|  |
| --- |
| **123**  **456**  **789** |

**文件編號：654**

**V1.0版**

**2025年03月18日**

**安碁資訊股份有限公司**

**文件制／修訂記錄**

| **版次** | **日期** | **說明** | **作者** | **修訂中** | **定稿** |
| --- | --- | --- | --- | --- | --- |
| 1.0 | 2025.03.18 | Auto | 系統 |  | V |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**【目　　錄】**

[壹、 關於本報告 1](#_Toc192691780)

[一、 報告說明 1](#_Toc192691781)

[二、 盡責聲明 1](#_Toc192691782)

[貳、 檢測作業 2](#_Toc192691783)

[一、 檢測目標 2](#_Toc192691784)

[二、 檢測時間 2](#_Toc192691785)

[三、 檢測方式 3](#_Toc192691786)

[（一） 網頁攀爬 3](#_Toc192691787)

[（二） 網站架構深度分析 3](#_Toc192691788)

[（三） 產出弱點檢測報告與修正建議 3](#_Toc192691789)

[四、 檢測來源IP 3](#_Toc192691790)

[五、 檢測工具 4](#_Toc192691791)

[參、 檢測結果 5](#_Toc192691792)

[一、 弱點摘要 5](#_Toc192691793)

[二、 弱點手動檢核驗證 9](#_Toc192691794)

[（一） https://www.air-go.com.tw 9](#_Toc192691795)

[肆、 弱點說明與修正建議 10](#_Toc192691796)

[一、 10](#_Toc192691797)

[（一） 10](#_Toc192691798)

**【圖 目 錄】**

[圖1 Appscan檢測工具版本畫面 4](#_Toc167927118)

[圖2 jQuery相關弱點手動檢核 7](#_Toc167927119)

[圖3 Cookie屬性弱點手動檢核 8](#_Toc167927120)

[圖4 不安全的第三方鏈結弱點手動檢核 8](#_Toc167927121)

[圖5 SSL相關弱點手動檢核 9](#_Toc167927122)

[圖6 漏洞元件弱點手動檢核-1 10](#_Toc167927123)

[圖7 漏洞元件弱點手動檢核-2 10](#_Toc167927124)

[圖8 SRI（子資源完整性）支援弱點手動檢核 11](#_Toc167927125)

[圖9 Cookie屬性弱點手動檢核 12](#_Toc167927126)

[圖10 SSL相關弱點手動檢核 13](#_Toc167927127)

[圖11 SRI（子資源完整性）支援弱點手動檢核 13](#_Toc167927128)

[圖12 Cookie屬性弱點手動檢核 14](#_Toc167927129)

[圖13 SSL相關弱點手動檢核 15](#_Toc167927130)

[圖14 SRI（子資源完整性）支援弱點手動檢核 15](#_Toc167927131)

**【表 目 錄】**

[表1 檢測目標列表 2](#_Toc167927132)

[表2 弱點檢測分析報告 5](#_Toc167927133)

# 關於本報告

## 報告說明

網站弱點掃描（Web Vulnerability Assessment）是一種評估網站安全性的技術。本報告為安碁資訊股份有限公司承接123（下稱 貴公司）「456」（以下稱本案），依貴我雙方約定之作業範圍，透過自動化工具和手工驗證的方式執行網站弱點掃描，挖掘網站可能存在的漏洞後，就本次作業所見之項目進行說明陳述，提供相關優化、精進之資安防護機制之建議，供 貴公司做為後續防護規劃之依循參考。

## 盡責聲明

自動化工具可以快速發現大量網站漏洞，並透過手工驗證漏洞來提高準確度，減少誤報率。網站弱點掃描並無法找出全部的網站漏洞，當攻擊者試圖利用邏輯性漏洞進行攻擊、或是有了全新的攻擊手法出現，都會使得網站的安全性降低。唯有透過定期地進行網站弱點掃描，才能確保網站的安全性足夠抵禦外來的惡意攻擊。

本報告所使用之分析與檢測手法，為當前資訊安全產業所通用之技術與工具，本公司謹盡合理可能之最大努力，就所見所聞如實提出本報告，容或有不足之處，仍請考量當前科技所能達成之極限，予以諒解。

# 檢測作業

## 檢測目標

網站掃描範圍以 貴單位提供掃描的弱點目標為主，檢測網站如下表。

1. 檢測目標列表

| **編號** | **目標** | **掃描時間** |
| --- | --- | --- |
| 1 | https://ginandjuice.shop | 2025/1/23 下午2:49:20 |
| 2 | https://testasp.vulnweb.com | 2025/2/5 下午11:23:07 |

## 檢測時間

檢測時間自654至987止。在執行階段，在執行階段，除了 貴單位有提出特別排除的測試時間限制，否則原則上一天24小時任何時段均可進行。

## 檢測方式

由外部連線方式執行網站安全檢測程序，如為到府作業則由本公司工程師攜帶掃描機器至 貴單位，對 貴單位測試標的以內網方式連線進行檢測，檢測方式與流程說明如下：

### 網頁攀爬

將測試標的所屬網站目錄進行全面性的網頁攀爬與URL、路徑紀錄等。

### 網站架構深度分析

對網頁攀爬所記錄之網站程式與路徑，進行網站架構、網頁應用程式的安全測試及分析。

### 產出弱點檢測報告與修正建議

## 檢測來源IP

* 321.0.0.1

## 檢測工具

本次網站安全檢測將採用國際知名的Appscan檢測軟體進行掃描，透過掃描可檢測網站主機之各式安全性問題與弱點。為避免影響　貴單位網站服務正常，測試過程中不會採用任何植入性或破壞性測試。

Appscan檢測工具使用版本如下圖：

|  |
| --- |
|  |

1. Appscan檢測工具版本畫面

主要檢測項目如下：

* 整體網站架構檢測。
* Web Application安全檢測，如：SQL Injection、Cross Site Scripting等弱點項目檢測。
* CGI程式存取權限檢查（如：GET、PUT、DELETE）。
* 網頁程式碼檢測與分析，依據不同程式碼PHP、JSP、ASP等，進行相對應的弱點測試與檢查。
* OWASP TOP 10 2021弱點檢核項目。

# 檢測結果

## 弱點摘要

本次網站弱點掃描結果共有 23 個弱點總數，其中有 2 個嚴重風險弱點，主要為未正確地消毒使用者所輸入的危險字元等弱點，弱點等級及數量如下圖。

|  |
| --- |
|  |

1. 風險等級弱點數目及總和

表列等級分為嚴重、高、中（Critical / High / Medium）等三級，建議立即安排修補與進一步確認。其餘低風險弱點請參閱附件說明。

1. 弱點檢測分析報告

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 編號 | 目標網站 | 弱點名稱 | 弱點描述 | 弱點等級 | 弱點總數 | 弱點數量 | | |
| **C** | **H** | **M** |
| 1 | https://ginandjuice.shop | 有弱點的元件 | 未安裝協力廠商產品的最新修補程式或緊急修復程式 | 高 | 2 | 0 | 2 | 9 |
| SameSite 屬性不安全、不適當或遺漏的 Cookie | SameSite 屬性不適當、不安全或遺漏的機密 Cookie | 中 | 3 |
| 低強度密碼組合 - ROBOT 攻擊：伺服器支援有弱點的密碼組合 | 使用不安全的方式配置 Web 伺服器或應用程式伺服器 | 中 | 1 |
| 偵測到低強度密碼：並非所有密碼組合都支援完全秘密轉遞 | 使用不安全的方式配置 Web 伺服器或應用程式伺服器 | 中 | 1 |
| 加密的階段作業 (SSL) Cookie 中遺漏安全屬性 | Web 應用程式會透過 SSL 傳送未受保護的 Cookie | 中 | 1 |
| 有弱點的元件 | 未安裝協力廠商產品的最新修補程式或緊急修復程式 | 中 | 3 |
| 2 | https://testasp.vulnweb.com | MsSQL 盲目 SQL 注入（頻外） | 未正確地消毒使用者所輸入的危險字元 | 嚴重 | 1 | 2 | 5 | 5 |
| 盲目的 SQL 注入 | 未正確地消毒使用者所輸入的危險字元 | 嚴重 | 1 |
| 反映型跨網站指令碼 | 未正確地消毒使用者所輸入的危險字元 | 高 | 1 |
| 擷取毒害空值位元組 Windows 檔案 | 未正確地消毒使用者所輸入的危險字元未檢查使用者輸入是否有 '..' （雙點）字串 | 高 | 1 |
| 整數溢位 | 未對送入的參數值執行適當的範圍檢查未執行驗證，以確定使用者輸入符合預期的資料類型 | 高 | 1 |
| 預存的跨網站指令碼 | 未正確地消毒使用者所輸入的危險字元 | 高 | 2 |
| Microsoft FrontPage 目錄清單 | 已使用不適當的安全設定來安裝 FrontPage Server Extensions | 中 | 1 |
| SameSite 屬性不安全、不適當或遺漏的 Cookie | SameSite 屬性不適當、不安全或遺漏的機密 Cookie | 中 | 2 |
| Windows 檔案參數變更 | 未正確地消毒使用者所輸入的危險字元未檢查使用者輸入是否有 '..' （雙點）字串 | 中 | 1 |
| 有弱點的元件 | 未安裝協力廠商產品的最新修補程式或緊急修復程式 | 中 | 1 |
| **總計** | | | | | | **2** | **7** | **14** |

## 弱點手動檢核驗證

本次檢測針對網掃工具檢核出弱點等級嚴重、高、中以上分類的弱點進行手動驗證作業，以確認工具掃描出的弱點確實存在。

### https://www.air-go.com.tw

#### jQuery 上的跨網站 Scripting (CVE-2020-11022)

* 弱點等級：中

#### jQuery 上的跨網站 Scripting (CVE-2020-11023)

* 一張含有 文字, 螢幕擷取畫面 的圖片

  自動產生的描述弱點等級：中

1. jQuery相關弱點手動檢核

# 弱點說明與修正建議

## https://ginandjuice.shop

### 有弱點的元件

#### 弱點原因

攻擊者有可能使用 Web 伺服器來攻擊其他網站，讓其身分更加隱密

#### 弱點說明

有弱點的元件可能導致應用程式出現各種弱點

#### 修補建議

升級到該組件的最新版本。如果該組件已達到其生命終結（EOL），請用最近支持的替代品替換該組件。我們強烈建議聯繫該產品的供應商，看看是否最近有補丁或修復可用。

### SameSite 屬性不安全、不適當或遺漏的 Cookie

#### 弱點原因

將 Cookie 限制為第一方或相同網站環境定義以預防 Cookie 資訊洩漏。如果沒有設置額外的保護措施（如反 CSRF 記號），攻擊者可以延伸為偽造跨網站要求 (CSRF) 攻擊。

