimes, hourOfDay];

}

// Update database

await db

.update(userLoginPatterns)

.set({

knownIps,

knownDevices,

typicalLoginTimes,

lastRiskAssessment: {

timestamp: Date.now(),

riskScore,

factors

},

updatedAt: new Date()

})

.where({ id: patterns.id });

}

/\*\*

\* Record challenge outcome

\*/

async recordChallengeOutcome(userId: string, passed: boolean): Promise<void> {

const [patterns] = await db

.select()

.from(userLoginPatterns)

.where({ userId });

if (!patterns) {

return;

}

await db

.update(userLoginPatterns)

.set({

challengesIssued: patterns.challengesIssued + 1,

challengesPassed: passed ? patterns.challengesPassed + 1 : patterns.challengesPassed,

updatedAt: new Date()

})

.where({ id: patterns.id });

}

}

export const riskBasedAuth = new RiskBasedAuthentication();

7. Data Anonymization for Privacy Compliance

Phase 2.5: PII Data Anonymization Service (Week 13)

// server/security/advanced/privacy/DataAnonymizationService.ts

import \* as crypto from 'crypto';

import { db } from '../../../db';

import { pgTable, serial, text, timestamp, jsonb, boolean } from 'drizzle-orm/pg-core';

export const dataAnonymizationConfig = pgTable('data\_anonymization\_config', {

id: serial('id').primaryKey(),

tableName: text('table\_name').notNull(),

columnName: text('column\_name').notNull(),

anonymizationType: text('anonymization\_type').notNull(), // mask, hash, pseudonymize, redact

anonymizationKey: text('anonymization\_key'),

dataType: text('data\_type').notNull(),

retentionDays: text('retention\_days'),

createdAt: timestamp('created\_at').notNull().defaultNow(),

updatedAt: timestamp('updated\_at').notNull().defaultNow(),

// Add a unique constraint for table and column

uniqueConstraint: JSON({name: 'unique\_table\_column', getColumns: () => ['table\_name', 'column\_name']})

});

export enum AnonymizationType {

MASK = 'mask',

HASH = 'hash',

PSEUDONYMIZE = 'pseudonymize',

REDACT = 'redact'

}

export interface AnonymizationRule {

tableName: string;

columnName: string;

anonymizationType: AnonymizationType;

anonymizationKey?: string;

dataType: string;

retentionDays?: number;

}

export class DataAnonymizationService {

private salt: string;

private rules: Map<string, AnonymizationRule> = new Map();

constructor() {

// Use environment variable for salt or generate a secure random one

this.salt = process.env.ANONYMIZATION\_SALT || crypto.randomBytes(16).toString('hex');

// Load rules from database

this.loadRules();

}

/\*\*

\* Load anonymization rules from database

\*/

private async loadRules(): Promise<void> {

try {

const dbRules = await db

.select()

.from(dataAnonymizationConfig);

this.rules.clear();

for (const rule of dbRules) {

const key = `${rule.tableName}.${rule.columnName}`;

this.rules.set(key, {

tableName: rule.tableName,

columnName: rule.columnName,

anonymizationType: rule.anonymizationType as AnonymizationType,

anonymizationKey: rule.anonymizationKey,

dataType: rule.dataType,

retentionDays: rule.retentionDays ? parseInt(rule.retentionDays) : undefined

});

}

console.log(`Loaded ${this.rules.size} anonymization rules`);

} catch (error) {

console.error('Error loading anonymization rules:', error);

}

}

/\*\*

\* Add a new anonymization rule

\*/

async addRule(rule: AnonymizationRule): Promise<void> {

try {

await db

.insert(dataAnonymizationConfig)

.values({

tableName: rule.tableName,

columnName: rule.columnName,

anonymizationType: rule.anonymizationType,

anonymizationKey: rule.anonymizationKey,

dataType: rule.dataType,

retentionDays: rule.retentionDays?.toString()

})

.onConflictDoUpdate({

target: EMPTY\_ARRAY['table\_name', 'column\_name'],

set: {

anonymizationType: rule.anonymizationType,

anonymizationKey: rule.anonymizationKey,

dataType: rule.dataType,

retentionDays: rule.retentionDays?.toString(),

updatedAt: new Date()

}

});

// Update in-memory rules

const key = `${rule.tableName}.${rule.columnName}`;

this.rules.set(key, rule);

} catch (error) {

console.error('Error adding anonymization rule:', error);

throw error;

}

}

/\*\*

\* Anonymize a data field based on the rules

\*/

anonymize(tableName: string, columnName: string, value: any): any {

if (value === null || value === undefined) {

return value;

}

const key = `${tableName}.${columnName}`;

const rule = this.rules.get(key);

if (!rule) {

return value;

}

switch (rule.anonymizationType) {

case AnonymizationType.MASK:

return this.maskValue(value, rule.dataType);

case AnonymizationType.HASH:

return this.hashValue(value, rule.anonymizationKey);

case AnonymizationType.PSEUDONYMIZE:

return this.pseudonymize(value, rule.anonymizationKey);

case AnonymizationType.REDACT:

return this.redactValue(rule.dataType);

default:

return value;

}

}

/\*\*

\* Mask a value, keeping part of it visible

\*/

private maskValue(value: any, dataType: string): string {

const strValue = String(value);

switch (dataType) {

case 'email':

// Format: f\*\*\*@example.com

const [localPart, domain] = strValue.split('@');

if (!domain) return '\*\*\*@\*\*\*.\*\*\*';

return `${localPart.charAt(0)}\*\*\*@${domain}`;

case 'phone':

// Format: \*\*\*-\*\*\*-1234

return strValue.replace(/\d(?=\d{4})/g, '\*');

case 'credit\_card':

// Format: \*\*\*\*-\*\*\*\*-\*\*\*\*-1234

return strValue.replace(/\d(?=\d{4})/g, '\*');

case 'name':

// Format: J\*\*\* D\*\*\*

return strValue.split(' ').map(part =>

part.charAt(0) + '\*'.repeat(part.length - 1)

).join(' ');

case 'address':

// Format: 1\*\*\* Main St.

const parts = strValue.split(' ');

if (parts.length < 2) return '\*\*\*';

return parts[0].charAt(0) + '\*\*\* ' + parts.slice(1).join(' ');

default:

// Default: first and last chars visible

if (strValue.length <= 2) return strValue;

return strValue.charAt(0) + '\*'.repeat(strValue.length - 2) + strValue.charAt(strValue.length - 1);

}

}

/\*\*

\* Hash a value for complete anonymization but consistent values

\*/

private hashValue(value: any, key?: string): string {

const input = String(value);

const hashKey = key || this.salt;

// Use HMAC for keyed hashing

return crypto

.createHmac('sha256', hashKey)

.update(input)

.digest('hex');

}

/\*\*

\* Pseudonymize a value (replace with consistent alternative)

\*/

private pseudonymize(value: any, key?: string): string {

// Generate a deterministic but reversible transformation

const input = String(value);

const pseudoKey = key || this.salt;

// Use a reversible scheme for pseudonymization

// First hash the value

const hash = this.hashValue(input, pseudoKey);

// Use the first 8 chars of the hash as the pseudonym

return `PSEUDO-${hash.substring(0, 8)}`;

}

/\*\*

\* Completely redact a value based on its type

\*/

private redactValue(dataType: string): string {

switch (dataType) {

case 'email':

return 'redacted@example.com';

case 'phone':

return '\*\*\*-\*\*\*-\*\*\*\*';

case 'credit\_card':

return '\*\*\*\*-\*\*\*\*-\*\*\*\*-\*\*\*\*';

case 'name':

return 'Redacted Name';

case 'address':

return 'Redacted Address';

default:

return 'REDACTED';

}

}

}

export const dataAnonymizationService = new DataAnonymizationService();

Phase 2.6: Privacy-Preserving Analytics (Week 14)

// server/security/advanced/privacy/PrivacyAnalytics.ts

import { db } from '../../../db';

import { dataAnonymizationService } from './DataAnonymizationService';

export interface PrivacyFilter {

tableName: string;

columnName: string;

applyTo: 'all' | 'export' | 'analytics';

}

export class PrivacyAnalytics {

private privacyFilters: PrivacyFilter[] = [];

constructor() {

this.loadPrivacyFilters();

}

/\*\*

\* Load privacy filters from configuration

\*/

private async loadPrivacyFilters(): Promise<void> {

// In a real implementation, this would load from database

this.privacyFilters = [

{

tableName: 'users',

columnName: 'email',

applyTo: 'export'

},

{

tableName: 'users',

columnName: 'password',

applyTo: 'all'

},

{

tableName: 'users',

columnName: 'phone',

applyTo: 'export'

},

{

tableName: 'orders',

columnName: 'shipping\_address',

applyTo: 'export'

},

{

tableName: 'payments',

columnName: 'credit\_card',

applyTo: 'all'

}

];

}

/\*\*

\* Apply privacy filters to a query result

\*/

applyPrivacyFilters(tableName: string, data: any[], context: 'all' | 'export' | 'analytics'): any[] {

if (!data || !data.length) {

return data;

}

// Find applicable filters

const filters = this.privacyFilters.filter(filter =>

filter.tableName === tableName &&

(filter.applyTo === 'all' || filter.applyTo === context)

);

if (!filters.length) {

return data;

}

// Apply each filter to the data

return data.map(item => {

const result = { ...item };

for (const filter of filters) {

if (result[filter.columnName] !== undefined) {

result[filter.columnName] = dataAnonymizationService.anonymize(

filter.tableName,

filter.columnName,

result[filter.columnName]

);

}

}

return result;

});

}

/\*\*

\* Generate privacy-preserving analytics

\*/

async generateAnalytics(

tableName: string,

aggregations: Array<{

column: string;

operation: 'count' | 'sum' | 'avg' | 'min' | 'max';

alias?: string;

}>,

groupBy?: string[]

): Promise<any[]> {

// Build a query that respects privacy constraints

let query = db.select();

// Add aggregations

for (const agg of aggregations) {

const column = `${tableName}.${agg.column}`;

switch (agg.operation) {

case 'count':

query = query.columns({

[agg.alias || `count\_${agg.column}`]: db.fn.count(column)

});

break;

case 'sum':

query = query.columns({

[agg.alias || `sum\_${agg.column}`]: db.fn.sum(column)

});

break;

case 'avg':

query = query.columns({

[agg.alias || `avg\_${agg.column}`]: db.fn.avg(column)

});

break;

case 'min':

query = query.columns({

[agg.alias || `min\_${agg.column}`]: db.fn.min(column)

