**Todo Application Security Implementation Report**

**Project Overview**

This document outlines the security architecture and implementation of a secure Todo application built with .NET Core MVC and Web API using JWT authentication. The application follows a two-project architecture with separation of concerns between the API (business logic/data) and MVC (presentation layer).

Architecture

Client Browser ↔ MVC App (Presentation Layer) ↔ Web API (Business Logic/Data) ↔ Database

Technology Stack

• Backend: .NET Core 8.0

• Authentication: JWT Bearer Tokens + Cookie Authentication

• Database: SQL Server with Entity Framework Core

• Frontend: ASP.NET Core MVC with Razor Views

• Security: ASP.NET Core Identity with custom ApplicationUser

**Security Implementation**

1. Authentication System

JWT Token Generation

public async Task<string> GenerateJwtToken(ApplicationUser user)

{

var tokenHandler = new JwtSecurityTokenHandler();

var key = Encoding.ASCII.GetBytes(\_configuration["JwtSettings:SecretKey"]);

// Get user roles

var roles = await \_userManager.GetRolesAsync(user);

var claims = new List<Claim>

{

new Claim(ClaimTypes.NameIdentifier, user.Id),

new Claim(ClaimTypes.Name, user.UserName),

new Claim(ClaimTypes.Email, user.Email),

};

// Add role claims to token

foreach (var role in roles)

{

claims.Add(new Claim(ClaimTypes.Role, role));

}

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(claims),

Expires = DateTime.UtcNow.AddHours(2),

SigningCredentials = new SigningCredentials(

new SymmetricSecurityKey(key),

SecurityAlgorithms.HmacSha256Signature)

};

var token = tokenHandler.CreateToken(tokenDescriptor);

return tokenHandler.WriteToken(token);

}

Dual Authentication Scheme

• API: JWT Bearer Token authentication

• MVC: Cookie authentication that stores JWT token

2. Authorization & Role Management

Role-Based Access Control

[Authorize(Roles = "Admin")]

public class UsersController : Controller

{

// Only users with Admin role can access these endpoints

}

User Management Features

• Admin role required for user management

• CRUD operations for users

• Role assignment functionality

• User activation/deactivation

• Password reset capability

3. Cross-Site Scripting (XSS) Protection

Input Sanitization

public string Sanitize(string input)

{

if (string.IsNullOrEmpty(input))

return input;

return input

.Replace(/</g, '&lt;')

.Replace(/>/g, '&gt;')

.Replace(/"/g, '&quot;')

.Replace(/'/g, '&#x27;')

.Replace(/\//g, '&#x2F;');

}

Content Security Policy Headers

context.Response.Headers.Add("Content-Security-Policy",

"default-src 'self'; script-src 'self'; style-src 'self' 'unsafe-inline';");

4. SQL Injection Protection

Parameterized Queries with Entity Framework

// Safe from SQL injection

var todos = await \_context.TodoItems

.Where(t => t.UserId == userId && t.IsCompleted == true)

.ToListAsync();

No Raw SQL with String Concatenation

Avoided patterns like:

// UNSAFE - Vulnerable to SQL injection

var sql = $"SELECT \* FROM Todos WHERE UserId = '{userId}'";

5. Cross-Site Request Forgery (CSRF) Protection

**Anti-Forgery Tokens**

services.AddAntiforgery()

and in Html @Html.AntiForgeryToken()

and controller [ValidateAntiForgeryToken]

**Token Validation**

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> CreateTodo(TodoItem todo)

{

// Action requires valid anti-forgery token

}

6. DDoS Protection

Rate Limiting Middleware

public class EnhancedRateLimitMiddleware

{

public async Task InvokeAsync(HttpContext context)

{

var ipAddress = context.Connection.RemoteIpAddress?.ToString();

var requestPath = context.Request.Path;

var cacheKey = $"{ipAddress}\_{requestPath}";

// Check if IP is blocked

var isBlocked = \_cache.Get<bool>($"{ipAddress}\_blocked");

if (isBlocked)

{

context.Response.StatusCode = StatusCodes.Status429TooManyRequests;

return;

}

// Implement request counting and blocking logic

// ...