#### 弱點說明

將 Cookie 限制為第一方或相同網站環境定義，藉此預防 Cookie 資訊洩漏。如果沒有設置額外的保護措施（如反 CSRF 記號），攻擊者可以延伸為偽造跨網站要求 (CSRF) 攻擊。SameSite 屬性控制跨網域要求的 Cookie 傳送方式。屬性的值有三個：「Lax」、「Strict」或「None」。如果您使用「None」，網站可以建立與其他網站之間的跨網域 POST HTTP 要求，而瀏覽器會自動將 Cookie 新增到該要求中。如果沒有設置額外的保護措施（如反 CSRF 記號），可能會引發偽造跨網站要求 (CSRF) 攻擊。模式與其用法：「Lax」模式：Cookie 只會連同最上層 GET 要求一同傳送。「Strict」模式：即使使用者遵循其他網站的鏈結，Cookie 也不會連同任何跨網站用法一同傳送。「None」模式：Cookie 將連同跨網站要求一同傳送。擁有「Lax」或「None」的屬性必須設定「Secure」旗標，而且必須透過 https 傳輸。範例 - Set-Cookie: key=value; SameSite=Lax;Secure建議選項是將屬性設定為「Strict」。範例 - Set-Cookie: key=value; SameSite=Strict

#### 修補建議

[1] 檢閱將 SameSite Cookie 屬性配置為建議值的可能解決方案。[2] 將 Cookie 限制為第一方或相同網站環境定義。[3] 將 Cookie 的 SameSite 屬性設定為 Strict 並加以驗證，確保 Cookie 只能在第一方環境定義中傳送。[4] 或者，如果您想要放鬆第一方環境定義的限制，請將 Cookie 的 SameSite 屬性設定為 Lax 並啟用 Secure 旗標，再透過 HTTPS 傳輸，同時加以驗證。

### 低強度密碼組合 - ROBOT 攻擊：伺服器支援有弱點的密碼組合

#### 弱點原因

有可能竊取或操作客戶階段作業和 Cookie，並可能用來假冒合法的使用者，讓駭客可以檢視或變更使用者記錄，以及用該使用者的身分來執行交易

#### 弱點說明

攻擊者可以獲得解密的 RSA 密文，或透過伺服器的私密金鑰簽署任意訊息。此攻擊允許攻擊者將有弱點主機的記錄流量解密。「ROBOT - Return Of Bleichenbacher's Oracle Threat」攻擊適用於 TLS-RSA 金鑰交換，開頭為 TLS\_RSA 的所有密碼組合名稱都會使用此金鑰交換。攻擊可能會惡意探索有弱點的伺服器，這類伺服器會根據密文的有效性，回覆不同的訊息。攻擊會仰賴 RSA 的彈性，允許具有 RSA 公開金鑰的任何人快速增加加密的純文字，並且使用 PKCS #1 v1.5 填補格式，讓攻擊者非常有可能建立有效訊息。在 TLS 實作中具有缺陷的主機（支援使用 RSA 加密 TLS\_RSA 密碼 的 TLS 密碼模式）容易遭到 ROBOT 攻擊。

#### 修補建議

如果您使用的是有弱點的產品，請檢查是否有更新和修補程式修補程式和受影響的產品清單可能隨著時間經過，而有所不同。如要進一步瞭解修補程式和受影響的產品，請參閱 https://robotattack.org/#patches我們也建議您完全淘汰 TLS 中 RSA 加密型的金鑰交換，因為其不會提供秘密轉遞。

### 偵測到低強度密碼：並非所有密碼組合都支援完全秘密轉遞

#### 弱點原因

有可能竊取或操作客戶階段作業和 Cookie，並可能用來假冒合法的使用者，讓駭客可以檢視或變更使用者記錄，以及用該使用者的身分來執行交易

#### 弱點說明

一旦主機的私密金鑰受損，攻擊者就可以產生秘密金鑰解密安全通訊AppScan 偵測到低強度密碼組合，其不支援 PFS - 完全秘密轉遞當主機不支援 PFS 時，在「用戶端」與「伺服器」建立連線的大部分情況下，「用戶端」會使用「伺服器」的「公用金鑰」解密預先主要秘密，接著將其傳送至「伺服器」。然後，伺服器會使用其「私密金鑰」來解密預先主秘密，以便「用戶端」與「伺服器」可以使用預先主金鑰產生階段作業金鑰。如果主機的私密金鑰受損，攻擊者可以解密預先主秘密，並在稍後產生用於安全通訊的「秘密金鑰」。這可能是因為未針對「用戶端」與「伺服器」之間的每個單一通訊產生任何唯一的階段作業金鑰，例如，當使用 DHE - Diffie-Hellman Ephemeral 或 ECDHE -Elliptic Curve Diffie-Hellman Ephemeral 金鑰交換時，可以實現這類通訊。藉由 PFS「用戶端」與「伺服器」之間的每一個新階段作業，會產生新的隨機金鑰以建立預先主秘密，因此，即使其中一個「金鑰」受損，攻擊者仍可以只危害「用戶端」與「伺服器」之間的一個階段作業，而不是所有先前的通訊階段作業。

#### 修補建議

使用其他組合搭配 ECDHE - Elliptic Curve Diffie-Hellman Ephemeral 及 DHE - Diffie-Hellman Ephemeral 金鑰交換，來支援 PFS

### 加密的階段作業 (SSL) Cookie 中遺漏安全屬性

#### 弱點原因

有可能竊取在加密階段作業期間傳送的使用者和階段作業資訊 (Cookie)

#### 弱點說明

有可能竊取在加密階段作業期間傳送的使用者和階段作業資訊 (Cookie)在應用程式測試期間，偵測到在加密階段作業中，所測試的 Web 應用程式設定了不含 "secure" 屬性的 Cookie。由於這個 Cookie 未包含 "secure" 屬性，因此，也可能在未加密的階段作業期間將它傳給網站。任何以明碼傳給伺服器的 Cookie、階段作業記號或使用者認證之類的資訊都可能被竊，稍後可用來盜用身分或使用者模擬。此外，若干隱私權法規指出，使用者認證之類的機密性資訊一律以加密方式傳給網站

#### 修補建議

基本上，Cookie 的唯一必要屬性是 "name" 欄位。常見的選用屬性如下："comment"、"domain"、"path"，等等。"secure" 屬性必須相應地設定，才能防止以未加密的方式來傳送 Cookie。如需如何設定安全旗標的相關資訊，請前往以下鏈結，並參閱 OWASP " 安全屬性 " 題要https://cheatsheetseries.owasp.org/cheatsheets/Session\_Management\_Cheat\_Sheet.html#secure-attributeRFC 2965 指出：「Secure 屬性（不含值）會指示使用者代理程式再次傳回這個 Cookie 時，都只用（未指定）安全方法來聯絡原始伺服器，以保護 Cookie 中之資訊的機密性與確實性。」如需進一步的參照，請參閱「HTTP 狀態管理機制」RFC 2965，位置如下： http://www.ietf.org/rfc/rfc2965.txt如需使用「HTTP 狀態管理」的「最佳現行實務」，請參閱： http://tools.ietf.org/html/rfc2964

### 有弱點的元件

#### 弱點原因

攻擊者有可能使用 Web 伺服器來攻擊其他網站，讓其身分更加隱密

#### 弱點說明

有弱點的元件可能導致應用程式出現各種弱點

#### 修補建議

升級到該組件的最新版本。如果該組件已達到其生命終結（EOL），請用最近支持的替代品替換該組件。我們強烈建議聯繫該產品的供應商，看看是否最近有補丁或修復可用。

## https://testasp.vulnweb.com

### MsSQL 盲目 SQL 注入（頻外）

#### 弱點原因

有可能檢視、修改或刪除資料庫項目和表格

#### 弱點說明

有可能檢視、修改或刪除資料庫項目和表格此軟體會使用受到外部影響的輸入，來建構全部或部分的 SQL 指令，但無法在傳送到資料庫時，讓可以修改 SQL 指令的元素失效。當使用者可控制的輸入中，沒有足夠的移除或引用 SQL 語法時，產生的 SQL 查詢可能導致系統將這些輸入解譯為 SQL，而非一般使用者資料。這可用於變更查詢邏輯以略過安全性檢查，或插入其他可修改後端資料庫的陳述式，可能包含執行系統指令。舉例來說，我們有一個包含登入表單的 HTML 頁面，最後會採用使用者輸入，在資料庫上執行下列 SQL 查詢： SELECT \* FROM accounts WHERE username='$user' AND password='$pass' $user 和 $pass 這兩個變數包含使用者在登入表單中所輸入的使用者憑證。如果使用者輸入「jsmith」作為使用者名稱，並且將「Demo1234」作為密碼，SQL 查詢看起來會像這樣： SELECT \* FROM accounts WHERE username='jsmith' AND password='Demo1234' 但如果使用者輸入「'」（單引號）作為使用者名稱和密碼，SQL 查詢看起來會像這樣： SELECT \* FROM accounts WHERE username=''' AND password=''' 這當然就是型態異常的 SQL 查詢，這會呼叫錯誤訊息（可能在 HTTP 回應中傳回）。這類錯誤會告知攻擊者 SQL 已成功注入，導致攻擊者嘗試進一步的攻擊向量。盲目 SQL 注入與 SQL 注入相似，其差別在於攻擊者如要利用盲目 SQL，不需要在回應中查詢 SQL 錯誤。因此 AppScan 用於識別的方法也不一樣。AppScan 不會嘗試呼叫 SQL 錯誤，但會找出容易遭受 SQL 注入的參數。AppScan 可透過讓 SQL 伺服器本身執行我們可識別的外部要求，來執行這個動作。這項技術需要傳送弱點參數（在 SQL 查詢中嵌入的參數）已經過修改的要求，以便外部要求表示資料是否在 SQL 查詢的脈絡中使用。每個 SQL 伺服器的修改都是獨一無二的。頻外盲目 SQL 注入背後的概念是，即使攻擊者沒有直接從資料庫中收到資料，也能夠收到資料庫的內容（透過錯誤訊息或遭洩漏資訊的形式）。

