Protection in the Case of a Breach: Password Hashing & Revocation

Scott Brady
IDENTITY & ACCESS CONTROL LEAD
@scottbrady91 www.scottbrady91.com

Overview

Protecting users after a breach with effective password hashing

Encouraging good passwords with password validation policies

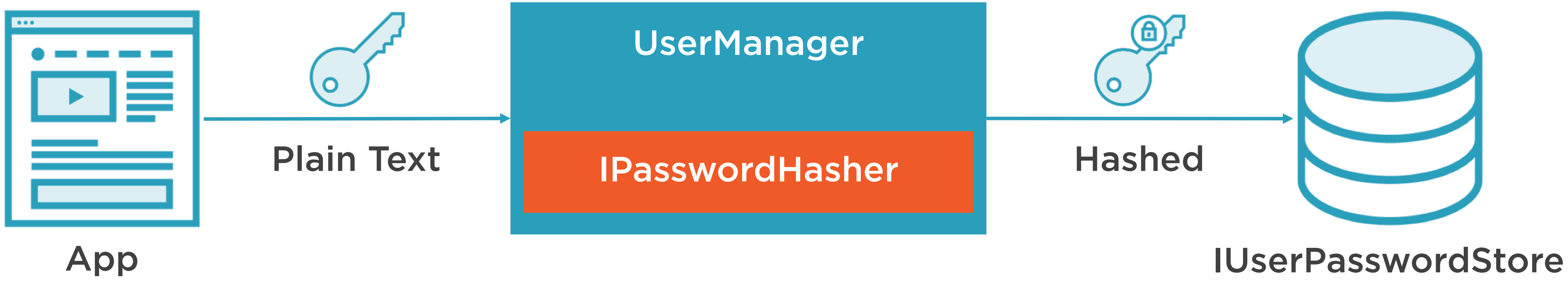
Protection against brute-force attacks

Why Hash Passwords?

Password Hashing

Password		Salt		Hash
my_password	+	QxLUF1bgIAdeQX...	=	9e209040c863f84a31e7...
my_password	+	bv5PehSMfV11CdA...	=	d1d3ec2e6f20fd420d50...

Password Hashing



```
string HashPassword(TUser user, string password);


PasswordVerificationResult VerifyHashedPassword(
    TUser user, string hashedPassword, string providedPassword);
```

IPasswordHasher<TUser>

Responsible for hashing & validating password

PBKDF2 with HMAC-SHA256

128-bit Salt

256-bit Subkey

10,000 Iterations (default)

PasswordHasher Configuration

```
public PasswordHasher(
    IOptions<PasswordHasherOptions> optionsAccessor = null) { ... }
```

PasswordHasher Configuration

```
public void ConfigureServices(IServiceCollection services) {
    // other registrations...
    services.Configure<PasswordHasherOptions>(options => {
        options.IterationCount = 100000;
    });
}
```

```
if (compatibilityMode == PasswordHasherCompatibilityMode.IdentityV2) {
    return Convert.ToBase64String(HashPasswordV2(password, rng));
}
else {
    return Convert.ToBase64String(HashPasswordV3(password, rng));
}
```

Migrating from ASP.NET Identity 2

PasswordHasherOptions.CompatibilityMode

- IdentityV2 (will keep V2 hash format)
- IdentityV3 (will migrate to V3 format)

PasswordHasher Comparison

ASP.NET Identity 2	ASP.NET Core Identity
PBKDF2 with HMAC-SHA1	PBKDF2 with HMAC-SHA256
128-bit Salt	128-bit Salt
256-bit Subkey	256-bit Subkey
1,000 Iterations	10,000 Iterations (default)

```
public void ConfigureServices(IServiceCollection services) {
    // other registrations...
    services.AddScoped<IPasswordHasher<TUser>,
                       BCryptPasswordHasher<TUser>>();
}
```

BCryptPasswordHasher

ScottBrady91.AspNetCore.Identity.BCryptPasswordHasher

“Improving the ASP.NET Core Identity Password Hasher” - scottbrady91.com

```
public interface IPasswordValidator<TUser> {

    Task<IdentityResult> ValidateAsync(
            UserManager<TUser> manager,
            TUser user,
            string password);
}
```