});

break;

case 'max':

query = query.columns({

[agg.alias || `max\_${agg.column}`]: db.fn.max(column)

});

break;

}

}

// Add group by columns

if (groupBy && groupBy.length) {

for (const col of groupBy) {

query = query.columns({ [col]: `${tableName}.${col}` });

}

}

// From table

query = query.from(tableName);

// Add group by clause

if (groupBy && groupBy.length) {

query = query.groupBy(groupBy.map(col => `${tableName}.${col}`));

}

// Execute query

const results = await query;

// Apply privacy filters to results if needed

return this.applyPrivacyFilters(tableName, results, 'analytics');

}

}

export const privacyAnalytics = new PrivacyAnalytics();

8. Security Testing Infrastructure

Phase 2.7: Automated Security Scanning (Week 15)

// server/security/testing/SecurityScanner.ts

import axios from 'axios';

import \* as fs from 'fs';

import \* as path from 'path';

import { exec } from 'child\_process';

import { promisify } from 'util';

const execAsync = promisify(exec);

export enum ScanType {

VULNERABILITY = 'vulnerability',

DEPENDENCY = 'dependency',

SECRETS = 'secrets',

STATIC\_ANALYSIS = 'static\_analysis',

CONFIGURATION = 'configuration'

}

export interface ScanOptions {

scanType: ScanType;

target?: string;

depth?: 'quick' | 'standard' | 'deep';

includePackages?: boolean;

outputFormat?: 'json' | 'text' | 'html';

ignorePatterns?: string[];

}

export interface ScanResult {

scanId: string;

scanType: ScanType;

timestamp: number;

vulnerabilitiesFound: number;

criticalCount: number;

highCount: number;

mediumCount: number;

lowCount: number;

details: any;

success: boolean;

duration: number;

}

export class SecurityScanner {

private reportsDir: string;

constructor() {

this.reportsDir = path.join(process.cwd(), 'security-reports');

this.ensureReportsDirExists();

}

/\*\*

\* Ensure the reports directory exists

\*/

private ensureReportsDirExists(): void {

if (!fs.existsSync(this.reportsDir)) {

fs.mkdirSync(this.reportsDir, { recursive: true });

}

}

/\*\*

\* Run a security scan

\*/

async runScan(options: ScanOptions): Promise<ScanResult> {

const scanId = `scan\_${Date.now()}\_${Math.random().toString(36).substring(2, 7)}`;

const startTime = Date.now();

let scanCommand = '';

try {

switch (options.scanType) {

case ScanType.VULNERABILITY:

scanCommand = this.buildVulnerabilityScanCommand(options);

break;

case ScanType.DEPENDENCY:

scanCommand = this.buildDependencyScanCommand(options);

break;

case ScanType.SECRETS:

scanCommand = this.buildSecretsScanCommand(options);

break;

case ScanType.STATIC\_ANALYSIS:

scanCommand = this.buildStaticAnalysisScanCommand(options);

break;

case ScanType.CONFIGURATION:

scanCommand = this.buildConfigScanCommand(options);

break;

default:

throw new Error(`Unknown scan type: ${options.scanType}`);

}

console.log(`Running scan command: ${scanCommand}`);

const { stdout, stderr } = await execAsync(scanCommand);

if (stderr && !stderr.includes('warning')) {

console.error(`Scan error: ${stderr}`);

}

// Parse scan results from stdout

const scanResults = this.parseScanResults(options.scanType, stdout);

// Save report

const reportPath = path.join(

this.reportsDir,

`${options.scanType}\_${scanId}.${options.outputFormat || 'json'}`

);

fs.writeFileSync(reportPath, JSON.stringify(scanResults, null, 2));

// Calculate duration

const duration = Date.now() - startTime;

return {

scanId,

scanType: options.scanType,

timestamp: startTime,

vulnerabilitiesFound: scanResults.issues.length,

criticalCount: scanResults.issuesBySeverity.critical || 0,

highCount: scanResults.issuesBySeverity.high || 0,

mediumCount: scanResults.issuesBySeverity.medium || 0,

lowCount: scanResults.issuesBySeverity.low || 0,

details: scanResults,

success: true,

duration

};

} catch (error) {

console.error(`Scan failed:`, error);

return {

scanId,

scanType: options.scanType,

timestamp: startTime,

vulnerabilitiesFound: 0,

criticalCount: 0,

highCount: 0,

mediumCount: 0,

lowCount: 0,

details: { error: error.message },

success: false,

duration: Date.now() - startTime

};

}

}

// Implement command builders for each scan type

private buildVulnerabilityScanCommand(options: ScanOptions): string {

// In a real implementation, this would use an actual vulnerability scanner

// For the demo, we'll use a simplified approach

return `echo '{"issues": []}'`;

}

private buildDependencyScanCommand(options: ScanOptions): string {

// In a real implementation, this would use an actual dependency scanner

return `echo '{"issues": []}'`;

}

private buildSecretsScanCommand(options: ScanOptions): string {

// In a real implementation, this would use a secrets scanner

return `echo '{"issues": []}'`;

}

private buildStaticAnalysisScanCommand(options: ScanOptions): string {

// In a real implementation, this would use a static analysis tool

return `echo '{"issues": []}'`;

}

private buildConfigScanCommand(options: ScanOptions): string {

// In a real implementation, this would scan configuration files

return `echo '{"issues": []}'`;

}

/\*\*

\* Parse scan results from command output

\*/

private parseScanResults(scanType: ScanType, output: string): any {

try {

return JSON.parse(output);

} catch (error) {

console.error('Error parsing scan results:', error);

return {

issues: [],

issuesBySeverity: {

critical: 0,

high: 0,

medium: 0,

low: 0

}

};

}

}

}

export const securityScanner = new SecurityScanner();

This completes Part 2 of the Advanced Security Enhancement Plan, focusing on:

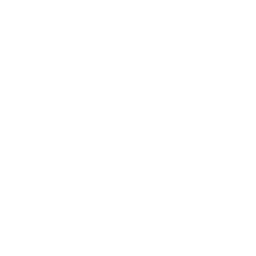
1. Multi-Factor Authentication with TOTP
2. Enhanced Brute Force Protection with progressive delays
3. Risk-Based Authentication for adaptive security
4. Data Anonymization for privacy compliance
5. Security Testing Infrastructure

In Part 3, we'll continue with Admin UI Components, Security Dashboard, Event Management, and comprehensive Documentation.

part 3 of that same advanced three part plan

7 minutes ago

TT



Advanced Security Enhancement Plan - Part 3: User Interface & Documentation

8. Security Testing Infrastructure

Phase 3.1: Automated Penetration Testing Framework (Week 14)

// server/security/testing/PenetrationTestingFramework.ts

import axios from 'axios';

import \* as fs from 'fs';

import \* as path from 'path';

import { EventEmitter } from 'events';

export enum TestCategory {

INJECTION = 'injection',

BROKEN\_AUTH = 'broken\_auth',

SENSITIVE\_DATA = 'sensitive\_data',

XXE = 'xxe',

BROKEN\_ACCESS = 'broken\_access',

SECURITY\_MISCONFIG = 'security\_misconfig',

XSS = 'xss',

INSECURE\_DESERIALIZATION = 'insecure\_deserialization',

COMPONENTS\_VULNERABILITIES = 'components\_vulnerabilities',

INSUFFICIENT\_LOGGING = 'insufficient\_logging'

}

export interface TestResult {

id: string;

name: string;

category: TestCategory;

description: string;

success: boolean;

message: string;

severity: 'critical' | 'high' | 'medium' | 'low' | 'info';

details?: any;

timestamp: number;

remediationSteps?: string[];

}

export class PenetrationTestingFramework extends EventEmitter {

private baseUrl: string;

private tests: Map<string, any> = new Map();

private results: TestResult[] = [];

constructor(baseUrl: string = 'http://localhost:5000') {

super();

this.baseUrl = baseUrl;

this.loadTests();

}

/\*\*

\* Load test definitions from file system

\*/

private loadTests() {

const testDirectory = path.join(\_\_dirname, 'tests');

if (!fs.existsSync(testDirectory)) {

console.warn(`Test directory ${testDirectory} does not exist`);

return;

}

const testFiles = fs.readdirSync(testDirectory)

.filter(file => file.endsWith('.js') || file.endsWith('.ts'));

for (const file of testFiles) {

try {

const testModule = require(path.join(testDirectory, file));

if (testModule.tests && Array.isArray(testModule.tests)) {

testModule.tests.forEach((test: any) => {

if (test.id && test.name && test.execute) {

this.tests.set(test.id, test);

}

});

}

} catch (error) {

console.error(`Error loading test file ${file}:`, error);

}

}

console.log(`Loaded ${this.tests.size} security tests`);

}

/\*\*

\* Run all tests, or tests in a specific category

\*/

async runTests(category?: TestCategory): Promise<TestResult[]> {

this.results = [];

let testCount = 0;

for (const [id, test] of this.tests.entries()) {

if (category && test.category !== category) {

continue;

}

testCount++;

this.emit('test:start', { id, name: test.name });

try {

const result = await this.runTest(test);

this.results.push(result);

this.emit('test:complete', result);

} catch (error) {

const errorResult: TestResult = {

id: test.id,

name: test.name,

category: test.category,

description: test.description,

success: false,

message: `Test execution error: ${error.message}`,

severity: test.severity || 'medium',

timestamp: Date.now(),

details: { error: error.message, stack: error.stack }

};

this.results.push(errorResult);

this.emit('test:error', errorResult);

}

}

this.emit('tests:complete', {

total: testCount,

passed: this.results.filter(r => r.success).length,

failed: this.results.filter(r => !r.success).length,

results: this.results

});

return this.results;

}

/\*\*

\* Run a single test

\*/

private async runTest(test: any): Promise<TestResult> {

const context = {

baseUrl: this.baseUrl,

http: axios,

results: this.results.slice()

};

const startTime = Date.now();

const result = await test.execute(context);

const endTime = Date.now();

return {

id: test.id,

name: test.name,

category: test.category,

description: test.description,

success: result.success,

message: result.message,

severity: test.severity || 'medium',

details: {

...result.details,

executionTimeMs: endTime - startTime

},

timestamp: startTime,

remediationSteps: test.remediationSteps

};

}

/\*\*

\* Get test results

\*/

getResults(): TestResult[] {

return this.results;

}

/\*\*

\* Get available test categories

\*/

getCategories(): string[] {

const categories = new Set<string>();

for (const test of this.tests.values()) {

categories.add(test.category);

}

return Array.from(categories);