#### 修補建議

以下是幾種減輕風險的技巧：[1] 策略：程式庫或架構使用檢查過且不容許此弱點的程式庫或架構，或提供使其更容易避免的建構項。[2] 策略：參數化如果可行，請使用結構化機制來自動強制分開資料與程式碼。這些機制可能可以自動提供相關的引用、編碼和驗證，而不是在每次產生輸出時，都依賴開發人員來提供。[3] 策略：環境強化 使用完成必要作業所需的最低專用權，來執行程式碼。[4] 策略：輸出編碼如果儘管有風險，您還是需要使用動態產生的查詢字串或指令，適當地括住引數且避免在這些引數內使用任何特殊字元。[5] 策略：輸入驗證假設所有輸入都是惡意的。請使用「接受已知良好」輸入驗證策略：嚴格符合規格的可接受輸入白名單。拒絕未嚴格符合規格的任何輸入，或將其轉換成嚴格符合規格者。請勿只靠使用黑名單來偵測惡意或形態異常的輸入。不過，在偵測潛在攻擊或判斷哪些輸入因形態異常而應該斷然拒絕時，黑名單就很有用。以下是保護 Web 應用程式免於遭受 SQL 注入攻擊的兩種可能方式：[1] 使用儲存程序，而不用動態建置的 SQL 查詢字串。參數傳給 SQL Server 儲存程序的方式，會防止使用單引號和連字號。如何在 ASP.NET 中使用儲存程序的範例如下： ' Visual Basic example Dim DS As DataSet Dim MyConnection As SqlConnection Dim MyCommand As SqlDataAdapter Dim SelectCommand As String = "select \* from users where username = @username" ... MyCommand.SelectCommand.Parameters.Add(New SqlParameter("@username", SqlDbType.NVarChar, 20)) MyCommand.SelectCommand.Parameters("@username").Value = UserNameField.Value // C# example String selectCmd = "select \* from Authors where state = @username"; SqlConnection myConnection = new SqlConnection("server=..."); SqlDataAdapter myCommand = new SqlDataAdapter(selectCmd, myConnection); myCommand.SelectCommand.Parameters.Add(new SqlParameter("@username", SqlDbType.NVarChar, 20)); myCommand.SelectCommand.Parameters["@username"].Value = UserNameField.Value; [2] 您可以利用驗證控制項，將輸入驗證新增到「Web 表單」頁面中。驗證控制項提供適用於標準驗證之所有一般類型的簡易使用機制（例如：測試日期是否有效，或值是否在範圍內），以及用來提供自訂編寫驗證的方式。此外，驗證控制項也可讓您完整自訂向使用者顯示錯誤資訊的方式。驗證控制項可以搭配「Web 表單」頁面類別檔所處理的任何控制項來使用，其中包括 HTML 和 Web 伺服器控制項。如果要確定使用者輸入只包含有效值，您可以使用下列其中一個驗證控制項：a. "RangeValidator"：檢查使用者的輸入（值）是否在指定的上下界限之間。您可以檢查數字、英文字母和日期之配對內的範圍。b. "RegularExpressionValidator"：檢查輸入是否符合正規表示式所定義的型樣。這類型的驗證可讓您檢查可預期的字元序列，例如：社會保險號碼、電子郵件位址、電話號碼、郵遞區號等等中的字元序列。重要注意事項：驗證控制項不會封鎖使用者輸入，或變更頁面處理流程；它們只會設定錯誤狀態，以及產生錯誤訊息。在執行進一步的應用程式專屬動作之前，程式設計師負責測試程式碼中的控制項狀態。檢查使用者輸入有效性有兩種方式：1. 測試一般錯誤狀態：在您的程式碼中，測試頁面的 IsValid 內容。這個內容會累積頁面上所有驗證控制項的 IsValid 內容值（使用邏輯 AND）。如果其中一個驗證控制項設為無效，頁面內容便會傳回 False。2. 測試個別控制項的錯誤狀態：在迴圈中處理頁面的驗證器集合，集合中含有指向所有驗證控制項的參照。之後，您便可以檢查每個驗證控制項的 IsValid 內容。\*\* 備妥陳述式：保護您的應用程式免於遭受 SQL 注入（亦即，惡意竄改 SQL 參數）有三種可能的方式。請利用下列方式，而非動態建置 SQL 陳述式：[1] PreparedStatement，經過前置編譯而存放在 PreparedStatement 物件儲存區中。PreparedStatement 定義用來登錄輸入參數的設定元，輸入參數相容於支援的 JDBC SQL 資料類型。例如，setString 應該用於 VARCHAR 或 LONGVARCHAR 類型的輸入參數（請參閱 Java API，以取得進一步的詳細資料）。這個設定輸入參數的方式，可以防止攻擊者注入不正確的字元（如單引號）來操作 SQL 陳述式。如何使用 J2EE 中之 PreparedStatement 的範例： // J2EE PreparedStatemenet Example // Get a connection to the database Connection myConnection; if (isDataSourceEnabled()) { // using the DataSource to get a managed connection Context ctx = new InitialContext(); myConnection = ((DataSource)ctx.lookup(datasourceName)).getConnection(dbUserName, dbPassword); } else { try { // using the DriverManager to get a JDBC connection Class.forName(jdbcDriverClassPath); myConnection = DriverManager.getConnection(jdbcURL, dbUserName, dbPassword); } catch (ClassNotFoundException e) { ... } } ... try { PreparedStatement myStatement = myConnection.prepareStatement("select \* from users where username = ?"); myStatement.setString(1, userNameField); ResultSet rs = myStatement.executeQuery(); ... rs.close(); } catch (SQLException sqlException) { ... } finally { myStatement.close(); myConnection.close(); } [2] CallableStatement，延伸 PreparedStatement 來執行資料庫 SQL 儲存程序。這個類別繼承 PreparedStatement 的輸入設定元（請參閱上面的 [1]）。下列範例假設已建立好這個資料庫儲存程序：CREATE PROCEDURE select\_user (@username varchar(20))AS SELECT \* FROM USERS WHERE USERNAME = @username;如何利用 J2EE 中的 CallableStatement 來執行上述儲存程序的範例： // J2EE PreparedStatemenet Example // Get a connection to the database Connection myConnection; if (isDataSourceEnabled()) { // using the DataSource to get a managed connection Context ctx = new InitialContext(); myConnection = ((DataSource)ctx.lookup(datasourceName)).getConnection(dbUserName, dbPassword); } else { try { // using the DriverManager to get a JDBC connection Class.forName(jdbcDriverClassPath); myConnection = DriverManager.getConnection(jdbcURL, dbUserName, dbPassword); } catch (ClassNotFoundException e) { ... } } ... try { PreparedStatement myStatement = myConnection.prepareCall("{?= call select\_user ?,?}"); myStatement.setString(1, userNameField); myStatement.registerOutParameter(1, Types.VARCHAR); ResultSet rs = myStatement.executeQuery(); ... rs.close(); } catch (SQLException sqlException) { ... } finally { myStatement.close(); myConnection.close(); } [3] Entity Bean，在持續性儲存庫機制中代表 EJB 商業物件。Entity Bean 有兩種類型：Bean 管理和儲存器管理。當使用 Bean 管理的持續性時，開發人員負責撰寫存取資料庫的 SQL 程式碼（請參閱上述 [1] 和 [2] 區段）。當使用儲存器管理的持續性時，EJB 儲存器會自動產生 SQL 程式碼。因此，儲存器要負責防止惡意嘗試竄改產生的 SQL 程式碼。如何使用 J2EE 中之 Entity Bean 的範例： // J2EE EJB Example try { // lookup the User home interface UserHome userHome = (UserHome)context.lookup(User.class); // find the User remote interface User = userHome.findByPrimaryKey(new UserKey(userNameField)); ... } catch (Exception e) { ... } 建議的 Java 工具N/A參照https://docs.oracle.com/javase/7/docs/api/java/sql/PreparedStatement.htmlhttps://docs.oracle.com/javase/7/docs/api/java/sql/CallableStatement.html\*\* 驗證輸入資料：雖然為了使用者的方便，可以在用戶端層提供資料驗證，但必須利用 Servlet，在伺服器層執行資料驗證。用戶端驗證原本就不安全，因為它們可以輕易略過，例如：停用 JavaScript。好的設計通常需要 Web 應用程式架構提供伺服器端公用程式常式來驗證下列項目：[1] 必要欄位[2] 欄位資料類型（依預設，所有 HTTP 要求參數都是 String）[3] 欄位長度[4] 欄位範圍[5] 欄位選項[6] 欄位型樣[7] Cookie 值[8] HTTP 回應在實務中，好的做法是在 "Validator" 公用程式類別中將上述常式當作靜態方法來實作。下列各節說明範例驗證器類別[1] 必要欄位一律檢查確認欄位不是空值，欄位長度大於零，且不含在前端及尾端的空格。如何驗證必要欄位的範例： // Java example to validate required fields public Class Validator { ... public static boolean validateRequired(String value) { boolean isFieldValid = false; if (value != null && value.trim().length() > 0) { isFieldValid = true; } return isFieldValid; } ... } ... String fieldValue = request.getParameter("fieldName"); if (Validator.validateRequired(fieldValue)) { // fieldValue is valid, continue processing request ... } [2] 欄位資料類型在 Web 應用程式中，所輸入的輸入參數設定不良。例如，所有 HTTP 要求參數或 Cookie 值都是 String 類型。開發人員負責確認輸入的資料類型正確。請利用 Java 基本封套類別來檢查欄位值是否能安全轉換成所需要的基本資料類型。如何驗證數值欄位（int 類型）的範例： // Java example to validate that a field is an int number public Class Validator { ... public static boolean validateInt(String value) { boolean isFieldValid = false; try { Integer.parseInt(value); isFieldValid = true; } catch (Exception e) { isFieldValid = false; } return isFieldValid; } ... } ... // check if the HTTP request parameter is of type int String fieldValue = request.getParameter("fieldName"); if (Validator.validateInt(fieldValue)) { // fieldValue is valid, continue processing request ... } 在實務中，好的做法是將所有 HTTP 要求參數轉換成它們的相關資料類型。例如，開發人員應該將要求參數的 "integerValue" 儲存在要求屬性中，再依照下列範例所示來使用它： // Example to convert the HTTP request parameter to a primitive wrapper data type // and store this value in a request attribute for further processing String fieldValue = request.getParameter("fieldName"); if (Validator.validateInt(fieldValue)) { // convert fieldValue to an Integer Integer integerValue = Integer.getInteger(fieldValue); // store integerValue in a request attribute request.setAttribute("fieldName", integerValue); } ... // Use the request attribute for further processing Integer integerValue = (Integer)request.getAttribute("fieldName"); ... 應用程式應該處理的主要 Java 資料類型：- Byte- Short- Integer- Long- Float- Double- Date[3] 欄位長度一律確定輸入參數（HTTP 要求參數或 Cookie 值）在長度下限及/或長度上限的範圍內。驗證 userName 欄位的長度是否在 8-20 個字元之間的範例： // Example to validate the field length public Class Validator { ... public static boolean validateLength(String value, int minLength, int maxLength) { String validatedValue = value; if (!validateRequired(value)) { validatedValue = ""; } return (validatedValue.length() >= minLength && validatedValue.length() <= maxLength); } ... } ... String userName = request.getParameter("userName"); if (Validator.validateRequired(userName)) { if (Validator.validateLength(userName, 8, 20)) { // userName is valid, continue further processing ... } } [4] 欄位範圍一律確定輸入參數在函數需求定義的範圍內。驗證輸入 numberOfChoices 是否在 10-20 之間的範例： // Example to validate the field range public Class Validator { ... public static boolean validateRange(int value, int min, int max) { return (value >= min && value <= max); } ... } ... String fieldValue = request.getParameter("numberOfChoices"); if (Validator.validateRequired(fieldValue)) { if (Validator.validateInt(fieldValue)) { int numberOfChoices = Integer.parseInt(fieldValue); if (Validator.validateRange(numberOfChoices, 10, 20)) { // numberOfChoices is valid, continue processing request ... } } } [5] 欄位選項Web 應用程式通常會向使用者呈現一組可供選擇的選項（例如：使用 SELECT HTML 標籤），但無法執行伺服器端驗證來確保所選的值是容許的選項之一。