Password Validation

Many implementations can be registered

Policy decisions can be made for the individual

PasswordOptions

```
public int RequiredLength { get; set; } = 6;
public int RequiredUniqueChars { get; set; } = 1;
public bool RequireNonAlphanumeric { get; set; } = true;
public bool RequireLowercase { get; set; } = true;
public bool RequireUppercase { get; set; } = true;
public bool RequireDigit { get; set; } = true;
public interface IUserValidator<TUser> {

   Task<IdentityResult> ValidateAsync(
         UserManager<TUser> manager,
         TUser user);

}
```

User Validation

Intended for any custom validation rules

UserOptions

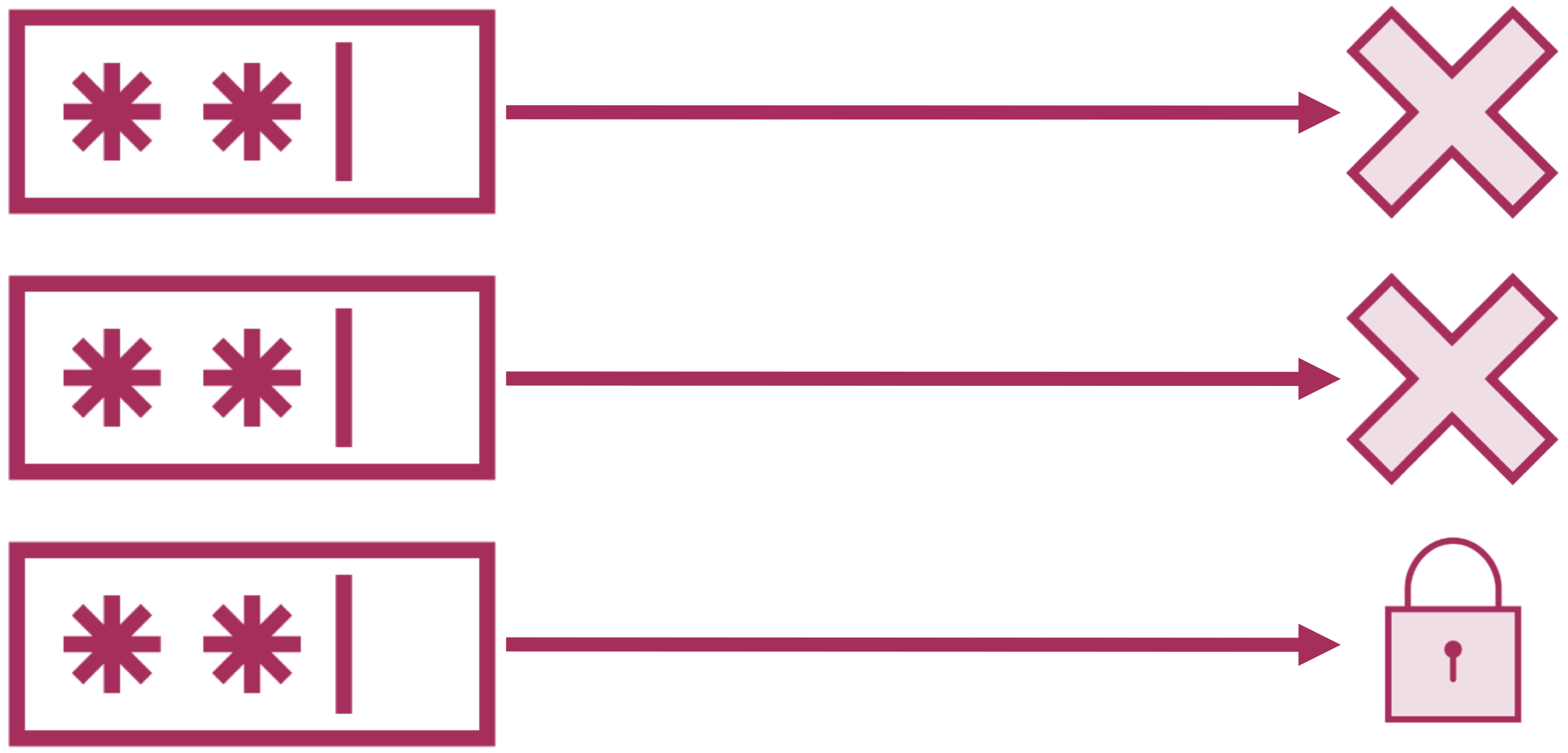
```
public string AllowedUserNameCharacters { get; set; } =
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012345
6789-._@+";

public bool RequireUniqueEmail { get; set; } = false;
```