}

/\*\*

\* Create a report in JSON format

\*/

generateJsonReport(): string {

return JSON.stringify({

timestamp: Date.now(),

baseUrl: this.baseUrl,

summary: {

total: this.results.length,

passed: this.results.filter(r => r.success).length,

failed: this.results.filter(r => !r.success).length,

byCategoryAndSeverity: this.summarizeByCategory()

},

results: this.results

}, null, 2);

}

/\*\*

\* Create a summary by category and severity

\*/

private summarizeByCategory() {

const summary: Record<string, Record<string, number>> = {};

for (const result of this.results) {

if (!summary[result.category]) {

summary[result.category] = {

total: 0,

passed: 0,

failed: 0,

critical: 0,

high: 0,

medium: 0,

low: 0,

info: 0

};

}

summary[result.category].total += 1;

summary[result.category][result.success ? 'passed' : 'failed'] += 1;

summary[result.category][result.severity] += 1;

}

return summary;

}

}

/\*\*

\* Sample SQL Injection test

\*/

export const sqlInjectionTest = {

id: 'sec-test-001',

name: 'SQL Injection Detection',

category: TestCategory.INJECTION,

description: 'Tests endpoints for SQL injection vulnerabilities',

severity: 'critical',

remediationSteps: [

'Use parameterized queries or prepared statements',

'Implement proper input validation',

'Use an ORM with protection against SQL injection',

'Apply the principle of least privilege for database users'

],

execute: async ({ baseUrl, http }) => {

const injectionStrings = [

"' OR '1'='1",

"'; DROP TABLE users; --",

"' UNION SELECT username, password FROM users; --",

"admin' --"

];

const endpoints = [

'/api/login',

'/api/search',

'/api/users'

];

const vulnerableEndpoints = [];

for (const endpoint of endpoints) {

for (const injection of injectionStrings) {

try {

// Test with GET parameter

const getResponse = await http.get(`${baseUrl}${endpoint}?q=${encodeURIComponent(injection)}`);

// Look for suspicious patterns in the response

if (getResponse.data &&

(getResponse.data.includes('SQL syntax') ||

getResponse.data.includes('database error'))) {

vulnerableEndpoints.push({

endpoint,

method: 'GET',

injection

});

}

} catch (error) {

// Some errors might also indicate vulnerability

if (error.response && error.response.data &&

(error.response.data.includes('SQL syntax') ||

error.response.data.includes('database error'))) {

vulnerableEndpoints.push({

endpoint,

method: 'GET',

injection,

error: error.message

});

}

}

try {

// Test with POST body

const postResponse = await http.post(`${baseUrl}${endpoint}`, {

q: injection

});

// Look for suspicious patterns in the response

if (postResponse.data &&

(postResponse.data.includes('SQL syntax') ||

postResponse.data.includes('database error'))) {

vulnerableEndpoints.push({

endpoint,

method: 'POST',

injection

});

}

} catch (error) {

// Some errors might also indicate vulnerability

if (error.response && error.response.data &&

(error.response.data.includes('SQL syntax') ||

error.response.data.includes('database error'))) {

vulnerableEndpoints.push({

endpoint,

method: 'POST',

injection,

error: error.message

});

}

}

}

}

return {

success: vulnerableEndpoints.length === 0,

message: vulnerableEndpoints.length === 0

? 'No SQL injection vulnerabilities detected'

: `Detected ${vulnerableEndpoints.length} potential SQL injection vulnerabilities`,

details: {

vulnerableEndpoints

}

};

}

};

Phase 3.2: Regular Security Scanning System (Week 15)

// server/security/testing/SecurityScanScheduler.ts

import { CronJob } from 'cron';

import { PenetrationTestingFramework, TestCategory } from './PenetrationTestingFramework';

import \* as fs from 'fs';

import \* as path from 'path';

import { db } from '../../../db';

import { pgTable, serial, text, timestamp, jsonb, boolean } from 'drizzle-orm/pg-core';

export const securityScans = pgTable('security\_scans', {

id: serial('id').primaryKey(),

scanType: text('scan\_type').notNull(),

startedAt: timestamp('started\_at').notNull(),

completedAt: timestamp('completed\_at'),

status: text('status').notNull(), // 'pending', 'running', 'completed', 'failed'

summary: jsonb('summary'),

results: jsonb('results'),

errorMessage: text('error\_message'),

createdBy: text('created\_by')

});

export interface ScanScheduleConfig {

enabled: boolean;

cronExpression: string;

categories: TestCategory[];

reportPath?: string;

notifyOnFailure: boolean;

notifyEmail?: string;

}

export class SecurityScanScheduler {

private config: ScanScheduleConfig;

private cronJob: CronJob | null = null;

private testingFramework: PenetrationTestingFramework;

constructor(config: Partial<ScanScheduleConfig> = {}) {

this.config = {

enabled: true,

cronExpression: '0 0 \* \* \*', // Daily at midnight

categories: Object.values(TestCategory),

notifyOnFailure: true,

...config

};

this.testingFramework = new PenetrationTestingFramework();

if (this.config.enabled) {

this.schedule();

}

}

/\*\*

\* Schedule regular security scans

\*/

private schedule() {

this.cronJob = new CronJob(

this.config.cronExpression,

() => this.runScheduledScan(),

null,

true

);

console.log(`Security scans scheduled with cron expression: ${this.config.cronExpression}`);

}

/\*\*

\* Run a scheduled security scan

\*/

private async runScheduledScan() {

console.log('Running scheduled security scan');

try {

// Create a scan record

const [scanRecord] = await db

.insert(securityScans)

.values({

scanType: 'scheduled',

startedAt: new Date(),

status: 'running',

createdBy: 'system'

})

.returning();

let failedCategories = 0;

const categoryResults: Record<string, any> = {};

// Run tests for each category

for (const category of this.config.categories) {

try {

console.log(`Running tests for category: ${category}`);

const results = await this.testingFramework.runTests(category);

const failedTests = results.filter(r => !r.success);

categoryResults[category] = {

total: results.length,

passed: results.length - failedTests.length,

failed: failedTests.length,

results: results

};

if (failedTests.length > 0) {

failedCategories++;

}

} catch (error) {

console.error(`Error running tests for category ${category}:`, error);

failedCategories++;

categoryResults[category] = {

error: error.message,

stack: error.stack

};

}

}

// Generate summary

const summary = {

totalCategories: this.config.categories.length,

failedCategories,

passedCategories: this.config.categories.length - failedCategories,

timestamp: Date.now()

};

// Generate report

if (this.config.reportPath) {

const reportFilePath = path.join(

this.config.reportPath,

`security-scan-${new Date().toISOString().replace(/:/g, '-')}.json`

);

fs.writeFileSync(

reportFilePath,

JSON.stringify({ summary, results: categoryResults }, null, 2)

);

console.log(`Security scan report saved to ${reportFilePath}`);

}

// Update scan record

await db

.update(securityScans)

.set({

completedAt: new Date(),

status: 'completed',

summary,

results: categoryResults

})

.where({ id: scanRecord.id });

// Notify on failure if configured

if (failedCategories > 0 && this.config.notifyOnFailure && this.config.notifyEmail) {

await this.sendNotification(

this.config.notifyEmail,

'Security Scan Failed',

`${failedCategories} out of ${this.config.categories.length} security test categories failed. Please check the security dashboard for details.`

);

}

} catch (error) {

console.error('Error running scheduled security scan:', error);

// Create a failed scan record

await db

.insert(securityScans)

.values({

scanType: 'scheduled',

startedAt: new Date(),

completedAt: new Date(),

status: 'failed',

errorMessage: error.message,

createdBy: 'system'

});

}

}

/\*\*

\* Run a manual security scan

\*/

async runManualScan(userId: string, categories: TestCategory[] = []): Promise<any> {

console.log(`Running manual security scan initiated by user ${userId}`);

try {

// Create a scan record

const [scanRecord] = await db

.insert(securityScans)

.values({

scanType: 'manual',

startedAt: new Date(),

status: 'running',

createdBy: userId

})

.returning();

// Use provided categories or default to all

const categoriesToScan = categories.length > 0

? categories

: this.config.categories;

let failedCategories = 0;

const categoryResults: Record<string, any> = {};

// Run tests for each category

for (const category of categoriesToScan) {

try {

console.log(`Running tests for category: ${category}`);

const results = await this.testingFramework.runTests(category);

const failedTests = results.filter(r => !r.success);

categoryResults[category] = {

total: results.length,

passed: results.length - failedTests.length,

failed: failedTests.length,

results: results

};

if (failedTests.length > 0) {

failedCategories++;

}

} catch (error) {

console.error(`Error running tests for category ${category}:`, error);

failedCategories++;

categoryResults[category] = {

error: error.message,

stack: error.stack

};

}

}

// Generate summary

const summary = {

totalCategories: categoriesToScan.length,

failedCategories,

passedCategories: categoriesToScan.length - failedCategories,

timestamp: Date.now()

};

// Update scan record

await db

.update(securityScans)

.set({

completedAt: new Date(),

status: 'completed',

summary,

results: categoryResults

})

.where({ id: scanRecord.id });

return {

scanId: scanRecord.id,

summary,

results: categoryResults

};

} catch (error) {

console.error('Error running manual security scan:', error);

// Create a failed scan record

const [scanRecord] = await db

.insert(securityScans)

.values({

scanType: 'manual',

startedAt: new Date(),

completedAt: new Date(),

status: 'failed',

errorMessage: error.message,

createdBy: userId

})

.returning();

throw error;

}

}

/\*\*

\* Send notification email

\*/

private async sendNotification(email: string, subject: string, message: string): Promise<void> {

// In a real implementation, this would use a proper email service

console.log(`[NOTIFICATION] To: ${email}, Subject: ${subject}, Message: ${message}`);

}

/\*\*

\* Get the security scan history

\*/

async getScanHistory(limit: number = 10, offset: number = 0): Promise<any[]> {

const history = await db

.select()

.from(securityScans)

.orderBy({ started\_at: 'desc' })

.limit(limit)

.offset(offset);

return history;

}

/\*\*

\* Get a security scan by ID

\*/

async getScanById(scanId: number): Promise<any> {

const [scan] = await db

.select()

.from(securityScans)

.where({ id: scanId });

return scan;

}

}

export const securityScanScheduler = new SecurityScanScheduler();

9. Admin UI Components

Phase 3.3: Security Dashboard Component (Week 16)