請記住，惡意的使用者可以輕易修改任何選項值。請一律對照函數需求所定義容許的選項來驗證所選的使用者值。對照容許的選項清單來驗證使用者選項的範例： // Example to validate user selection against a list of options public Class Validator { ... public static boolean validateOption(Object[] options, Object value) { boolean isValidValue = false; try { List list = Arrays.asList(options); if (list != null) { isValidValue = list.contains(value); } } catch (Exception e) { } return isValidValue; } ... } ... // Allowed options String[] options = {"option1", "option2", "option3"); // Verify that the user selection is one of the allowed options String userSelection = request.getParameter("userSelection"); if (Validator.validateOption(options, userSelection)) { // valid user selection, continue processing request ... } [6] 欄位型樣一律檢查使用者輸入是否符合功能需求所定義的型樣。比方說，如果 userName 欄位只應接受英數字元，且不區分大小寫，請使用下列正規表示式：^[a-zA-Z0-9]\*$Java 1.3 或更早的版本不包括任何正規表示式套件。建議搭配 Java 1.3 來使用「Apache 正規表示式套件」（請參閱下列「資源」），以解決這項支援欠缺。執行正規表示式驗證的範例： // Example to validate that a given value matches a specified pattern // using the Apache regular expression package import org.apache.regexp.RE; import org.apache.regexp.RESyntaxException; public Class Validator { ... public static boolean matchPattern(String value, String expression) { boolean match = false; if (validateRequired(expression)) { RE r = new RE(expression); match = r.match(value); } return match; } ... } ... // Verify that the userName request parameter is alpha-numeric String userName = request.getParameter("userName"); if (Validator.matchPattern(userName, "^[a-zA-Z0-9]\*$")) { // userName is valid, continue processing request ... } Java 1.4 引進了新的正規表示式套件 (java.util.regex)。以下是 Validator.matchPattern 的修正版，使用新的 Java 1.4 正規表示式套件： // Example to validate that a given value matches a specified pattern // using the Java 1.4 regular expression package import java.util.regex.Pattern; import java.util.regexe.Matcher; public Class Validator { ... public static boolean matchPattern(String value, String expression) { boolean match = false; if (validateRequired(expression)) { match = Pattern.matches(expression, value); } return match; } ... } [7] Cookie 值請利用 javax.servlet.http.Cookie 物件來驗證 Cookie 值。依應用程式需求而定，相同的驗證規則（說明如上）也適用於 Cookie 值，例如：驗證必要的值、驗證長度，等等。驗證必要 Cookie 值的範例： // Example to validate a required cookie value // First retrieve all available cookies submitted in the HTTP request Cookie[] cookies = request.getCookies(); if (cookies != null) { // find the "user" cookie for (int i=0; i<cookies.length; ++i) { if (cookies[i].getName().equals("user")) { // validate the cookie value if (Validator.validateRequired(cookies[i].getValue()) { // valid cookie value, continue processing request ... } } } } [8] HTTP 回應[8-1] 過濾使用者輸入如果要保護應用程式免於遭受跨網站指令碼，請將機密字元轉換成對應的字元實體來消毒 HTML。以下是 HTML 區分字元：< > " ' % ; ) ( & +將機密字元轉換成對應的字元實體來過濾指定字串的範例： // Example to filter sensitive data to prevent cross-site scripting public Class Validator { ... public static String filter(String value) { if (value == null) { return null; } StringBuffer result = new StringBuffer(value.length()); for (int i=0; i<value.length(); ++i) { switch (value.charAt(i)) { case '<': result.append("<"); break; case '>': result.append(">"); break; case '"': result.append("""); break; case '\'': result.append("'"); break; case '%': result.append("%"); break; case ';': result.append(";"); break; case '(': result.append("("); break; case ')': result.append(")"); break; case '&': result.append("&"); break; case '+': result.append("+"); break; default: result.append(value.charAt(i)); break; } return result; } ... } ... // Filter the HTTP response using Validator.filter PrintWriter out = response.getWriter(); // set output response out.write(Validator.filter(response)); out.close(); Java Servlet API 2.3 引進了「過濾器」，它支援截取及轉換 HTTP 要求或回應。利用「Servlet 過濾器」，以 Validator.filter 來消毒回應的範例： // Example to filter all sensitive characters in the HTTP response using a Java Filter. // This example is for illustration purposes since it will filter all content in the response, including HTML tags! public class SensitiveCharsFilter implements Filter { ... public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException { PrintWriter out = response.getWriter(); ResponseWrapper wrapper = new ResponseWrapper((HttpServletResponse)response); chain.doFilter(request, wrapper); CharArrayWriter caw = new CharArrayWriter(); caw.write(Validator.filter(wrapper.toString())); response.setContentType("text/html"); response.setContentLength(caw.toString().length()); out.write(caw.toString()); out.close(); } ... public class CharResponseWrapper extends HttpServletResponseWrapper { private CharArrayWriter output; public String toString() { return output.toString(); } public CharResponseWrapper(HttpServletResponse response){ super(response); output = new CharArrayWriter(); } public PrintWriter getWriter(){ return new PrintWriter(output); } } } } [8-2] 維護 Cookie 安全當機密資料儲存在 Cookie 中，請務必利用 Cookie.setSecure（布林旗標），在 HTTP 回應中設定 Cookie 的安全旗標，以指示瀏覽器利用 HTTPS 或 SSL 之類的安全通訊協定來傳送 Cookie。維護 "user" Cookie 安全的範例： // Example to secure a cookie, i.e. instruct the browser to // send the cookie using a secure protocol Cookie cookie = new Cookie("user", "sensitive"); cookie.setSecure(true); response.addCookie(cookie); 建議的 Java 工具適用於伺服器端驗證的兩個主要的 Java 架構如下：[1] Jakarta 一般驗證器（整合 Struts 1.1）「Jakarta 一般驗證器」實作上述所有資料驗證需求，是一個功能強大的架構。這些規則配置在定義表單欄位輸入驗證規則的 XML 檔中。依預設，Struts 支援在利用 Struts 'bean:write' 標籤撰寫的所有資料上，過濾 [8] HTTP 回應中輸出的危險字元。設定 'filter=false' 旗標可以停用這個過濾。Struts 定義下列基本輸入驗證器，但也可以定義自訂驗證器：required：如果欄位含有空格以外的任何字元，便告成功。mask：如果值符合 mask 屬性給定的正規表示式，便告成功。range：如果值在 min 和 max 屬性給定的值範圍內（(value >= min) 且 (value <= max)），便告成功。maxLength：如果欄位長度小於或等於 max 屬性，便告成功。minLength：如果欄位長度大於或等於 min 屬性，便告成功。byte、short、integer、long、float、double：如果值可以轉換成對應的基本類型，便告成功。date：如果值代表有效日期，便告成功。可提供日期型樣。creditCard：如果值可能是有效的信用卡號碼，便告成功。e-mail：如果值可能是有效的電子郵件位址，便告成功。利用「Struts 驗證器」驗證 loginForm 之 userName 欄位的範例： <form-validation> <global> ... <validator name="required" classname="org.apache.struts.validator.FieldChecks" method="validateRequired" msg="errors.required"> </validator> <validator name="mask" classname="org.apache.struts.validator.FieldChecks" method="validateMask" msg="errors.invalid"> </validator> ... </global> <formset> <form name="loginForm"> <!-- userName is required and is alpha-numeric case insensitive --> <field property="userName" depends="required,mask"> <!-- message resource key to display if validation fails --> <msg name="mask" key="login.userName.maskmsg"/> <arg0 key="login.userName.displayname"/> <var> <var-name>mask</var-name> <var-value>^[a-zA-Z0-9]\*$</var-value> </var> </field> ... </form> ... </formset> </form-validation> [2] JavaServer Faces 技術「JavaServer Faces 技術」是一組代表使用者介面元件、管理元件狀態、處理事件及輸入驗證的 Java API (JSR 127)。JavaServer Faces API 實作下列基本驗證器，但可以定義自訂驗證器：validate\_doublerange：登錄元件的 DoubleRangeValidatorvalidate\_length：登錄元件的 LengthValidatorvalidate\_longrange：登錄元件的 LongRangeValidatorvalidate\_required：登錄元件的 RequiredValidatorvalidate\_stringrange：登錄元件的 StringRangeValidatorvalidator：登錄元件的自訂 ValidatorJavaServer Faces API 定義下列 UIInput 和 UIOutput 展現器（標籤）：input\_date：接受以 java.text.Date 實例格式化的 java.util.Dateoutput\_date：顯示以 java.text.Date 實例格式化的 java.util.Dateinput\_datetime：接受以 java.text.Time 實例格式化的 java.util.Dateoutput\_datetime：顯示以 java.text.Time 實例格式化的 java.util.Dateinput\_number：顯示以 java.text.NumberFormat 格式化的數值資料類型（java.lang.Number 或基本資料類型）output\_number：顯示以 java.text.NumberFormat 格式化的數值資料類型（java.lang.Number 或基本資料類型）input\_text：接受單行字串。output\_text：顯示單行字串。