Broken Access Control

Alexander Tushinsky

Cybersecurity & Software Development Consultant

@ltmodcs alextushinsky.com

Overview

- What is Broken Access Control?
- Examples
- How does OWASP handle it?
- How do we address it?

Broken Access Control

- Leads to sensitive data exposure
- User has access to data they are not authorized to see
- Authorization is not checked properly, allowing accounts to access unauthorized functionality

Demo

Broken Access Control

- Broken user access

OWASP Recommendations

OWASP Top 10

- Deny by default
- Centralized access controls
- Enforce record ownership
- Business limit requirements should be enforced by domain models
- No directory listing in web applications
- No backups of code left within a web-accessible folder
- Alert and notify of suspicious activity
- Rate limit APIs
- Invalidate sessions after logout

OWASP Proactive Controls

- C1: Define Security Requirements
- C2: Leverage Security Frameworks
- C3: Validate All Input
- C6: Implement Digital Identity
- C7: Enforce Access Controls
- C8: Protect Data Everywhere

OWASP ASVS V4 Access Control

- Verify application enforces access control
- Verify that the user can't manipulate access controls
- Use the principle of least privilege
- Access controls should fail securely
- Prevent Insecure Direct Object Reference (IDOR)
- Verify anti-CSRF (Cross-Site Request Forgery)
- Implement 2FA (Two-Factor Authentication)
- No Directory Browsing
- Segregation of roles and duties

Checklist

- Known, centralized Access Control implementation
- Roles or Claims for most applications
- Deny by default
- Fail securely

ASP.NET Core Identity is Microsoft's answer!

ASP.NET Core Identity Implementation

Role-Based Access Control (RBAC)

– ASP.NET Core Identity Roles

- User has a specific role within the application
- Stored in AspNetRoles table
- Relationship between AspNetUsers and AspNetRoles is stored in the AspNetUserRoles table
- Single user can have multiple roles

```
builder.Services.AddDefaultIdentity<IdentityUser>(
        options =>
          options.SignIn.RequireConfirmedAccount = true)
    .AddEntityFrameworkStores<ApplicationDbContext>()
    .AddRoles<IdentityRole>();
```

◀ Enable Roles

```
[Authorize(Roles="Administrator")]
public class AdminController : Controller
{
    private readonly IConfiguration _config;
    private readonly VcDbContext _context;



[Authorize(Roles="User")]
public IActionResult Index()
{
    return View();
}

[Authorize(Roles="User, Administrator")]


[Authorize]
public IActionResult Index()
{
    return View();
}


[AllowAnonymous]
public IActionResult Index()
{
    return View();
}
```

- The “Administrator” role is required for access to any member of the Controller

- Method level “User” role authorization

- Multiple roles allowed

- Any role can access the method, but user must be logged

- Allows anonymous access

Claims-Based Access Control (CBAC)

- Claims
  - Name and value pair
  - More granular than a role
  - Enforced through an Authorization Policy
  - Stored in AspNetUserClaims table
  - Single user can have multiple claims

```
builder.Services.AddAuthorization(options =>
{
    options.AddPolicy("Admin",
        policy => policy
            .RequireClaim("Admin")
            .RequireClaim("Company",
                          "Wired Brain Coffee Co"));
});
```

- Claims policy requiring that a user has an “Admin” claim, as well as have a company claim of “Wired Brain Coffee Co”

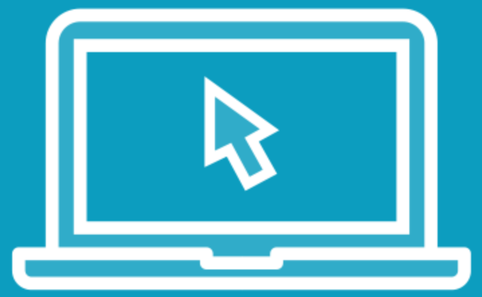
```
[Authorize(Policy=“Admin")]
public class AdminController : Controller
{
    private readonly IConfiguration _config;
    private readonly VcDbContext _context;



[Authorize(Policy="User")]
public IActionResult Index()
{
    return View();
}

[Authorize(Policy=“Admin“, Roles=“Administrator")]
```

- The “Admin” policy at the class level
- Method level “User” policy
- We can assign a mix of claims and roles

User Access

- Filter data by user
- Test your work

Demo



Broken Access Control

- Enable Roles
- Add Authorization annotations to controllers
- Implement HashId NuGet Library
- Review the results

Summary

Broken Access Control

- Looked at the potential issues leading up to broken access control
- Reviewed recommendations from OWASP's Top 10 list, Proactive Controls, and ASVS
- Reviewed ASP.NET Core Identity Roles and Claims

Up Next:
Cryptographic Failures