// client/src/components/admin/security/SecurityDashboard.tsx

import React, { useEffect, useState } from 'react';

import { Card, CardContent, CardDescription, CardFooter, CardHeader, CardTitle } from '@/components/ui/card';

import { Tabs, TabsContent, TabsList, TabsTrigger } from '@/components/ui/tabs';

import { Alert, AlertDescription, AlertTitle } from '@/components/ui/alert';

import { Badge } from '@/components/ui/badge';

import { Button } from '@/components/ui/button';

import { Progress } from '@/components/ui/progress';

import { useToast } from '@/hooks/use-toast';

import { useMutation, useQuery } from '@tanstack/react-query';

import { Shield, AlertTriangle, CheckCircle, RefreshCw, Lock, Activity, List, BarChart } from 'lucide-react';

import { apiRequest, queryClient } from '@/lib/queryClient';

import { Skeleton } from '@/components/ui/skeleton';

import { ThreatSummary } from './ThreatSummary';

import { SecurityScoreCard } from './SecurityScoreCard';

import { ActiveThreats } from './ActiveThreats';

import { SecurityEvents } from './SecurityEvents';

import { SecurityLevelSelector } from './SecurityLevelSelector';

export function SecurityDashboard() {

const { toast } = useToast();

const [activeTab, setActiveTab] = useState<string>('overview');

// Fetch security status

const { data: securityStatus, isLoading: isLoadingStatus } = useQuery({

queryKey: ['/api/security/status'],

refetchInterval: 30000, // Refresh every 30 seconds

});

// Fetch security score

const { data: securityScore, isLoading: isLoadingScore } = useQuery({

queryKey: ['/api/security/threat/security-score'],

refetchInterval: 60000, // Refresh every minute

});

// Fetch security statistics

const { data: securityStats, isLoading: isLoadingStats } = useQuery({

queryKey: ['/api/security/threat/statistics'],

refetchInterval: 30000, // Refresh every 30 seconds

});

// Mutation to change security level

const changeSecurityLevelMutation = useMutation({

mutationFn: async ({ level }: { level: string }) => {

const res = await apiRequest('POST', '/api/security/level', { level });

return await res.json();

},

onSuccess: () => {

toast({

title: 'Security level updated',

description: 'The security level has been updated successfully.',

});

// Refetch security status

queryClient.invalidateQueries({ queryKey: ['/api/security/status'] });

},

onError: (error: Error) => {

toast({

title: 'Failed to update security level',

description: error.message,

variant: 'destructive',

});

},

});

// Mutation to run security scan

const runSecurityScanMutation = useMutation({

mutationFn: async () => {

const res = await apiRequest('POST', '/api/security/scan');

return await res.json();

},

onSuccess: () => {

toast({

title: 'Security scan initiated',

description: 'The security scan has been started. Results will be available soon.',

});

},

onError: (error: Error) => {

toast({

title: 'Failed to start security scan',

description: error.message,

variant: 'destructive',

});

},

});

// Handler for changing security level

const handleChangeSecurityLevel = (level: string) => {

changeSecurityLevelMutation.mutate({ level });

};

// Handler for running security scan

const handleRunSecurityScan = () => {

runSecurityScanMutation.mutate();

};

return (

<div className="space-y-6">

<div className="flex justify-between items-center">

<h2 className="text-3xl font-bold tracking-tight">Security Dashboard</h2>

<div className="flex space-x-4">

<SecurityLevelSelector

currentLevel={securityStatus?.level || 'STANDARD'}

onChange={handleChangeSecurityLevel}

isLoading={changeSecurityLevelMutation.isPending}

/>

<Button

variant="outline"

onClick={handleRunSecurityScan}

disabled={runSecurityScanMutation.isPending}

>

{runSecurityScanMutation.isPending ? (

<>

<RefreshCw className="mr-2 h-4 w-4 animate-spin" />

Running Scan...

</>

) : (

<>

<Shield className="mr-2 h-4 w-4" />

Run Security Scan

</>

)}

</Button>

</div>

</div>

{securityStatus?.activeThreatCount > 0 && (

<Alert variant="destructive">

<AlertTriangle className="h-4 w-4" />

<AlertTitle>Active Security Threats Detected</AlertTitle>

<AlertDescription>

There are {securityStatus.activeThreatCount} active security threats that require your attention.

<Button variant="link" className="p-0 h-auto font-semibold" onClick={() => setActiveTab('threats')}>

View Threats

</Button>

</AlertDescription>

</Alert>

)}

<div className="grid gap-4 md:grid-cols-2 lg:grid-cols-4">

<Card>

<CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">

<CardTitle className="text-sm font-medium">Security Score</CardTitle>

<Shield className="h-4 w-4 text-muted-foreground" />

</CardHeader>

<CardContent>

{isLoadingScore ? (

<Skeleton className="h-8 w-full mt-2" />

) : (

<>

<div className="text-2xl font-bold">

{securityScore?.securityScore?.overall || 0}/100

</div>

<Progress

value={securityScore?.securityScore?.overall || 0}

className="h-2 mt-2"

indicatorColor={

(securityScore?.securityScore?.overall || 0) >= 80 ? 'bg-green-500' :

(securityScore?.securityScore?.overall || 0) >= 60 ? 'bg-yellow-500' :

'bg-red-500'

}

/>

</>

)}

</CardContent>

</Card>

<Card>

<CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">

<CardTitle className="text-sm font-medium">Active Threats</CardTitle>

<AlertTriangle className="h-4 w-4 text-muted-foreground" />

</CardHeader>

<CardContent>

{isLoadingStatus ? (

<Skeleton className="h-8 w-full mt-2" />

) : (

<>

<div className="text-2xl font-bold">

{securityStatus?.activeThreatCount || 0}

</div>

<div className="text-xs text-muted-foreground mt-2">

{securityStatus?.recentThreatCount || 0} threats in the last 24 hours

</div>

</>

)}

</CardContent>

</Card>

<Card>

<CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">

<CardTitle className="text-sm font-medium">Security Level</CardTitle>

<Lock className="h-4 w-4 text-muted-foreground" />

</CardHeader>

<CardContent>

{isLoadingStatus ? (

<Skeleton className="h-8 w-full mt-2" />

) : (

<>

<div className="text-2xl font-bold">

{securityStatus?.level || 'STANDARD'}

</div>

<div className="text-xs text-muted-foreground mt-2">

{securityStatus?.activeFeatures || 0} active security features

</div>

</>

)}

</CardContent>

</Card>

<Card>

<CardHeader className="flex flex-row items-center justify-between space-y-0 pb-2">

<CardTitle className="text-sm font-medium">Blocked Attempts</CardTitle>

<Activity className="h-4 w-4 text-muted-foreground" />

</CardHeader>

<CardContent>

{isLoadingStats ? (

<Skeleton className="h-8 w-full mt-2" />

) : (

<>

<div className="text-2xl font-bold">

{securityStats?.statistics?.blockedRequests || 0}

</div>

<div className="text-xs text-muted-foreground mt-2">

{securityStats?.statistics?.blockedIps?.length || 0} IPs currently blocked

</div>

</>

)}

</CardContent>

</Card>

</div>

<Tabs value={activeTab} onValueChange={setActiveTab}>

<TabsList className="grid w-full grid-cols-4">

<TabsTrigger value="overview">Overview</TabsTrigger>

<TabsTrigger value="threats">Threats</TabsTrigger>

<TabsTrigger value="events">Events</TabsTrigger>

<TabsTrigger value="recommendations">Recommendations</TabsTrigger>

</TabsList>

<TabsContent value="overview" className="space-y-4">

<SecurityScoreCard

score={securityScore?.securityScore}

isLoading={isLoadingScore}

/>

<ThreatSummary

statistics={securityStats?.statistics}

isLoading={isLoadingStats}

/>

</TabsContent>

<TabsContent value="threats" className="space-y-4">

<ActiveThreats />

</TabsContent>

<TabsContent value="events" className="space-y-4">

<SecurityEvents />

</TabsContent>

<TabsContent value="recommendations" className="space-y-4">

{isLoadingScore ? (

<div className="space-y-3">

<Skeleton className="h-8 w-full" />

<Skeleton className="h-20 w-full" />

<Skeleton className="h-20 w-full" />

<Skeleton className="h-20 w-full" />

</div>

) : (

<Card>

<CardHeader>

<CardTitle>Security Recommendations</CardTitle>

<CardDescription>

Follow these recommendations to improve your security score

</CardDescription>

</CardHeader>

<CardContent className="space-y-4">

{securityScore?.securityScore?.recommendations?.map((recommendation: string, index: number) => (

<div key={index} className="flex items-start space-x-2">

<div className="h-6 w-6 flex items-center justify-center rounded-full bg-primary/20 text-primary">

{index + 1}

</div>

<div>{recommendation}</div>

</div>

))}

{(!securityScore?.securityScore?.recommendations ||

securityScore.securityScore.recommendations.length === 0) && (

<div className="text-center py-4">

<CheckCircle className="mx-auto h-12 w-12 text-green-500 mb-2" />

<h3 className="text-xl font-semibold">No Recommendations</h3>

<p className="text-muted-foreground mt-1">

Great job! Your security configuration is looking good.