input\_time：接受以 java.text.DateFormat 時間實例格式化的 java.util.Dateoutput\_time：顯示以 java.text.DateFormat 時間實例格式化的 java.util.Dateinput\_hidden：可讓頁面作者將隱藏變數併入頁面input\_secret：接受不含空格的單行文字，輸入之時，將它顯示成一組星號input\_textarea：接受多行文字output\_errors：顯示整個頁面的錯誤訊息，或指定之用戶端 ID 的相關錯誤訊息output\_label：將巢狀元件顯示為指定輸入欄位的標籤output\_message：顯示本地化訊息利用 JavaServer Faces 驗證 loginForm 之 userName 欄位的範例： <%@ taglib uri="https://docs.oracle.com/javaee/6/tutorial/doc/glxce.html" prefix="h" %> <%@ taglib uri="http://mrbool.com/how-to-create-a-login-validation-with-jsf-java-server-faces/27046" prefix="f" %> ... <jsp:useBean id="UserBean" class="myApplication.UserBean" scope="session" /> <f:use\_faces> <h:form formName="loginForm" > <h:input\_text id="userName" size="20" modelReference="UserBean.userName"> <f:validate\_required/> <f:validate\_length minimum="8" maximum="20"/> </h:input\_text> <!-- display errors if present --> <h:output\_errors id="loginErrors" clientId="userName"/> <h:command\_button id="submit" label="Submit" commandName="submit" /><p> </h:form> </f:use\_faces> 參照Java API 1.3 - https://www.oracle.com/java/technologies/java-archive-13docs-downloads.htmlJava API 1.4 - https://www.oracle.com/java/technologies/java-archive-142docs-downloads.htmlJava Servlet API 2.3 - https://mvnrepository.com/artifact/javax.servlet/javax.servlet-apiJava 正規表示式套件 - http://jakarta.apache.org/regexp/Jakarta 驗證器 - http://jakarta.apache.org/commons/validator/JavaServer Faces 技術 - http://www.javaserverfaces.org/\*\* 錯誤處理：許多 J2EE Web 應用程式架構都遵循「模型視圖控制器 (MVC)」型樣。在這個型樣中，Servlet 扮演控制器的角色。Servlet 將應用程式處理程序委派給 JavaBean，例如，EJB Session Bean（模型）。之後，Servlet 再將要求轉遞給 JSP（視圖）來呈現處理結果。Servlet 應該檢查所有輸入、輸出、回覆碼、錯誤碼及已知的異常狀況，以確保會實際進行預期的處理程序。資料驗證可以保護應用程式免於遭受惡意的資料竄改，有效的錯誤處理策略則是防止應用程式意外揭露內部錯誤訊息（例如：異常狀況堆疊追蹤）所不可或缺。好的錯誤處理策略會處理下列項目：[1] 定義錯誤[2] 報告錯誤[3] 呈現錯誤[4] 錯誤對映[1] 定義錯誤應該避免將錯誤訊息寫在應用程式層（例如：Servlet）。相反地，應用程式應該使用對映至已知之應用程式失敗的錯誤索引鍵。在實務中，好的做法是定義錯誤索引鍵來對映至 HTML 表單欄位或其他 Bean 內容的驗證規則。比方說，如果需要 "user\_name" 欄位，它必須是英數字元，且在資料庫中必須是唯一，便應該定義下列錯誤索引鍵：(a) ERROR\_USERNAME\_REQUIRED：這個錯誤索引鍵用來顯示一則訊息，通知使用者需要 "user\_name" 欄位；(b) ERROR\_USERNAME\_ALPHANUMERIC：這個錯誤索引鍵用來顯示一則訊息，通知使用者 "user\_name" 欄位應該是英數字元；(c) ERROR\_USERNAME\_DUPLICATE：這個錯誤索引鍵用來顯示一則訊息，通知使用者 "user\_name" 值在資料庫中重複；(d) ERROR\_USERNAME\_INVALID：這個錯誤索引鍵用來顯示一則一般訊息，通知使用者 "user\_name" 值無效；在實務中，好的做法是定義下列架構 Java 類別來儲存及報告應用程式錯誤：- ErrorKeys：定義所有錯誤索引鍵 // Example: ErrorKeys defining the following error keys: // - ERROR\_USERNAME\_REQUIRED // - ERROR\_USERNAME\_ALPHANUMERIC // - ERROR\_USERNAME\_DUPLICATE // - ERROR\_USERNAME\_INVALID // ... public Class ErrorKeys { public static final String ERROR\_USERNAME\_REQUIRED = "error.username.required"; public static final String ERROR\_USERNAME\_ALPHANUMERIC = "error.username.alphanumeric"; public static final String ERROR\_USERNAME\_DUPLICATE = "error.username.duplicate"; public static final String ERROR\_USERNAME\_INVALID = "error.username.invalid"; ... } - Error：封裝個別錯誤 // Example: Error encapsulates an error key. // Error is serializable to support code executing in multiple JVMs. public Class Error implements Serializable { // Constructor given a specified error key public Error(String key) { this(key, null); } // Constructor given a specified error key and array of placeholder objects public Error(String key, Object[] values) { this.key = key; this.values = values; } // Returns the error key public String getKey() { return this.key; } // Returns the placeholder values public Object[] getValues() { return this.values; } private String key = null; private Object[] values = null; } - Errors：封裝錯誤的集合 // Example: Errors encapsulates the Error objects being reported to the presentation layer. // Errors are stored in a HashMap where the key is the bean property name and value is an // ArrayList of Error objects. public Class Errors implements Serializable { // Adds an Error object to the Collection of errors for the specified bean property. public void addError(String property, Error error) { ArrayList propertyErrors = (ArrayList)errors.get(property); if (propertyErrors == null) { propertyErrors = new ArrayList(); errors.put(property, propertyErrors); } propertyErrors.put(error); } // Returns true if there are any errors public boolean hasErrors() { return (errors.size > 0); } // Returns the Errors for the specified property public ArrayList getErrors(String property) { return (ArrayList)errors.get(property); } private HashMap errors = new HashMap(); } 以下是利用上述架構類別來處理 "user\_name" 欄位驗證錯誤的範例： // Example to process validation errors of the "user\_name" field. Errors errors = new Errors(); String userName = request.getParameter("user\_name"); // (a) Required validation rule if (!Validator.validateRequired(userName)) { errors.addError("user\_name", new Error(ErrorKeys.ERROR\_USERNAME\_REQUIRED)); } // (b) Alpha-numeric validation rule else if (!Validator.matchPattern(userName, "^[a-zA-Z0-9]\*$")) { errors.addError("user\_name", new Error(ErrorKeys.ERROR\_USERNAME\_ALPHANUMERIC)); } else { // (c) Duplicate check validation rule // We assume that there is an existing UserValidationEJB session bean that implements // a checkIfDuplicate() method to verify if the user already exists in the database. try { ... if (UserValidationEJB.checkIfDuplicate(userName)) { errors.addError("user\_name", new Error(ErrorKeys.ERROR\_USERNAME\_DUPLICATE)); } } catch (RemoteException e) { // log the error logger.error("Could not validate user for specified userName: " + userName); errors.addError("user\_name", new Error(ErrorKeys.ERROR\_USERNAME\_DUPLICATE); } } // set the errors object in a request attribute called "errors" request.setAttribute("errors", errors); ... [2] 報告錯誤報告 Web 層應用程式錯誤的方式有兩種：(a) Servlet 錯誤機制(b) JSP 錯誤機制[2-a] Servlet 錯誤機制Servlet 報告錯誤的可能方式如下：- 轉遞至輸入 JSP（已將錯誤儲存在要求屬性中），或- 利用 HTTP 錯誤碼引數來呼叫 response.sendError，或- 擲出異常狀況在實務中，好的做法是處理所有已知的應用程式錯誤（依照 [1] 區段所說明），將它們儲存在要求屬性中，再轉遞給輸入 JSP。輸入 JSP 應該顯示錯誤訊息，並提示使用者重新輸入資料。下列範例說明如何轉遞到輸入 JSP (userInput.jsp)： // Example to forward to the userInput.jsp following user validation errors RequestDispatcher rd = getServletContext().getRequestDispatcher("/user/userInput.jsp"); if (rd != null) { rd.forward(request, response); } 如果 Servlet 無法轉遞到已知的 JSP 頁面，第二個選項是利用 response.sendError 方法，設定 HttpServletResponse.SC\_INTERNAL\_SERVER\_ERROR 引數（狀態碼 500）來報告錯誤。請參閱 javax.servlet.http.HttpServletResponse javadoc，以取得各種 HTTP 狀態碼的詳細資訊。傳回 HTTP 錯誤的範例： // Example to return a HTTP error code RequestDispatcher rd = getServletContext().getRequestDispatcher("/user/userInput.jsp"); if (rd == null) { // messages is a resource bundle with all message keys and values response.sendError(HttpServletResponse.SC\_INTERNAL\_SERVER\_ERROR, messages.getMessage(ErrorKeys.ERROR\_USERNAME\_INVALID)); } 作為最後的手段，Servlet 可以擲出異常狀況，它必須是下列類別之一的子類別：- RuntimeException- ServletException- IOException[2-b] JSP 錯誤機制 JSP 頁面依照下列範例所示來定義 errorPage 指引，從而提供了執行時期異常狀況的處理機制： <%@ page errorPage="/errors/userValidation.jsp" %> 未捕捉的 JSP 異常狀況是轉遞到指定的 errorPage，而原始異常狀況則設在稱為 javax.servlet.jsp.jspException 的要求參數中。錯誤頁面必須包含 isErrorPage 指引，如下所示： <%@ page isErrorPage="true" %> isErrorPage 指引會使 "exception" 變數起始設定為所擲出的異常狀況物件。[3] 呈現錯誤J2SE Internationalization API 提供用來提出應用程式資源以及將訊息格式化的公用程式類別，其中包括：(a) 資源組(b) 訊息格式化[3-a] 資源組資源組會將本地化的資料與使用它的原始碼分開，從而支援國際化。每個資源組都會儲存特定語言環境之鍵值配對的對映。通常是利用或延伸 java.util.PropertyResourceBundle，它會將內容儲存在外部內容檔中，如下列範例所示： ################################################ # ErrorMessages.properties ################################################ # required user name error message error.username.required=User name field is required # invalid user name format error.username.alphanumeric=User name must be alphanumeric # duplicate user name error message error.username.duplicate=User name {0} already exists, please choose another one ... 定義多重資源可以支援不同的語言環境（因此稱為資源組）。例如，定義 ErrorMessages\_fr.properties 可以支援資源組系列的法國成員。如果要求之語言環境的資源成員不存在，便會使用預設成員。在上述範例中，預設資源是 ErrorMessages.properties。依使用者的語言環境而定，應用程式（JSP 或 Servlet）會從適當的資源擷取內容。[3-b] 訊息格式化J2SE 標準類別 java.util.MessageFormat 提供以取代位置保留元來建立訊息的一般方式。