Security Guide

Ape Framework - Security Guide

Introduction

Security is a core focus of the Ape Framework. This guide covers the security features, best practices, and recommendations for keeping your application secure.

Authentication Security

ASP.NET Core Identity

The framework uses ASP.NET Core Identity, providing:

- **Secure password hashing** - PBKDF2 with SHA-256
- **Account lockout** - Protection against brute force attacks
- **Email confirmation** - Verify user email addresses
- **Two-factor authentication** - Additional security layer
- **Security stamps** - Automatic session invalidation on security changes

Password Requirements

Default password policy: - Minimum 6 characters - Requires uppercase letter - Requires lowercase letter - Requires digit - Requires non-alphanumeric character

Two-Factor Authentication (2FA)

Users can enable 2FA using authenticator apps: - Google Authenticator - Microsoft Authenticator - Authy - Any TOTP-compatible app

Setup Process: 1. User navigates to profile → Two-Factor Authentication 2. Scans QR code with authenticator app 3. Enters verification code 4. Downloads recovery codes

Recovery Codes: - 10 single-use codes provided - For emergency access if authenticator lost - Should be stored securely offline

Account Lockout

Default settings: - Lockout after 5 failed attempts - Lockout duration: 5 minutes - Protects against password guessing attacks

---

Data Encryption

Credential Encryption

All sensitive credentials stored in the database are encrypted using:

- **Algorithm:** AES-256 (Advanced Encryption Standard)
- **Mode:** CBC (Cipher Block Chaining)
- **Padding:** PKCS7
- **Key Derivation:** SHA-256 from master key
- **IV:** Randomly generated per encryption, prepended to ciphertext

Master Key Management

The master encryption key (`MASTER_CREDENTIAL_KEY_ILLUSTRATE`):

DO: - Store in environment variables - Use at least 32 characters - Include mixed case, numbers, symbols - Keep secure backup

DON'T: - Commit to source control - Store in config files - Share via email or chat - Use predictable values

Key Rotation

To rotate the master key: 1. Export all credentials (decrypted) 2. Update master key in environment 3. Re-import all credentials 4. Verify all credentials work

Note: Changing the key invalidates all existing encrypted data.

Access Control

Role-Based Authorization

Two built-in roles:

Role	Access Level
Admin	Full system access
Member	Limited to member-accessible content

Authorization Attributes

Controllers use `[Authorize]` attributes:

```
[Authorize(Roles = "Admin")]
public class SystemCredentialsController
```

Content Access Levels

Documents and galleries have access levels: - **Member:** Visible to all authenticated users - **Admin:** Visible to administrators only

Access is enforced at: 1. Controller level (authorization checks) 2. View level (conditional display) 3. Query level (filtered results)

Audit Trail

Credential Audit Logging

Every credential access is logged:

Event	Logged Data
Viewed	User, time, IP, user agent
Created	User, time, IP, user agent
Updated	User, time, IP, user agent
Deleted	User, time, IP, user agent
Tested	User, time, IP, success/failure

Email Logging

All email attempts logged: - Recipient - Subject - Provider used (Azure/SMTP) - Success/failure - Error messages - Timestamp

Activity Tracking

User activity tracked: - Last activity timestamp - Updated every 2 minutes (throttled) -

Used for active users dashboard

Input Validation

Anti-Forgery Tokens

All forms include CSRF protection:

```
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create(Model model)
```

Views include token:

```
@Html.AntiForgeryToken()
```

Contact Form Protection

Three-layer spam protection:

1. **Honeypot Fields**
   - Hidden fields (Comment, Website, Company, Url)
   - Bots fill these; humans don't
   - Submission rejected if filled
2. **JavaScript Token**
   - Generated client-side
   - Validates JavaScript execution
   - Bots without JS can't generate
3. **Timing Validation**
   - Form must be open 3+ seconds
   - Instant submissions rejected
   - Prevents automated form bots

SQL Injection Prevention

Entity Framework Core provides: - Parameterized queries by default - No raw SQL concatenation - LINQ-based query building

XSS Prevention

Razor views automatically encode output:

```
@Model.UserInput  <!-- Automatically HTML encoded -->
```

Use `@Html.Raw()` only for trusted content.