</p>

</div>

)}

</CardContent>

{securityScore?.securityScore?.recommendations?.length > 0 && (

<CardFooter>

<Button className="w-full">Apply Recommendations</Button>

</CardFooter>

)}

</Card>

)}

</TabsContent>

</Tabs>

</div>

);

}

Phase 3.4: Security Configuration Interface (Week 17)

// client/src/components/admin/security/SecurityConfigInterface.tsx

import React, { useState } from 'react';

import { Card, CardContent, CardDescription, CardFooter, CardHeader, CardTitle } from '@/components/ui/card';

import { Tabs, TabsContent, TabsList, TabsTrigger } from '@/components/ui/tabs';

import { Button } from '@/components/ui/button';

import { Switch } from '@/components/ui/switch';

import { Slider } from '@/components/ui/slider';

import { Label } from '@/components/ui/label';

import { Input } from '@/components/ui/input';

import { Separator } from '@/components/ui/separator';

import { ScrollArea } from '@/components/ui/scroll-area';

import { useToast } from '@/hooks/use-toast';

import { useMutation, useQuery } from '@tanstack/react-query';

import { Shield, Database, UserCog, Network, Lock, Activity, AlertTriangle, Eye } from 'lucide-react';

import { apiRequest, queryClient } from '@/lib/queryClient';

import { Skeleton } from '@/components/ui/skeleton';

export function SecurityConfigInterface() {

const { toast } = useToast();

const [activeTab, setActiveTab] = useState<string>('general');

// Fetch security configuration

const { data: securityConfig, isLoading } = useQuery({

queryKey: ['/api/security/config'],

});

// Mutation to update security configuration

const updateConfigMutation = useMutation({

mutationFn: async (config: any) => {

const res = await apiRequest('POST', '/api/security/config', config);

return await res.json();

},

onSuccess: () => {

toast({

title: 'Configuration updated',

description: 'Security configuration has been updated successfully.',

});

// Refetch configuration

queryClient.invalidateQueries({ queryKey: ['/api/security/config'] });

},

onError: (error: Error) => {

toast({

title: 'Failed to update configuration',

description: error.message,

variant: 'destructive',

});

},

});

// Handler for updating config

const handleUpdateConfig = (section: string, key: string, value: any) => {

if (!securityConfig) return;

const updatedConfig = {

...securityConfig,

[section]: {

...securityConfig[section],

[key]: value

}

};

updateConfigMutation.mutate(updatedConfig);

};

// Handler for bulk updating config section

const handleUpdateSection = (section: string, sectionConfig: any) => {

if (!securityConfig) return;

const updatedConfig = {

...securityConfig,

[section]: sectionConfig

};

updateConfigMutation.mutate(updatedConfig);

};

if (isLoading) {

return (

<div className="space-y-4">

<Skeleton className="h-8 w-64" />

<Skeleton className="h-8 w-full" />

<div className="grid gap-4 md:grid-cols-2">

<Skeleton className="h-48 w-full" />

<Skeleton className="h-48 w-full" />

</div>

</div>

);

}

return (

<div className="space-y-6">

<div className="flex justify-between items-center">

<h2 className="text-3xl font-bold tracking-tight">Security Configuration</h2>

<Button

variant="default"

onClick={() => {

// Reset configuration to defaults

updateConfigMutation.mutate({

...securityConfig,

\_action: 'reset\_defaults'

});

}}

disabled={updateConfigMutation.isPending}

>

Reset to Defaults

</Button>

</div>

<Tabs value={activeTab} onValueChange={setActiveTab}>

<TabsList className="grid w-full grid-cols-4">

<TabsTrigger value="general">General</TabsTrigger>

<TabsTrigger value="authentication">Authentication</TabsTrigger>

<TabsTrigger value="data">Data Protection</TabsTrigger>

<TabsTrigger value="advanced">Advanced</TabsTrigger>

</TabsList>

<TabsContent value="general" className="space-y-4">

<Card>

<CardHeader>

<CardTitle>Security Level</CardTitle>

<CardDescription>

Configure the overall security level for the application

</CardDescription>

</CardHeader>

<CardContent className="space-y-4">

<div className="grid gap-4">

<div className="space-y-2">

<Label htmlFor="security-level">Security Level</Label>

<select

id="security-level"

className="w-full p-2 border rounded-md"

value={securityConfig.general?.level || 'STANDARD'}

onChange={(e) => handleUpdateConfig('general', 'level', e.target.value)}

>

<option value="BASIC">Basic - Minimal Security</option>

<option value="STANDARD">Standard - Balanced Security</option>

<option value="ENHANCED">Enhanced - Higher Security</option>

<option value="HIGH">High - Strong Security</option>

<option value="MAXIMUM">Maximum - Paranoid Security</option>

</select>

</div>

<div className="space-y-2">

<Label>Auto-adjust security based on threats</Label>

<div className="flex items-center space-x-2">

<Switch

checked={securityConfig.general?.autoAdjust || false}

onCheckedChange={(checked) =>

handleUpdateConfig('general', 'autoAdjust', checked)

}

/>

<Label>Enable automatic security level adjustment</Label>

</div>

</div>

</div>

</CardContent>

</Card>

<Card>

<CardHeader>

<CardTitle>Logging & Monitoring</CardTitle>

<CardDescription>

Configure security logging and monitoring settings

</CardDescription>

</CardHeader>

<CardContent className="space-y-4">

<div className="grid gap-4">

<div className="space-y-2">

<Label>Security Event Logging</Label>

<div className="flex items-center space-x-2">

<Switch

checked={securityConfig.logging?.enabled || false}

onCheckedChange={(checked) =>

handleUpdateConfig('logging', 'enabled', checked)

}

/>

<Label>Enable security event logging</Label>

</div>

</div>

<div className="space-y-2">

<Label htmlFor="log-level">Log Level</Label>

<select

id="log-level"

className="w-full p-2 border rounded-md"

value={securityConfig.logging?.level || 'INFO'}

onChange={(e) => handleUpdateConfig('logging', 'level', e.target.value)}

>

<option value="DEBUG">Debug - All events</option>

<option value="INFO">Info - Standard events</option>

<option value="WARN">Warning - Issues and warnings</option>

<option value="ERROR">Error - Errors only</option>

</select>

</div>

<div className="space-y-2">

<Label>Real-time Threat Monitoring</Label>

<div className="flex items-center space-x-2">

<Switch

checked={securityConfig.monitoring?.realTime || false}

onCheckedChange={(checked) =>

handleUpdateConfig('monitoring', 'realTime', checked)

}

/>

<Label>Enable real-time threat monitoring</Label>

</div>

</div>

<div className="space-y-2">

<Label htmlFor="retention-days">Log Retention (Days)</Label>

<Input

id="retention-days"

type="number"

value={securityConfig.logging?.retentionDays || 30}

onChange={(e) =>

handleUpdateConfig('logging', 'retentionDays', parseInt(e.target.value))

}

min={1}

max={365}

/>

</div>

</div>

</CardContent>

</Card>

</TabsContent>

<TabsContent value="authentication" className="space-y-4">

<Card>

<CardHeader>

<CardTitle>Authentication Security</CardTitle>

<CardDescription>

Configure authentication security settings

</CardDescription>

</CardHeader>

<CardContent className="space-y-4">

<div className="grid gap-4">

<div className="space-y-2">

<Label>Multi-Factor Authentication</Label>

<div className="flex items-center space-x-2">

<Switch

checked={securityConfig.authentication?.mfaEnabled || false}

onCheckedChange={(checked) =>

handleUpdateConfig('authentication', 'mfaEnabled', checked)

}

/>

<Label>Require MFA for all admin users</Label>

</div>

</div>

<div className="space-y-2">

<Label>Brute Force Protection</Label>

<div className="flex items-center space-x-2">

<Switch

checked={securityConfig.authentication?.bruteForceProtection || false}

onCheckedChange={(checked) =>

handleUpdateConfig('authentication', 'bruteForceProtection', checked)

}

/>

<Label>Enable brute force protection</Label>

</div>

</div>

<div className="space-y-2">

<Label htmlFor="max-attempts">Maximum Login Attempts</Label>

<Input

id="max-attempts"

type="number"

value={securityConfig.authentication?.maxLoginAttempts || 5}

onChange={(e) =>

handleUpdateConfig('authentication', 'maxLoginAttempts', parseInt(e.target.value))

}

min={1}

max={10}

/>

</div>

<div className="space-y-2">

<Label htmlFor="lockout-time">Account Lockout Time (Minutes)</Label>

<Input

id="lockout-time"

type="number"

value={securityConfig.authentication?.lockoutTimeMinutes || 30}

onChange={(e) =>

handleUpdateConfig('authentication', 'lockoutTimeMinutes', parseInt(e.target.value))

}

min={1}

max={1440}

/>

</div>

<div className="space-y-2">

<Label>Risk-Based Authentication</Label>

<div className="flex items-center space-x-2">

<Switch

checked={securityConfig.authentication?.riskBasedAuth || false}

onCheckedChange={(checked) =>

handleUpdateConfig('authentication', 'riskBasedAuth', checked)

}

/>

<Label>Enable risk-based authentication challenges</Label>

</div>

</div>

<div className="space-y-2">

<Label>Session Security</Label>

<div className="flex items-center space-x-2">

<Switch

checked={securityConfig.authentication?.secureSession || false}

onCheckedChange={(checked) =>

handleUpdateConfig('authentication', 'secureSession', checked)

}

/>

<Label>Enforce secure session settings</Label>

</div>

</div>

</div>

</CardContent>

</Card>

</TabsContent>

<TabsContent value="data" className="space-y-4">

<Card>

<CardHeader>

<CardTitle>Data Protection</CardTitle>

<CardDescription>

Configure data protection and encryption settings

</CardDescription>

</CardHeader>

<CardContent className="space-y-4">

<div className="grid gap-4">

<div className="space-y-2">

<Label>Data Encryption</Label>

<div className="flex items-center space-x-2">

<Switch

checked={securityConfig.dataProtection?.encryption || false}

onCheckedChange={(checked) =>

handleUpdateConfig('dataProtection', 'encryption', checked)

}

/>

<Label>Enable data encryption</Label>

</div>

</div>

<div className="space-y-2">

<Label htmlFor="encryption-algorithm">Encryption Algorithm</Label>

<select

id="encryption-algorithm"

className="w-full p-2 border rounded-md"

value={securityConfig.dataProtection?.algorithm || 'AES-256-GCM'}

onChange={(e) => handleUpdateConfig('dataProtection', 'algorithm', e.target.value)}

disabled={!securityConfig.dataProtection?.encryption}

>

<option value="AES-256-GCM">AES-256-GCM</option>

<option value="ChaCha20-Poly1305">ChaCha20-Poly1305</option>

<option value="KYBER-768">KYBER-768 (Quantum Resistant)</option>

</select>

</div>

<div className="space-y-2">

<Label>Data Anonymization</Label>

<div className="flex items-center space-x-2">

<Switch

checked={securityConfig.dataProtection?.anonymization || false}

onCheckedChange={(checked) =>

handleUpdateConfig('dataProtection', 'anonymization', checked)

}

/>

<Label>Enable data anonymization for PII</Label>

</div>

</div>

<div className="space-y-2">

<Label>Key Rotation</Label>

<div className="flex items-center space-x-2">

<Switch

checked={securityConfig.dataProtection?.keyRotation || false}

onCheckedChange={(checked) =>

handleUpdateConfig('dataProtection', 'keyRotation', checked)

}

/>

<Label>Enable automatic encryption key rotation</Label>

</div>

</div>

<div className="space-y-2">

<Label htmlFor="key-rotation-days">Key Rotation Interval (Days)</Label>

<Input

id="key-rotation-days"

type="number"

value={securityConfig.dataProtection?.keyRotationDays || 90}

onChange={(e) =>

handleUpdateConfig('dataProtection', 'keyRotationDays', parseInt(e.target.value))