MessageFormat 物件包含內嵌了格式指定元的型樣字串，如下所示： // Example to show how to format a message using placeholder parameters String pattern = "User name {0} already exists, please choose another one"; String userName = request.getParameter("user\_name"); Object[] args = new Object[1]; args[0] = userName; String message = MessageFormat.format(pattern, args); 以下是利用 ResourceBundle 和 MessageFormat 來呈現錯誤訊息的更綜合性的範例： // Example to render an error message from a localized ErrorMessages resource (properties file) // Utility class to retrieve locale-specific error messages public Class ErrorMessageResource { // Returns the error message for the specified error key in the environment locale public String getErrorMessage(String errorKey) { return getErrorMessage(errorKey, defaultLocale); } // Returns the error message for the specified error key in the specified locale public String getErrorMessage(String errorKey, Locale locale) { return getErrorMessage(errorKey, null, locale); } // Returns a formatted error message for the specified error key in the specified locale public String getErrorMessage(String errorKey, Object[] args, Locale locale) { // Get localized ErrorMessageResource ResourceBundle errorMessageResource = ResourceBundle.getBundle("ErrorMessages", locale); // Get localized error message String errorMessage = errorMessageResource.getString(errorKey); if (args != null) { // Format the message using the specified placeholders args return MessageFormat.format(errorMessage, args); } else { return errorMessage; } } // default environment locale private Locale defaultLocale = Locale.getDefaultLocale(); } ... // Get the user's locale Locale userLocale = request.getLocale(); // Check if there were any validation errors Errors errors = (Errors)request.getAttribute("errors"); if (errors != null && errors.hasErrors()) { // iterate through errors and output error messages corresponding to the "user\_name" property ArrayList userNameErrors = errors.getErrors("user\_name"); ListIterator iterator = userNameErrors.iterator(); while (iterator.hasNext()) { // Get the next error object Error error = (Error)iterator.next(); String errorMessage = ErrorMessageResource.getErrorMessage(error.getKey(), userLocale); output.write(errorMessage + "\r\n"); } } 建議您定義自訂 JSP 標籤（如 displayErrors）來疊代處理及呈現錯誤訊息，如上述範例所示。[4] 錯誤對映一般而言，「Servlet 儲存器」會傳回對應於回應狀態碼或異常狀況的預設錯誤頁面。您可以利用自訂錯誤頁面來指定狀態碼或異常狀況與 Web 資源之間的對映。在實務中，好的做法是開發不揭露內部錯誤狀態的靜態錯誤頁面（依預設，大部分 Servlet 儲存器都會報告內部錯誤訊息）。這項對映是依照下列範例所指定，配置在「Web 部署描述子 (web.xml)」中： <!-- Mapping of HTTP error codes and application exceptions to error pages --> <error-page> <exception-type>UserValidationException</exception-type> <location>/errors/validationError.html</error-page> </error-page> <error-page> <error-code>500</exception-type> <location>/errors/internalError.html</error-page> </error-page> <error-page> ... </error-page> ... 建議的 Java 工具適用於伺服器端驗證的兩個主要的 Java 架構如下：[1] Jakarta 一般驗證器（整合 Struts 1.1）「Jakarta 一般驗證器」是依照上述說明來定義錯誤處理機制的 Java 架構。驗證規則配置在 XML 檔中，檔案定義了表單欄位的輸入驗證規則以及對應的驗證錯誤索引鍵。Struts 提供國際化支援，供您利用資源組和訊息格式化來建置本地化應用程式。利用「Struts 驗證器」驗證 loginForm 之 userName 欄位的範例： <form-validation> <global> ... <validator name="required" classname="org.apache.struts.validator.FieldChecks" method="validateRequired" msg="errors.required"> </validator> <validator name="mask" classname="org.apache.struts.validator.FieldChecks" method="validateMask" msg="errors.invalid"> </validator> ... </global> <formset> <form name="loginForm"> <!-- userName is required and is alpha-numeric case insensitive --> <field property="userName" depends="required,mask"> <!-- message resource key to display if validation fails --> <msg name="mask" key="login.userName.maskmsg"/> <arg0 key="login.userName.displayname"/> <var> <var-name>mask</var-name> <var-value>^[a-zA-Z0-9]\*$</var-value> </var> </field> ... </form> ... </formset> </form-validation> Struts JSP 標籤庫定義了有條件地顯示一組累計錯誤訊息的 "errors" 標籤，如下列範例所示： <%@ page language="java" %> <%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %> <%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %> <html:html> <head> <body> <html:form action="/logon.do"> <table border="0" width="100%"> <tr> <th align="right"> <html:errors property="username"/> <bean:message key="prompt.username"/> </th> <td align="left"> <html:text property="username" size="16"/> </td> </tr> <tr> <td align="right"> <html:submit><bean:message key="button.submit"/></html:submit> </td> <td align="right"> <html:reset><bean:message key="button.reset"/></html:reset> </td> </tr> </table> </html:form> </body> </html:html> [2] JavaServer Faces 技術「JavaServer Faces 技術」是一組代表使用者介面元件、管理元件狀態、處理事件、驗證輸入，以及支援國際化的 Java API (JSR 127)。JavaServer Faces API 定義了 "output\_errors" UIOutput 展現器，以顯示整個頁面的錯誤訊息，或指定之用戶端 ID 的相關錯誤訊息。利用 JavaServer Faces 驗證 loginForm 之 userName 欄位的範例： <%@ taglib uri="https://docs.oracle.com/javaee/6/tutorial/doc/glxce.html" prefix="h" %> <%@ taglib uri="http://mrbool.com/how-to-create-a-login-validation-with-jsf-java-server-faces/27046" prefix="f" %> ... <jsp:useBean id="UserBean" class="myApplication.UserBean" scope="session" /> <f:use\_faces> <h:form formName="loginForm" > <h:input\_text id="userName" size="20" modelReference="UserBean.userName"> <f:validate\_required/> <f:validate\_length minimum="8" maximum="20"/> </h:input\_text> <!-- display errors if present --> <h:output\_errors id="loginErrors" clientId="userName"/> <h:command\_button id="submit" label="Submit" commandName="submit" /><p> </h:form> </f:use\_faces> 參照Java API 1.3 - https://www.oracle.com/java/technologies/java-archive-13docs-downloads.htmlJava API 1.4 - https://www.oracle.com/java/technologies/java-archive-142docs-downloads.htmlJava Servlet API 2.3 - https://mvnrepository.com/artifact/javax.servlet/javax.servlet-apiJava 正規表示式套件 - http://jakarta.apache.org/regexp/Jakarta 驗證器 - http://jakarta.apache.org/commons/validator/JavaServer Faces 技術 - http://www.javaserverfaces.org/\*\* 過濾使用者輸入將任何資料傳給 SQL 查詢之前，一律應該先利用白名單技術來適當過濾。這無論如何強調都不為過。過濾使用者輸入可讓許多注入缺失，在抵達資料庫之前便得到更正。\*\* 引用使用者輸入不論任何資料類型，只要資料庫允許，便用單引號括住所有使用者資料，始終是好的觀念。MySQL 接受這個格式化技術。\*\* 跳出資料值如果您使用 MySQL 4.3.0 或更新的版本，您應該用 mysql\_real\_escape\_string() 來跳出所有字串。如果使用舊版的 MySQL，便應該使用 mysql\_escape\_string() 函數。如果未使用 MySQL，您可以選擇使用特定資料庫的特定跳出函數。如果不知道跳出函數，您可以選擇使用較一般的跳出函數，例如，addslashes()。如果使用 PEAR DB 資料庫抽象層，您可以使用 DB::quote() 方法或使用 ? 之類的查詢位置保留元，它會自動跳出取代位置保留元的值。參照http://ca3.php.net/mysql\_real\_escape\_stringhttp://ca.php.net/mysql\_escape\_stringhttp://ca.php.net/addslasheshttp://pear.php.net/package-info.php?package=DB\*\* 驗證輸入資料：雖然為了使用者的方便，可以在用戶端層提供資料驗證，但一律必須在伺服器層執行資料驗證。用戶端驗證原本就不安全，因為它們可以輕易略過，例如：停用 JavaScript。好的設計通常需要 Web 應用程式架構提供伺服器端公用程式常式來驗證下列項目：[1] 必要欄位[2] 欄位資料類型（依預設，所有 HTTP 要求參數都是 String）[3] 欄位長度[4] 欄位範圍[5] 欄位選項[6] 欄位型樣[7] Cookie 值[8] HTTP 回應在實務中，好的做法是實作一或多個驗證各個應用程式參數的函數。下列各節說明一些檢查範例。[1] 必要欄位一律檢查確認欄位不是空值，欄位長度大於零，且不含在前端及尾端的空格。如何驗證必要欄位的範例： // PHP example to validate required fields function validateRequired($input) { ... $pass = false; if (strlen(trim($input))>0){ $pass = true; } return $pass; ... } ... if (validateRequired($fieldName)) { // fieldName is valid, continue processing request ... } [2] 欄位資料類型在 Web 應用程式中，所輸入的輸入參數設定不良。例如，所有 HTTP 要求參數或 Cookie 值都是 String 類型。開發人員負責確認輸入的資料類型正確。[3] 欄位長度一律確定輸入參數（HTTP 要求參數或 Cookie 值）在長度下限及/或長度上限的範圍內。[4] 欄位範圍一律確定輸入參數在函數需求定義的範圍內。[5] 欄位選項Web 應用程式通常會向使用者呈現一組可供選擇的選項（例如：使用 SELECT HTML 標籤），但無法執行伺服器端驗證來確保所選的值是容許的選項之一。請記住，惡意的使用者可以輕易修改任何選項值。請一律對照函數需求所定義容許的選項來驗證所選的使用者值。[6] 欄位型樣一律檢查使用者輸入是否符合功能需求所定義的型樣。比方說，如果 userName 欄位只應接受英數字元，且不區分大小寫，請使用下列正規表示式：^[a-zA-Z0-9]+$[7] Cookie 值依應用程式需求而定，相同的驗證規則（說明如上）也適用於 Cookie 值，例如：驗證必要的值、驗證長度，等等。[8] HTTP 回應[8-1] 過濾使用者輸入如果要保護應用程式免於遭受跨網站指令碼，開發人員應該將機密字元轉換成對應的字元實體來消毒 HTML。以下是 HTML 區分字元：< > " ' % ; ) ( & +PHP 包括一些 htmlentities() 之類的自動化消毒公用程式函數： $input = htmlentities($input, ENT\_QUOTES, 'UTF-8'); 此外，為了避免「跨網站指令碼」的 UTF-7 變式，您應該明確定義回應的 Content-Type 標頭，例如： <?php header('Content-Type: text/html; charset=UTF-8'); ?> [8-2] 維護 Cookie 安全當機密資料儲存在 Cookie 中，且透過 SSL 來傳輸它時，請務必先在 HTTP 回應中設定 Cookie 的安全旗標。這會指示瀏覽器只透過 SSL 連線來使用這個 Cookie。您可以利用下列程式碼範例來維護 Cookie 安全： <$php $value = "some\_value"; $time = time()+3600; $path = "/application/"; $domain = ".example.com"; $secure = 1; setcookie("CookieName", $value, $time, $path, $domain, $secure, TRUE); ?> 此外，我們建議您使用 HttpOnly 旗標。當 HttpOnly 旗標設為 TRUE 時，只能透過 HTTP 通訊協定來存取 Cookie。這表示無法用 JavaScript 之類的 Scripting 語言存取 Cookie。這個設定有助於實際減少透過 XSS 攻擊來盜用身分的情況（不過，並非所有瀏覽器都支援）。PHP 5.2.0 新增 HttpOnly 旗標。參照[1] 利用「HTTP 專用 Cookie」緩和「跨網站指令碼」： http://msdn2.microsoft.com/en-us/library/ms533046.aspx[2] PHP 安全聯盟： http://phpsec.org/[3] PHP & Web 應用程式安全部落格 (Chris Shiflett)： http://shiflett.org/