}

}

**Configuration**

• 60 requests per minute per endpoint per IP

• 5-minute block for exceeding rate limits

• IP-based tracking with memory cache

**7. Secure Headers Implementation**

Security Headers Middleware

public async Task Invoke(HttpContext context)

{

context.Response.Headers.Add("X-Content-Type-Options", "nosniff");

context.Response.Headers.Add("X-Frame-Options", "DENY");

context.Response.Headers.Add("X-XSS-Protection", "1; mode=block");

context.Response.Headers.Add("Referrer-Policy", "no-referrer");

context.Response.Headers.Add("Content-Security-Policy",

"default-src 'self'; script-src 'self'; style-src 'self' 'unsafe-inline';");

await \_next(context);

}

8. Password Security

Strong Password Requirements

services.AddIdentity<ApplicationUser, IdentityRole>(options =>

{

options.Password.RequireDigit = true;

options.Password.RequireLowercase = true;

options.Password.RequireNonAlphanumeric = true;

options.Password.RequireUppercase = true;

options.Password.RequiredLength = 8;

options.Password.RequiredUniqueChars = 1;

})

Password Hashing

• ASP.NET Core Identity uses PBKDF2 with HMAC-SHA256

• 10,000 iterations by default (configurable)

• 128-bit salt + 256-bit subkey

9. Session Management

JWT Token Configuration

• 2-hour expiration time

• Secure token validation

• Clock skew minimization

Cookie Security

options.Cookie.HttpOnly = true;

options.Cookie.SecurePolicy = CookieSecurePolicy.Always;

options.Cookie.SameSite = SameSiteMode.Strict;

10. Data Protection

Entity Framework Data Annotations

public class TodoItem

{

[Required]

[MaxLength(100)]

[RegularExpression(@"^[a-zA-Z0-9\s\.\-\_',!?]\*$",

ErrorMessage = "Title contains invalid characters")]

public string Title { get; set; }

[MaxLength(500)]

public string Description { get; set; }

}

**Security Testing Approach**

1. Authentication Testing

• Verify JWT token generation and validation

• Test role-based access control

• Validate password complexity requirements

2. Authorization Testing

• Confirm admin privileges work correctly

• Test user isolation (users can only access their own todos)

• Verify role assignment functionality

3. Input Validation Testing

• Test XSS protection with script payloads

• Verify SQL injection protection

• Validate model binding and data annotations

4. Rate Limiting Testing

• Test request throttling implementation

• Verify IP blocking functionality

• Confirm rate limit reset behavior

5. Security Headers Verification

• Check all security headers are present

• Verify CSP prevents inline scripts

• Confirm X-Frame-Options denial

Deployment Security Considerations

1. Environment Configuration

• Use different settings for development vs production

• Secure secret management (Azure Key Vault, environment variables)

• Database connection string protection

2. HTTPS Enforcement

// In production environment

app.UseHttpsRedirection();

3. Cookie Security

• Secure flags in production

• HTTPOnly and SameSite attributes

• Proper domain and path settings

4. API Security

• CORS configuration for MVC app only

• JWT token validation settings

• Rate limiting appropriate for production loads

Monitoring and Logging

Security Event Logging

\_logger.LogWarning("Rate limit exceeded by {IPAddress} to {Path}", ipAddress, requestPath);

Audit Logging

• User authentication events

• Role changes

• User management actions

• Security policy violations

**Conclusion**

This Todo application implements comprehensive security measures including:

1. Strong authentication with JWT tokens and cookie sessions

2. Role-based authorization with proper access controls

3. Input validation and sanitization against XSS and SQL injection

4. CSRF protection with anti-forgery tokens

5. DDoS mitigation through rate limiting

6. Secure headers for browser security features

7. Password security with strong complexity requirements

8. Proper session management with secure cookies

The two-project architecture provides separation of concerns while maintaining security across both layers. The implementation follows security best practices and provides defense in depth against common web application vulnerabilities.