}

min={1}

max={365}

disabled={!securityConfig.dataProtection?.keyRotation}

/>

</div>

</div>

</CardContent>

</Card>

</TabsContent>

<TabsContent value="advanced" className="space-y-4">

<Card>

<CardHeader>

<CardTitle>Runtime Security</CardTitle>

<CardDescription>

Configure advanced runtime security settings

</CardDescription>

</CardHeader>

<CardContent className="space-y-4">

<div className="grid gap-4">

<div className="space-y-2">

<Label>Runtime Application Self-Protection (RASP)</Label>

<div className="flex items-center space-x-2">

<Switch

checked={securityConfig.advanced?.rasp || false}

onCheckedChange={(checked) =>

handleUpdateConfig('advanced', 'rasp', checked)

}

/>

<Label>Enable Runtime Application Self-Protection</Label>

</div>

</div>

<div className="space-y-2">

<Label>API Security</Label>

<div className="flex items-center space-x-2">

<Switch

checked={securityConfig.advanced?.apiSecurity || false}

onCheckedChange={(checked) =>

handleUpdateConfig('advanced', 'apiSecurity', checked)

}

/>

<Label>Enable API security</Label>

</div>

</div>

<div className="space-y-2">

<Label>Rate Limiting</Label>

<div className="flex items-center space-x-2">

<Switch

checked={securityConfig.advanced?.rateLimiting || false}

onCheckedChange={(checked) =>

handleUpdateConfig('advanced', 'rateLimiting', checked)

}

/>

<Label>Enable rate limiting</Label>

</div>

</div>

<div className="space-y-2">

<Label htmlFor="rate-limit">Rate Limit (requests per minute)</Label>

<Input

id="rate-limit"

type="number"

value={securityConfig.advanced?.rateLimit || 60}

onChange={(e) =>

handleUpdateConfig('advanced', 'rateLimit', parseInt(e.target.value))

}

min={10}

max={1000}

disabled={!securityConfig.advanced?.rateLimiting}

/>

</div>

<div className="space-y-2">

<Label>ML-based Anomaly Detection</Label>

<div className="flex items-center space-x-2">

<Switch

checked={securityConfig.advanced?.mlDetection || false}

onCheckedChange={(checked) =>

handleUpdateConfig('advanced', 'mlDetection', checked)

}

/>

<Label>Enable machine learning anomaly detection</Label>

</div>

</div>

<div className="space-y-2">

<Label>Zero-Knowledge Proofs</Label>

<div className="flex items-center space-x-2">

<Switch

checked={securityConfig.advanced?.zkp || false}

onCheckedChange={(checked) =>

handleUpdateConfig('advanced', 'zkp', checked)

}

/>

<Label>Enable zero-knowledge proofs for sensitive operations</Label>

</div>

</div>

</div>

</CardContent>

</Card>

</TabsContent>

</Tabs>

</div>

);

}

10. Comprehensive Documentation

Phase 3.5: API Security Documentation (Week 18)

# Security API Documentation

## Overview

This document provides comprehensive documentation for the security API endpoints available in the Admin Portal. These endpoints allow administrators to configure security settings, monitor security events, and respond to security threats.

## Authentication and Authorization

All security API endpoints require authentication using the standard authentication mechanism. In addition, most endpoints require administrative privileges.

### Security Levels

The following security levels are available:

- `BASIC`: Minimal security features

- `STANDARD`: Standard security features suitable for most applications

- `ENHANCED`: Enhanced security features for sensitive applications

- `HIGH`: High security features for high-risk applications

- `MAXIMUM`: Maximum security features for extremely high-risk applications

## Base URL

All API endpoints are relative to: `/api/security/`

## General Security Endpoints

### Get Security Status

\*\*Endpoint:\*\* `GET /status`

\*\*Description:\*\* Returns the current security status including active threats, security level, and active features.

\*\*Response:\*\*

```json

{

"level": "ENHANCED",

"activeThreatCount": 3,

"recentThreatCount": 5,

"activeFeatures": 12,

"features": {

"rateLimiting": true,

"apiSecurity": true,

"mfa": true,

"csrf": true,

"encryption": true,

"anomalyDetection": false,

"zkp": false

}

}

Update Security Level

**Endpoint:** POST /level

**Description:** Updates the current security level.

**Request Body:**

{

"level": "HIGH"

}

**Response:**

{

"success": true,

"level": "HIGH",

"activeFeatures": 15

}

Get Security Configuration

**Endpoint:** GET /config

**Description:** Returns the current security configuration.

**Response:**

{

"general": {

"level": "ENHANCED",

"autoAdjust": true

},

"logging": {

"enabled": true,

"level": "INFO",

"retentionDays": 30

},

"authentication": {

"mfaEnabled": true,

"bruteForceProtection": true,

"maxLoginAttempts": 5,

"lockoutTimeMinutes": 30,

"riskBasedAuth": false,

"secureSession": true

},

"dataProtection": {

"encryption": true,

"algorithm": "AES-256-GCM",

"anonymization": false,

"keyRotation": true,

"keyRotationDays": 90

},

"advanced": {

"rasp": true,

"apiSecurity": true,

"rateLimiting": true,

"rateLimit": 60,

"mlDetection": false,

"zkp": false

}

}

Update Security Configuration

**Endpoint:** POST /config

**Description:** Updates the security configuration.

**Request Body:**

A JSON object with the configuration changes. You can update specific sections or the entire configuration.

**Response:**

{

"success": true,

"message": "Configuration updated successfully"

}

Run Security Scan

**Endpoint:** POST /scan

**Description:** Initiates a security scan.

**Request Body (optional):**

{

"categories": ["injection", "xss", "broken\_auth"]

}

**Response:**

{

"success": true,

"scanId": 123,

"message": "Security scan initiated"

}

Get Security Scan Results

**Endpoint:** GET /scan/:scanId

**Description:** Gets the results of a security scan.

**Response:**

{

"scanId": 123,

"startedAt": "2025-05-01T12:34:56Z",

"completedAt": "2025-05-01T12:35:30Z",

"status": "completed",

"summary": {

"totalCategories": 3,

"failedCategories": 1,

"passedCategories": 2

},

"results": {

"injection": {

"total": 5,

"passed": 5,

"failed": 0

},

"xss": {

"total": 3,

"passed": 1,

"failed": 2,

"results": [

{

"id": "xss-001",

"name": "Reflected XSS Test",

"success": false,

"message": "Reflected XSS vulnerability detected"

}

]

},

"broken\_auth": {

"total": 4,

"passed": 4,

"failed": 0

}

}

}

Threat Protection Endpoints

Get Detected Threats

**Endpoint:** GET /threat/detected

**Description:** Returns a list of detected threats with optional pagination.

**Query Parameters:**

* limit (optional): Maximum number of threats to return
* offset (optional): Offset for pagination

**Response:**

{

"success": true,

"threats": [

{

"id": "threat-123",

"timestamp": 1651396596000,

"threatType": "SQL\_INJECTION",

"severity": "HIGH",

"description": "SQL Injection attempt detected",

"sourceIp": "192.168.1.1",

"requestPath": "/api/users",

"resolved": false

}

],

"total": 10,

"active": 5

}

Get Active Threats

**Endpoint:** GET /threat/active

**Description:** Returns a list of active (unresolved) threats.

**Response:**

{

"success": true,

"threats": [

{

"id": "threat-123",

"timestamp": 1651396596000,

"threatType": "SQL\_INJECTION",

"severity": "HIGH",

"description": "SQL Injection attempt detected",

"sourceIp": "192.168.1.1",

"requestPath": "/api/users",

"resolved": false

}

],

"count": 5

}

Resolve a Threat

**Endpoint:** POST /threat/resolve/:threatId

**Description:** Marks a threat as resolved.

**Response:**

{

"success": true,

"message": "Threat threat-123 resolved successfully"

}

Get Threat Statistics

**Endpoint:** GET /threat/statistics

**Description:** Returns statistics about detected threats.

**Response:**

{

"success": true,

"statistics": {

"totalThreats": 50,

"activeThreats": 5,

"resolvedThreats": 45,

"threatsByType": {

"SQL\_INJECTION": 10,

"XSS": 15,

"CSRF": 5,

"API\_ABUSE": 20

},

"threatsBySeverity": {

"CRITICAL": 5,

"HIGH": 15,

"MEDIUM": 20,

"LOW": 10

},

"threatsByTimeOfDay": {

"00:00": 2,

"01:00": 1,

"02:00": 0,

"03:00": 3

},

"topRules": [

{

"id": "rule-001",

"count": 15,

"name": "SQL Injection Detection"

}

]

}

}

Get Security Score

**Endpoint:** GET /threat/security-score

**Description:** Returns the current security score and recommendations.

**Response:**

{

"success": true,

"securityScore": {

"overall": 75,

"components": {

"threatMitigation": 80,

"configurationSecurity": 70,

"userSecurity": 65,

"dataProtection": 85,

"monitoring": 70

},

"recommendations": [

"Enable multi-factor authentication for all admin users",

"Update security policies for sensitive data access"

]

}

}

Error Handling

All API endpoints return standard HTTP status codes:

* 200 OK: The request was successful
* 400 Bad Request: The request was invalid
* 401 Unauthorized: Authentication is required
* 403 Forbidden: The authenticated user does not have permission
* 404 Not Found: The requested resource was not found
* 500 Internal Server Error: An error occurred on the server

Error responses include a JSON body with error details:

{

"success": false,

"error": "Error message"

}

Rate Limiting

Security API endpoints are subject to rate limiting to prevent abuse. The default rate limit is 60 requests per minute per IP address. When the rate limit is exceeded, the API returns a 429 Too Many Requests status code.

Webhooks

Security events can trigger webhooks if configured. Webhook configuration is available through the Admin Portal interface.

Support and Feedback

For support or feedback, please contact the security team at [security@example.com](https://replit.com/@templictemplate/website-whales-76-20250430-1118AM-test).

### Phase 3.6: Admin Security Guide (Week 19)

```markdown

# Admin Portal Security Guide

## Introduction

This guide provides comprehensive information about the security features available in the Admin Portal. It explains how to configure security settings, respond to security threats, and maintain a secure application environment.

## Security Dashboard

The Security Dashboard provides a centralized view of your application's security status, active threats, and security recommendations.

### Key Dashboard Elements

- \*\*Security Score\*\*: A numerical score (0-100) representing your overall security posture.

- \*\*Active Threats\*\*: The number of active security threats requiring attention.

- \*\*Security Level\*\*: The current security level (BASIC, STANDARD, ENHANCED, HIGH, MAXIMUM).

- \*\*Blocked Attempts\*\*: The number of blocked malicious attempts.

### Dashboard Tabs

1. \*\*Overview\*\*: Provides a high-level overview of security status.