### 盲目的 SQL 注入

#### 弱點原因

有可能檢視、修改或刪除資料庫項目和表格

#### 弱點說明

有可能檢視、修改或刪除資料庫項目和表格軟體會使用受外部影響的輸入來建構所有或部分 SQL 指令，但是在傳送至資料庫時，其無法使可修改 SQL 指令的元素失效。如果未針對可由使用者控制的輸入進行充分移除或加引號，其所產生的 SQL 查詢可能會造成這些輸入被解譯為 SQL 而非一般使用者資料。這可能會被用來改變查詢邏輯以略過安全檢查，或插入其他陳述式以修改後端資料庫，其中可能包括執行系統指令。例如，假設有一個 HTML 頁面含有登入表單，最終會使用使用者輸入對資料庫執行下列 SQL 查詢： SELECT \* FROM accounts WHERE username='$user' AND password='$pass' $user 和 $pass 這兩個變數包含使用者在登入表單中所輸入的使用者認證。如果使用者輸入 "jsmith" 作為使用者名稱，且輸入 "Demo1234" 作為密碼，則 SQL 查詢會看起來如下： SELECT \* FROM accounts WHERE username='jsmith' AND password='Demo1234' 但是，如果使用者輸入 "'"（一個單引號）作為使用者名稱，且輸入 "'"（一個單引號）作為密碼，則 SQL 查詢會看起來如下： SELECT \* FROM accounts WHERE username=''' AND password=''' 這當然是形態異常的 SQL 查詢，且會呼叫可能在 HTTP 回應中傳回的錯誤訊息。這類錯誤會通知攻擊者 SQL 注入已成功，而會引導攻擊者嘗試進一步攻擊手法。「盲目的 SQL 注入」與「SQL 注入」類似。差異在於運用它，攻擊者不需尋找回應中的 SQL 錯誤。因此，AppScan 用來識別它的方法也不同。AppScan 不會嘗試呼叫 SQL 錯誤，而是透過多項要求來操作應用程式的邏輯，以找出很容易遭到 SQL 注入攻擊的 Script。這項技術需要傳送有弱點的參數（被內嵌在 SQL 查詢中的參數）已遭修改的要求，使得回應能夠指出是否在 SQL 查詢環境定義中使用資料。這項修改包括搭配原始字串來使用 AND 布林表示式，使它一時得出 True 一時得出 False。在一個情況中，最後結果應該與原始結果相同（登入成功）；在另一情況中，結果應該明顯不同（登入失敗）。在某些少見的情況中，得出 True 的 OR 表示式也很有用。如果原始資料是數值，就可以採用較簡單的技巧。假設原始資料是 123。這項資料可以在某項要求中取代為 0+123，在另一項要求中取代為 456+123。第一項要求的結果應該與原始結果相同，第二項要求的結果應該不同（因為得出的數字是 579）。在某些情況中，我們仍需要上面所說明的攻擊版本（使用 AND 和 OR），但並不跳出字串上下文。「盲目的 SQL 注入」背後的概念是，即使未直接收到資料庫的資料（以錯誤訊息或洩漏資訊的形式），也可能擷取資料庫中的資料（每次一個位元），或以惡意方式修改查詢。觀念在於，應用程式的行為（傳回的回應與原始回應相同或不同）可以提供關於所評估（已修改）查詢的單一位元資訊，也就是說，攻擊者有可能規劃使用 SQL 布林表示式，並以應用程式行為（相同/不同於原始行為）的形式來侵害其評估（單一位元）。