Secure File Handling

Document Library

- Files stored outside web root (`/ProtectedFiles/`)
- Access controlled through controller
- File type validation (PDF only)
- Filename sanitization

Photo Gallery

- Images in web root (`/wwwroot/Galleries/`)
- File type validation (images only)
- Image processing sanitizes content
- Filename sanitization

File Upload Security

```
// Validate file type
if (!IsValidPdf(file))
    return BadRequest("Invalid file type");

// Sanitize filename
var safeName = SanitizeFileName(file.FileName);

// Check file size
if (file.Length > MaxFileSize)
    return BadRequest("File too large");
```

Network Security

HTTPS Enforcement

Production deployment should: 1. Use HTTPS only 2. Enable HSTS headers 3. Redirect HTTP to HTTPS

Configure in `Program.cs`:

```
app.UseHttpsRedirection();
app.UseHsts();
```

Cookie Security

Identity cookies configured with: - HttpOnly flag (prevents JavaScript access) - Secure flag (HTTPS only in production) - SameSite policy (CSRF protection)

Environment Security

Development vs Production

Setting	Development	Production
Error details	Shown	Hidden
Logging	Verbose	Minimal
HTTPS	Optional	Required
Debug mode	Enabled	Disabled

Environment Variables

Sensitive values stored in environment: - Database credentials - Encryption keys - API keys

Production Setup (IIS): 1. Set at Application Pool level 2. Not visible in web.config 3. Process-level isolation

---

Security Checklist

Initial Setup

- [ ] Set strong master encryption key (32+ characters)
- [ ] Change default admin password immediately
- [ ] Configure email for account verification
- [ ] Enable HTTPS in production
- [ ] Review default password policy

Ongoing Maintenance

- [ ] Rotate credentials periodically
- [ ] Review credential audit logs weekly
- [ ] Monitor email logs for anomalies
- [ ] Keep framework and packages updated
- [ ] Review active users for suspicious activity
- [ ] Backup database and encryption keys

User Management

- [ ] Remove unused accounts
- [ ] Review admin role assignments
- [ ] Encourage 2FA adoption
- [ ] Document account recovery process

Content Security

- [ ] Review document/gallery access levels
- [ ] Audit uploaded content periodically
- [ ] Remove outdated sensitive documents

Incident Response

Suspected Breach

1. **Assess scope** - What systems/data affected?
2. **Contain** - Disable compromised accounts
3. **Preserve evidence** - Don't delete logs
4. **Notify** - Inform affected users if required
5. **Remediate** - Fix vulnerabilities
6. **Review** - Update security practices

Credential Compromise

If credentials are exposed: 1. Immediately rotate affected credentials 2. Review audit logs for unauthorized access 3. Check for data exfiltration 4. Update all systems using those credentials

Account Takeover

If user account compromised: 1. Lock the account immediately 2. Reset password 3. Revoke all sessions (change security stamp) 4. Disable 2FA and have user re-enable 5. Review account activity logs

---

Compliance Considerations

Data Protection

The framework supports: - User data export (GDPR) - Account deletion capability - Audit trails for accountability - Encryption for data at rest

Logging and Retention

Consider your retention requirements: - Email logs: How long to keep? - Audit logs: Compliance requirements? - User data: Data minimization principles?

Implement log cleanup routines as needed.

Security Resources

ASP.NET Core Security

- Official Security Documentation
- OWASP ASP.NET Core Security Cheat Sheet

General Security

- OWASP Top 10
- CIS Benchmarks

Version: 1.0.0 **Framework:** Ape Framework **Site:** https://Illustrate.net