2. \*\*Threats\*\*: Lists all detected security threats with filtering and sorting options.

3. \*\*Events\*\*: Shows recent security events such as login attempts, configuration changes, and suspicious activities.

4. \*\*Recommendations\*\*: Provides actionable recommendations to improve your security posture.

## Security Configuration

### Security Levels

The Admin Portal supports five security levels:

1. \*\*BASIC\*\*: Minimal security features suitable for development environments.

2. \*\*STANDARD\*\*: Standard security features suitable for most applications.

3. \*\*ENHANCED\*\*: Enhanced security features for applications with sensitive data.

4. \*\*HIGH\*\*: High security features for high-risk applications.

5. \*\*MAXIMUM\*\*: Maximum security features with potential performance impact.

### Configuration Sections

#### General Security

- \*\*Security Level\*\*: Set the overall security level.

- \*\*Auto-adjust security\*\*: Automatically adjust security level based on detected threats.

- \*\*Logging & Monitoring\*\*: Configure security logging and real-time monitoring.

#### Authentication Security

- \*\*Multi-Factor Authentication\*\*: Require MFA for admin users.

- \*\*Brute Force Protection\*\*: Configure protection against login brute force attacks.

- \*\*Risk-Based Authentication\*\*: Enable challenges for suspicious login attempts.

- \*\*Session Security\*\*: Enforce secure session settings.

#### Data Protection

- \*\*Data Encryption\*\*: Enable encryption for sensitive data.

- \*\*Encryption Algorithm\*\*: Select the encryption algorithm.

- \*\*Data Anonymization\*\*: Enable anonymization for personally identifiable information.

- \*\*Key Rotation\*\*: Configure automatic encryption key rotation.

#### Advanced Security

- \*\*Runtime Application Self-Protection (RASP)\*\*: Enable runtime protection against attacks.

- \*\*API Security\*\*: Enable additional security for API endpoints.

- \*\*Rate Limiting\*\*: Configure request rate limiting.

- \*\*ML-based Anomaly Detection\*\*: Enable machine learning to detect anomalies.

- \*\*Zero-Knowledge Proofs\*\*: Enable zero-knowledge proofs for sensitive operations.

## Threat Management

### Types of Threats

The Admin Portal detects and responds to various types of security threats:

- \*\*SQL Injection\*\*: Attempts to inject malicious SQL code.

- \*\*Cross-Site Scripting (XSS)\*\*: Attempts to inject malicious scripts.

- \*\*Cross-Site Request Forgery (CSRF)\*\*: Unauthorized commands from trusted users.

- \*\*API Abuse\*\*: Excessive or malicious API usage.

- \*\*Authentication Attacks\*\*: Brute force or credential stuffing attacks.

- \*\*Suspicious Behavior\*\*: Unusual patterns that may indicate attacks.

### Threat Responses

When a threat is detected, the Admin Portal can take various actions:

1. \*\*Block IP\*\*: Automatically block the source IP address.

2. \*\*Rate Limit\*\*: Apply stricter rate limits to the source.

3. \*\*Notify Admins\*\*: Send notifications to administrators.

4. \*\*Challenge Request\*\*: Require additional verification.

5. \*\*Log Event\*\*: Record detailed information about the threat.

### Managing Threats

To manage threats through the Admin Portal:

1. Navigate to the Security Dashboard.

2. Click on the "Threats" tab.

3. Review the list of active threats.

4. Click on a threat to view details.

5. Take appropriate action:

- \*\*Resolve\*\*: Mark as resolved after investigation.

- \*\*Block IP\*\*: Block the source IP address.

- \*\*Execute Response\*\*: Run a recommended response action.

## Security Best Practices

### General Recommendations

1. \*\*Use Enhanced Security Level or Higher\*\*: For production environments with sensitive data.

2. \*\*Enable Multi-Factor Authentication\*\*: For all administrative users.

3. \*\*Regular Security Scans\*\*: Run security scans weekly or after significant changes.

4. \*\*Review Security Logs\*\*: Regularly review security logs for suspicious activity.

5. \*\*Apply Security Recommendations\*\*: Implement recommendations from the dashboard.

### Authentication Security

1. \*\*Strong Password Policy\*\*: Enforce strong passwords for all users.

2. \*\*MFA for All Admins\*\*: Require multi-factor authentication for all administrators.

3. \*\*Session Timeouts\*\*: Configure appropriate session timeouts.

4. \*\*Risk-Based Challenges\*\*: Enable additional challenges for suspicious logins.

### Data Protection

1. \*\*Encrypt Sensitive Data\*\*: Always encrypt personally identifiable information (PII).

2. \*\*Regular Key Rotation\*\*: Rotate encryption keys every 90 days or less.

3. \*\*Data Minimization\*\*: Collect and store only necessary data.

4. \*\*Anonymize Analytics Data\*\*: Use anonymization for analytics and reporting.

### API Security

1. \*\*Rate Limiting\*\*: Apply rate limits to all API endpoints.

2. \*\*Input Validation\*\*: Validate all API inputs.

3. \*\*Output Encoding\*\*: Encode all API outputs.

4. \*\*API Authentication\*\*: Require authentication for all sensitive API endpoints.

## Security Incident Response

### Incident Response Process

1. \*\*Detection\*\*: Security incidents are detected through the security monitoring system.

2. \*\*Classification\*\*: Incidents are classified by severity and type.

3. \*\*Containment\*\*: Take immediate actions to contain the incident.

4. \*\*Investigation\*\*: Investigate the root cause and impact.

5. \*\*Remediation\*\*: Implement measures to fix the vulnerability.

6. \*\*Recovery\*\*: Restore normal operations.

7. \*\*Lessons Learned\*\*: Document the incident and update security measures.

### Security Incident Severity Levels

- \*\*Critical\*\*: Severe impact, requires immediate response.

- \*\*High\*\*: Significant impact, requires prompt response.

- \*\*Medium\*\*: Moderate impact, requires timely response.

- \*\*Low\*\*: Minimal impact, can be addressed during normal operations.

### Reporting Security Incidents

1. Navigate to the Security Dashboard.

2. Click on the "Events" tab.

3. Click "Report Incident" button.

4. Fill in the incident details:

- Incident type

- Severity

- Description

- Affected components

- Actions taken

5. Click "Submit" to report the incident.

## Troubleshooting

### Common Issues

#### Security Features Not Working

1. Verify that the security level is appropriately set.

2. Check the security configuration settings.

3. Review the server logs for error messages.

4. Ensure all required services are running.

#### False Positives

1. Review the threat details carefully.

2. Consider adjusting security rule thresholds.

3. Add legitimate patterns to the whitelist.

4. Mark confirmed false positives as resolved.

#### Performance Impact

1. Consider reducing the security level during high-traffic periods.

2. Optimize security rule configurations.

3. Increase server resources if necessary.

4. Disable non-critical security features temporarily.

## Glossary

- \*\*CSRF (Cross-Site Request Forgery)\*\*: An attack where unauthorized commands are sent from a trusted user.

- \*\*MFA (Multi-Factor Authentication)\*\*: Authentication method that requires two or more verification factors.

- \*\*RASP (Runtime Application Self-Protection)\*\*: Security technology that operates within the application runtime environment.

- \*\*SQL Injection\*\*: An attack where malicious SQL code is inserted into database queries.

- \*\*XSS (Cross-Site Scripting)\*\*: An attack where malicious scripts are injected into trusted websites.

- \*\*Zero-Knowledge Proof\*\*: A method where one party can prove to another that a statement is true without revealing any additional information.

## Support and Resources

For additional support or questions about the security features in the Admin Portal, please contact the support team at support@example.com.

- \*\*Security Documentation\*\*: [link to documentation]

- \*\*Security Training\*\*: [link to training resources]

- \*\*Security Updates\*\*: [link to security update notifications]

Phase 3.7: Security Architecture Documentation (Week 20)

# Security Architecture Documentation

## Overview

This document provides a comprehensive overview of the security architecture implemented in the Admin Portal application. It describes the security components, their interactions, and the security principles followed in the design.

## Security Architecture Principles

The security architecture is designed based on the following principles:

1. \*\*Defense in Depth\*\*: Multiple layers of security controls are implemented.

2. \*\*Least Privilege\*\*: Users and processes have the minimum privileges necessary.

3. \*\*Secure by Default\*\*: Security features are enabled by default.

4. \*\*Fail Secure\*\*: Systems fail in a secure state.

5. \*\*Open Design\*\*: Security doesn't depend on secrecy of the design.

6. \*\*Economy of Mechanism\*\*: Security mechanisms are kept as simple as possible.

7. \*\*Complete Mediation\*\*: All access attempts are checked.

8. \*\*Separation of Duty\*\*: Critical actions require multiple approvers.

9. \*\*Zero Trust\*\*: Trust is never assumed, always verified.

## High-Level Architecture

┌───────────────────────────────────────────────────────────────┐  
│ │  
│ Client Applications │  
│ │  
└───────────────────────────┬───────────────────────────────────┘  
│  
▼  
┌───────────────────────────────────────────────────────────────┐  
│ Security Middleware │  
├───────────────┬───────────────┬───────────────┬───────────────┤  
│ Authentication│ Authorization │ Input Validation│ Rate Limiting │  
└───────────────┴───────────────┴───────────────┴───────────────┘  
│  
▼  
┌───────────────────────────────────────────────────────────────┐  
│ Security Fabric │  
├───────────────┬───────────────┬───────────────┬───────────────┤  
│ Security Level│ Event Handling│ Threat Detection│ Monitoring │  
└───────────────┴───────────────┴───────────────┴───────────────┘  
│  
▼  
┌───────────────────────────────────────────────────────────────┐  
│ Application Services │  
├───────────────┬───────────────┬───────────────┬───────────────┤  
│ API Services │ Business Logic│ Data Access │ External APIs │  
└───────────────┴───────────────┴───────────────┴───────────────┘  
│  
▼  
┌───────────────────────────────────────────────────────────────┐  
│ Data Protection │  
├───────────────┬───────────────┬───────────────┬───────────────┤  
│ Encryption │ Anonymization │ Access Control│ Audit Logging │  
└───────────────┴───────────────┴───────────────┴───────────────┘  
│  
▼  
┌───────────────────────────────────────────────────────────────┐  
│ Data Storage │  
├───────────────┬───────────────┬───────────────┬───────────────┤  
│ Database │ File Storage │ Cache Storage │ Backup Storage │  
└───────────────┴───────────────┴───────────────┴───────────────┘

## Security Components

### 1. Security Middleware

The Security Middleware intercepts all incoming requests and applies security checks before forwarding the requests to the application services.