Demo

Configuring the default user and password validators

Implementing and registering a custom password validator

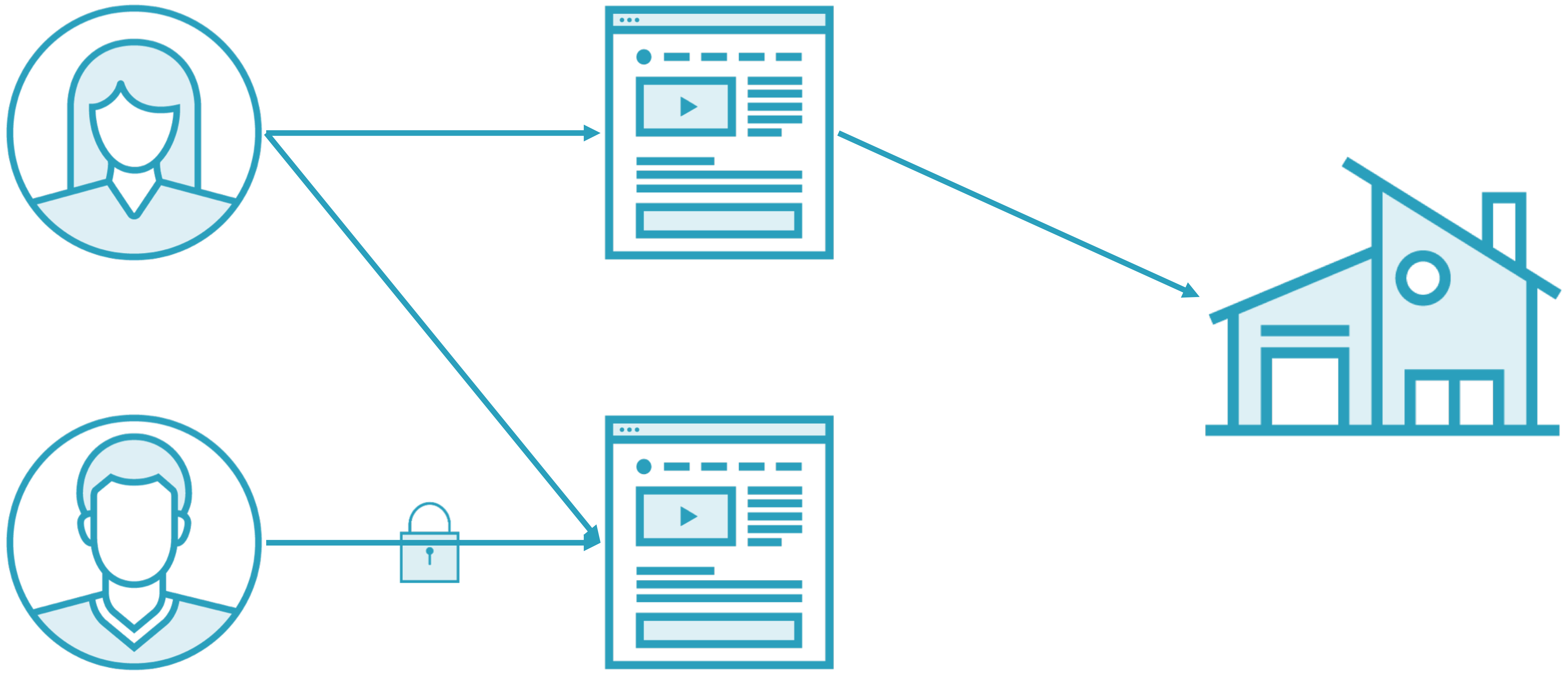
User Lockout



User Lockout Benefits

Brute Force

User Lockout Dangers



Demo

Protecting against brute-force attacks with user lockout

```
.AddCookie(IdentityConstants.ApplicationScheme, o => {
    o.LoginPath = new PathString("/Account/Login");
    o.Events = new CookieAuthenticationEvents {
        OnValidatePrincipal = SecurityStampValidator.ValidatePrincipalAsync
    };
})
```

Security Stamp Validator

Validates every 30 minutes (configurable in `SecurityStampValidatorOptions`)

Ability to log out a user upon account change

Summary

Protecting users after a breach with effective password hashing

Encouraging good passwords with password validation policies

Protection against brute-force attacks