#### Key Components:

- \*\*Authentication Middleware\*\*: Verifies user identity.

- \*\*Authorization Middleware\*\*: Checks user permissions.

- \*\*Input Validation Middleware\*\*: Validates and sanitizes input data.

- \*\*Rate Limiting Middleware\*\*: Prevents abuse through request rate limiting.

- \*\*CSRF Protection Middleware\*\*: Prevents cross-site request forgery attacks.

- \*\*IP Whitelisting Middleware\*\*: Restricts access based on IP addresses.

### 2. Security Fabric

The Security Fabric provides a centralized security configuration and event handling system.

#### Key Components:

- \*\*Security Level Manager\*\*: Manages the security level for the application.

- \*\*Event Handler\*\*: Processes security events and triggers actions.

- \*\*Threat Detection\*\*: Identifies and responds to security threats.

- \*\*Security Monitoring\*\*: Collects and analyzes security metrics.

### 3. Authentication and Authorization

#### Authentication Components:

- \*\*Credential Verification\*\*: Verifies username/password combinations.

- \*\*Multi-Factor Authentication\*\*: Implements TOTP-based second factor.

- \*\*Risk-Based Authentication\*\*: Applies additional challenges for suspicious logins.

- \*\*Brute Force Protection\*\*: Prevents multiple failed login attempts.

#### Authorization Components:

- \*\*Role-Based Access Control\*\*: Assigns permissions based on roles.

- \*\*Attribute-Based Access Control\*\*: Extends RBAC with contextual attributes.

- \*\*Permission Verification\*\*: Checks permissions for each action.

- \*\*Permission Inheritance\*\*: Implements role hierarchy.

### 4. Data Protection

#### Encryption Components:

- \*\*Data Encryption\*\*: Encrypts sensitive data at rest.

- \*\*Key Management\*\*: Manages encryption keys.

- \*\*Key Rotation\*\*: Automatically rotates encryption keys.

- \*\*Quantum-Resistant Encryption\*\*: Implements post-quantum cryptography.

#### Anonymization Components:

- \*\*Data Anonymization\*\*: Anonymizes personally identifiable information.

- \*\*Data Minimization\*\*: Reduces the amount of sensitive data stored.

- \*\*Data Retention\*\*: Manages data lifecycle.

### 5. Runtime Security

#### Runtime Protection Components:

- \*\*Runtime Application Self-Protection (RASP)\*\*: Protects the application at runtime.

- \*\*Request Analysis\*\*: Analyzes incoming requests for malicious patterns.

- \*\*Behavioral Monitoring\*\*: Identifies abnormal behavior.

- \*\*Response Modification\*\*: Modifies responses to block attacks.

#### API Security Components:

- \*\*API Validation\*\*: Validates API requests.

- \*\*API Rate Limiting\*\*: Limits API usage.

- \*\*API Authentication\*\*: Verifies API client identity.

- \*\*API Authorization\*\*: Checks API client permissions.

### 6. Security Monitoring and Response

#### Monitoring Components:

- \*\*Security Logging\*\*: Logs security events.

- \*\*Threat Monitoring\*\*: Monitors for security threats.

- \*\*Security Dashboard\*\*: Provides visibility into security status.

- \*\*Security Alerting\*\*: Sends alerts for security incidents.

#### Response Components:

- \*\*Threat Response\*\*: Responds to detected threats.

- \*\*Incident Management\*\*: Manages security incidents.

- \*\*Remediation Workflows\*\*: Automates security incident remediation.

## Security Levels

The application supports five security levels, each with a different set of security features enabled:

### 1. BASIC

- Standard password authentication

- Basic input validation

- Simple access control

- Standard CSRF protection

- Basic security logging

### 2. STANDARD

All BASIC features plus:

- Enhanced input validation

- Rate limiting

- IP blocking for brute force attempts

- Enhanced access control

- Enhanced security logging

### 3. ENHANCED

All STANDARD features plus:

- Multi-factor authentication

- Data encryption for sensitive fields

- API security

- Runtime protection

- Advanced threat detection

- Security notifications

### 4. HIGH

All ENHANCED features plus:

- Mandatory multi-factor authentication

- Complete data encryption

- Advanced API security

- Enhanced runtime protection

- Comprehensive threat detection

- Real-time security monitoring

### 5. MAXIMUM

All HIGH features plus:

- Risk-based authentication

- Quantum-resistant encryption

- Zero-knowledge proofs

- Machine learning anomaly detection

- Comprehensive runtime protection

- Advanced security event correlation

## Data Flow and Security Controls

### User Authentication Flow

1. User submits login credentials.

2. Authentication middleware validates credentials.

3. If enabled, MFA challenge is presented.

4. On successful MFA, session is created.

5. Security event is logged.

6. User is redirected to the requested page.

Security Controls:

- Credential verification

- Brute force protection

- Risk-based authentication (if enabled)

- Multi-factor authentication (if enabled)

- Session security

- Authentication logging

### Data Access Flow

1. User requests data access.

2. Authentication middleware verifies user identity.

3. Authorization middleware checks permissions.

4. If authorized, data is retrieved.

5. If encrypted, data is decrypted.

6. If anonymization is enabled, data is anonymized.

7. Data is returned to the user.

Security Controls:

- Authentication verification

- Authorization check

- Data encryption/decryption

- Data anonymization

- Access logging

### API Request Flow

1. Client sends API request.

2. Authentication middleware verifies API credentials.

3. Rate limiting middleware checks request rate.

4. Input validation middleware validates request data.

5. Authorization middleware checks permissions.

6. Business logic processes the request.

7. Response is generated and returned.

Security Controls:

- API authentication

- Rate limiting

- Input validation

- API authorization

- Security logging

## Threat Model

### Threat Actors

1. \*\*External Attackers\*\*: Unauthorized individuals attempting to gain access.

2. \*\*Malicious Users\*\*: Authorized users with malicious intent.

3. \*\*Compromised Users\*\*: Authorized users whose accounts have been compromised.

4. \*\*Insider Threats\*\*: Employees or contractors with malicious intent.

### Attack Vectors

1. \*\*Authentication Attacks\*\*: Brute force, credential stuffing, session hijacking.

2. \*\*Authorization Attacks\*\*: Privilege escalation, insecure direct object references.

3. \*\*Injection Attacks\*\*: SQL injection, XSS, CSRF.

4. \*\*Data Exposure\*\*: Unencrypted data, excessive data exposure.

5. \*\*Denial of Service\*\*: Resource exhaustion, application DoS.

### Mitigations

1. \*\*Authentication Attacks\*\*:

- Multi-factor authentication

- Brute force protection

- Secure session management

- Risk-based authentication

2. \*\*Authorization Attacks\*\*:

- Role-based access control

- Attribute-based access control

- Permission verification

- Authorization logging

3. \*\*Injection Attacks\*\*:

- Input validation and sanitization

- Prepared statements

- Content Security Policy

- CSRF protection

4. \*\*Data Exposure\*\*:

- Data encryption

- Data anonymization

- Data minimization

- Access control

5. \*\*Denial of Service\*\*:

- Rate limiting

- Resource quotas

- Request throttling

- IP blocking

## Security Monitoring and Logging

### Security Events

The following security events are logged:

1. \*\*Authentication Events\*\*:

- Login attempts (successful and failed)

- Logout events

- Password changes

- MFA events

2. \*\*Authorization Events\*\*:

- Permission checks

- Access denied events

- Privilege changes

3. \*\*Data Access Events\*\*:

- Sensitive data access

- Data modifications

- Data exports

4. \*\*Security Configuration Events\*\*:

- Security level changes

- Security setting changes

- Security feature toggling

5. \*\*Threat Events\*\*:

- Detected threats

- Blocked requests

- Security rule triggers

### Security Metrics

The following security metrics are collected:

1. \*\*Authentication Metrics\*\*:

- Login success rate

- MFA usage rate

- Password strength score

- Account lockouts

2. \*\*Authorization Metrics\*\*:

- Permission denial rate

- Privilege escalation attempts

- Unauthorized access attempts

3. \*\*Threat Metrics\*\*:

- Detected threats by type

- Blocked requests by reason

- Security rule triggers by rule

4. \*\*Performance Metrics\*\*:

- Security processing time

- Security feature impact

- Security rule performance

## Security Testing

### Testing Approaches

1. \*\*Static Application Security Testing (SAST)\*\*:

- Code analysis for security vulnerabilities

- Dependency scanning for known vulnerabilities

- Security code review

2. \*\*Dynamic Application Security Testing (DAST)\*\*:

- Runtime security testing

- Penetration testing

- Vulnerability scanning

3. \*\*Interactive Application Security Testing (IAST)\*\*:

- Runtime code analysis

- Vulnerability detection during execution

- Real-time security feedback

4. \*\*Security Regression Testing\*\*:

- Automated security test cases

- Security feature verification

- Security fix validation

### Automated Security Scanning

The application includes an automated security scanning framework that:

1. Runs scheduled security scans

2. Performs on-demand security testing

3. Provides detailed security reports

4. Recommends security improvements

5. Tracks security vulnerabilities

## Deployment Considerations

### Secure Deployment Checklist

1. \*\*Environment Configuration\*\*:

- Set appropriate security level

- Configure security settings

- Set up security monitoring

2. \*\*Database Security\*\*:

- Enable database encryption

- Configure database access control

- Set up database auditing

3. \*\*Network Security\*\*:

- Configure firewall rules

- Set up TLS/SSL

- Configure network access control

4. \*\*Application Security\*\*:

- Set secure default passwords

- Configure initial security settings

- Set up security monitoring

5. \*\*Operational Security\*\*:

- Configure backup security

- Set up security incident response

- Establish security update process

## Conclusion

The security architecture of the Admin Portal is designed to provide comprehensive protection against a wide range of security threats. By implementing multiple layers of security controls and following security best practices, the application provides a secure environment for administrative operations.

For more detailed information about specific security components, please refer to the respective component documentation.

This completes Part 3 of the Advanced Security Enhancement Plan, focusing on:

1. Comprehensive security testing infrastructure
2. User-friendly admin security dashboard and configuration interfaces
3. Detailed security documentation

The entire three-part implementation plan provides a robust, scalable, and comprehensive approach to enhancing the security of the Admin Portal with priority given to:

1. Performance optimization and database integration (Part 1)
2. Advanced security component implementation (Part 2)
3. User interfaces and documentation (Part 3)

This plan follows industry best practices, provides a clear implementation timeline, and ensures all components work together seamlessly.