#### 修補建議

以下是幾種減輕風險的技巧：[1] 策略：程式庫或架構使用檢查過且不容許此弱點的程式庫或架構，或提供使其更容易避免的建構項。[2] 策略：參數化如果可行，請使用結構化機制來自動強制分開資料與程式碼。這些機制可能可以自動提供相關的引用、編碼和驗證，而不是在每次產生輸出時，都依賴開發人員來提供。[3] 策略：環境強化 使用完成必要作業所需的最低專用權，來執行程式碼。[4] 策略：輸出編碼如果儘管有風險，您還是需要使用動態產生的查詢字串或指令，適當地括住引數且避免在這些引數內使用任何特殊字元。[5] 策略：輸入驗證假設所有輸入都是惡意的。請使用「接受已知良好」輸入驗證策略：嚴格符合規格的可接受輸入白名單。拒絕未嚴格符合規格的任何輸入，或將其轉換成嚴格符合規格者。請勿只靠使用黑名單來偵測惡意或形態異常的輸入。不過，在偵測潛在攻擊或判斷哪些輸入因形態異常而應該斷然拒絕時，黑名單就很有用。以下是保護 Web 應用程式免於遭受 SQL 注入攻擊的兩種可能方式：[1] 使用儲存程序，而不用動態建置的 SQL 查詢字串。參數傳給 SQL Server 儲存程序的方式，會防止使用單引號和連字號。如何在 ASP.NET 中使用儲存程序的範例如下： ' Visual Basic example Dim DS As DataSet Dim MyConnection As SqlConnection Dim MyCommand As SqlDataAdapter Dim SelectCommand As String = "select \* from users where username = @username" ... MyCommand.SelectCommand.Parameters.Add(New SqlParameter("@username", SqlDbType.NVarChar, 20)) MyCommand.SelectCommand.Parameters("@username").Value = UserNameField.Value // C# example String selectCmd = "select \* from Authors where state = @username"; SqlConnection myConnection = new SqlConnection("server=..."); SqlDataAdapter myCommand = new SqlDataAdapter(selectCmd, myConnection); myCommand.SelectCommand.Parameters.Add(new SqlParameter("@username", SqlDbType.NVarChar, 20)); myCommand.SelectCommand.Parameters["@username"].Value = UserNameField.Value; [2] 您可以利用驗證控制項，將輸入驗證新增到「Web 表單」頁面中。驗證控制項提供適用於標準驗證之所有一般類型的簡易使用機制（例如：測試日期是否有效，或值是否在範圍內），以及用來提供自訂編寫驗證的方式。此外，驗證控制項也可讓您完整自訂向使用者顯示錯誤資訊的方式。驗證控制項可以搭配「Web 表單」頁面類別檔所處理的任何控制項來使用，其中包括 HTML 和 Web 伺服器控制項。如果要確定使用者輸入只包含有效值，您可以使用下列其中一個驗證控制項：a. "RangeValidator"：檢查使用者的輸入（值）是否在指定的上下界限之間。您可以檢查數字、英文字母和日期之配對內的範圍。b. "RegularExpressionValidator"：檢查輸入是否符合正規表示式所定義的型樣。這類型的驗證可讓您檢查可預期的字元序列，例如：社會保險號碼、電子郵件位址、電話號碼、郵遞區號等等中的字元序列。重要注意事項：驗證控制項不會封鎖使用者輸入，或變更頁面處理流程；它們只會設定錯誤狀態，以及產生錯誤訊息。在執行進一步的應用程式專屬動作之前，程式設計師負責測試程式碼中的控制項狀態。檢查使用者輸入有效性有兩種方式：1. 測試一般錯誤狀態：在您的程式碼中，測試頁面的 IsValid 內容。這個內容會累積頁面上所有驗證控制項的 IsValid 內容值（使用邏輯 AND）。如果其中一個驗證控制項設為無效，頁面內容便會傳回 False。2. 測試個別控制項的錯誤狀態：在迴圈中處理頁面的驗證器集合，集合中含有指向所有驗證控制項的參照。之後，您便可以檢查每個驗證控制項的 IsValid 內容。\*\* 備妥陳述式：保護您的應用程式免於遭受 SQL 注入（亦即，惡意竄改 SQL 參數）有三種可能的方式。請利用下列方式，而非動態建置 SQL 陳述式：[1] PreparedStatement，經過前置編譯而存放在 PreparedStatement 物件儲存區中。PreparedStatement 定義用來登錄輸入參數的設定元，輸入參數相容於支援的 JDBC SQL 資料類型。例如，setString 應該用於 VARCHAR 或 LONGVARCHAR 類型的輸入參數（請參閱 Java API，以取得進一步的詳細資料）。這個設定輸入參數的方式，可以防止攻擊者注入不正確的字元（如單引號）來操作 SQL 陳述式。如何使用 J2EE 中之 PreparedStatement 的範例： // J2EE PreparedStatemenet Example // Get a connection to the database Connection myConnection; if (isDataSourceEnabled()) { // using the DataSource to get a managed connection Context ctx = new InitialContext(); myConnection = ((DataSource)ctx.lookup(datasourceName)).getConnection(dbUserName, dbPassword); } else { try { // using the DriverManager to get a JDBC connection Class.forName(jdbcDriverClassPath); myConnection = DriverManager.getConnection(jdbcURL, dbUserName, dbPassword); } catch (ClassNotFoundException e) { ... } } ... try { PreparedStatement myStatement = myConnection.prepareStatement("select \* from users where username = ?"); myStatement.setString(1, userNameField); ResultSet rs = myStatement.executeQuery(); ... rs.close(); } catch (SQLException sqlException) { ... } finally { myStatement.close(); myConnection.close(); } [2] CallableStatement，延伸 PreparedStatement 來執行資料庫 SQL 儲存程序。這個類別繼承 PreparedStatement 的輸入設定元（請參閱上面的 [1]）。下列範例假設已建立好這個資料庫儲存程序：CREATE PROCEDURE select\_user (@username varchar(20))AS SELECT \* FROM USERS WHERE USERNAME = @username;如何利用 J2EE 中的 CallableStatement 來執行上述儲存程序的範例： // J2EE PreparedStatemenet Example // Get a connection to the database Connection myConnection; if (isDataSourceEnabled()) { // using the DataSource to get a managed connection Context ctx = new InitialContext(); myConnection = ((DataSource)ctx.lookup(datasourceName)).getConnection(dbUserName, dbPassword); } else { try { // using the DriverManager to get a JDBC connection Class.forName(jdbcDriverClassPath); myConnection = DriverManager.getConnection(jdbcURL, dbUserName, dbPassword); } catch (ClassNotFoundException e) { ... } } ... try { PreparedStatement myStatement = myConnection.prepareCall("{?= call select\_user ?,?}"); myStatement.setString(1, userNameField); myStatement.registerOutParameter(1, Types.VARCHAR); ResultSet rs = myStatement.executeQuery(); ... rs.close(); } catch (SQLException sqlException) { ... } finally { myStatement.close(); myConnection.close(); } [3] Entity Bean，在持續性儲存庫機制中代表 EJB 商業物件。Entity Bean 有兩種類型：Bean 管理和儲存器管理。當使用 Bean 管理的持續性時，開發人員負責撰寫存取資料庫的 SQL 程式碼（請參閱上述 [1] 和 [2] 區段）。當使用儲存器管理的持續性時，EJB 儲存器會自動產生 SQL 程式碼。因此，儲存器要負責防止惡意嘗試竄改產生的 SQL 程式碼。如何使用 J2EE 中之 Entity Bean 的範例： // J2EE EJB Example try { // lookup the User home interface UserHome userHome = (UserHome)context.lookup(User.class); // find the User remote interface User = userHome.findByPrimaryKey(new UserKey(userNameField)); ... } catch (Exception e) { ... } 建議的 Java 工具N/A參照https://docs.oracle.com/javase/7/docs/api/java/sql/PreparedStatement.htmlhttps://docs.oracle.com/javase/7/docs/api/java/sql/CallableStatement.html\*\* 驗證輸入資料：雖然為了使用者的方便，可以在用戶端層提供資料驗證，但必須利用 Servlet，在伺服器層執行資料驗證。用戶端驗證原本就不安全，因為它們可以輕易略過，例如：停用 JavaScript。好的設計通常需要 Web 應用程式架構提供伺服器端公用程式常式來驗證下列項目：[1] 必要欄位[2] 欄位資料類型（依預設，所有 HTTP 要求參數都是 String）[3] 欄位長度[4] 欄位範圍[5] 欄位選項[6] 欄位型樣[7] Cookie 值[8] HTTP 回應在實務中，好的做法是在 "Validator" 公用程式類別中將上述常式當作靜態方法來實作。下列各節說明範例驗證器類別[1] 必要欄位一律檢查確認欄位不是空值，欄位長度大於零，且不含在前端及尾端的空格。如何驗證必要欄位的範例： // Java example to validate required fields public Class Validator { ... public static boolean validateRequired(String value) { boolean isFieldValid = false; if (value != null && value.trim().length() > 0) { isFieldValid = true; } return isFieldValid; } ... } ... String fieldValue = request.getParameter("fieldName"); if (Validator.validateRequired(fieldValue)) { // fieldValue is valid, continue processing request ... } [2] 欄位資料類型在 Web 應用程式中，所輸入的輸入參數設定不良。例如，所有 HTTP 要求參數或 Cookie 值都是 String 類型。開發人員負責確認輸入的資料類型正確。請利用 Java 基本封套類別來檢查欄位值是否能安全轉換成所需要的基本資料類型。如何驗證數值欄位（int 類型）的範例： // Java example to validate that a field is an int number public Class Validator { ... public static boolean validateInt(String value) { boolean isFieldValid = false; try { Integer.parseInt(value); isFieldValid = true; } catch (Exception e) { isFieldValid = false; } return isFieldValid; } ... } ... // check if the HTTP request parameter is of type int String fieldValue = request.getParameter("fieldName"); if (Validator.validateInt(fieldValue)) { // fieldValue is valid, continue processing request ... } 在實務中，好的做法是將所有 HTTP 要求參數轉換成它們的相關資料類型。例如，開發人員應該將要求參數的 "integerValue" 儲存在要求屬性中，再依照下列範例所示來使用它： // Example to convert the HTTP request parameter to a primitive wrapper data type // and store this value in a request attribute for further processing String fieldValue = request.getParameter("fieldName"); if (Validator.validateInt(fieldValue)) { // convert fieldValue to an Integer Integer integerValue = Integer.getInteger(fieldValue); // store integerValue in a request attribute request.setAttribute("fieldName", integerValue); } ... // Use the request attribute for further processing Integer integerValue = (Integer)request.getAttribute("fieldName"); ... 應用程式應該處理的主要 Java 資料類型：- Byte- Short- Integer- Long- Float- Double- Date[3] 欄位長度一律確定輸入參數（HTTP 要求參數或 Cookie 值）在長度下限及/或長度上限的範圍內。驗證 userName 欄位的長度是否在 8-20 個字元之間的範例： // Example to validate the field length public Class Validator { ... public static boolean validateLength(String value, int minLength, int maxLength) { String validatedValue = value; if (!validateRequired(value)) { validatedValue = ""; } return (validatedValue.length() >= minLength && validatedValue.length() <= maxLength); } ... } ... String userName = request.getParameter("userName"); if (Validator.validateRequired(userName)) { if (Validator.validateLength(userName, 8, 20)) { // userName is valid, continue further processing ... } } [4] 欄位範圍一律確定輸入參數在函數需求定義的範圍內。驗證輸入 numberOfChoices 是否在 10-20 之間的範例： // Example to validate the field range public Class Validator { ... public static boolean validateRange(int value, int min, int max) { return (value >= min && value <= max); } ... } ... String fieldValue = request.getParameter("numberOfChoices"); if (Validator.validateRequired(fieldValue)) { if (Validator.validateInt(fieldValue)) { int numberOfChoices = Integer.parseInt(fieldValue); if (Validator.validateRange(numberOfChoices, 10, 20)) { // numberOfChoices is valid, continue processing request ... } } } [5] 欄位選項Web 應用程式通常會向使用者呈現一組可供選擇的選項（例如：使用 SELECT HTML 標籤），但無法執行伺服器端驗證來確保所選的值是容許的選項之一。請記住，惡意的使用者可以輕易修改任何選項值。請一律對照函數需求所定義容許的選項來驗證所選的使用者值。對照容許的選項清單來驗證使用者選項的範例： // Example to validate user selection against a list of options public Class Validator { ... public static boolean validateOption(Object[] options, Object value) { boolean isValidValue = false; try { List list = Arrays.asList(options); if (list != null) { isValidValue = list.contains(value); } } catch (Exception e) { } return isValidValue; } ... } ... // Allowed options String[] options = {"option1", "option2", "option3"); // Verify that the user selection is one of the allowed options String userSelection = request.getParameter("userSelection"); if (Validator.validateOption(options, userSelection)) { // valid user selection, continue processing request ... } [6] 欄位型樣一律檢查使用者輸入是否符合功能需求所定義的型樣。比方說，如果 userName 欄位只應接受英數字元，且不區分大小寫，請使用下列正規表示式：^[a-zA-Z0-9]\*$Java 1.3 或更早的版本不包括任何正規表示式套件。建議搭配 Java 1.3 來使用「Apache 正規表示式套件」（請參閱下列「資源」），以解決這項支援欠缺。執行正規表示式驗證的範例： // Example to validate that a given value matches a specified pattern // using the Apache regular expression package import org.apache.regexp.RE; import org.apache.regexp.RESyntaxException; public Class Validator { ... public static boolean matchPattern(String value, String expression) { boolean match = false; if (validateRequired(expression)) { RE r = new RE(expression); match = r.match(value); } return match; } ... } ... // Verify that the userName request parameter is alpha-numeric String userName = request.getParameter("userName"); if (Validator.matchPattern(userName, "^[a-zA-Z0-9]\*$")) { // userName is valid, continue processing request ... } Java 1.4 引進了新的正規表示式套件 (java.util.regex)。以下是 Validator.matchPattern 的修正版，使用新的 Java 1.4 正規表示式套件： // Example to validate that a given value matches a specified pattern // using the Java 1.4 regular expression package import java.util.regex.Pattern; import java.util.regexe.Matcher; public Class Validator { ... public static boolean matchPattern(String value, String expression) { boolean match = false; if (validateRequired(expression)) { match = Pattern.matches(expression, value); } return match; } ... } [7] Cookie 值請利用 javax.servlet.http.Cookie 物件來驗證 Cookie 值。依應用程式需求而定，相同的驗證規則（說明如上）也適用於 Cookie 值，例如：驗證必要的值、驗證長度，等等。驗證必要 Cookie 值的範例： // Example to validate a required cookie value // First retrieve all available cookies submitted in the HTTP request Cookie[] cookies = request.getCookies(); if (cookies != null) { // find the "user" cookie for (int i=0; i<cookies.length; ++i) { if (cookies[i].getName().equals("user")) { // validate the cookie value if (Validator.validateRequired(cookies[i].getValue()) { // valid cookie value, continue processing request ... } } } } [8] HTTP 回應[8-1] 過濾使用者輸入如果要保護應用程式免於遭受跨網站指令碼，請將機密字元轉換成對應的字元實體來消毒 HTML。以下是 HTML 區分字元：< > " ' % ; ) ( & +將機密字元轉換成對應的字元實體來過濾指定字串的範例： // Example to filter sensitive data to prevent cross-site scripting public Class Validator { ... public static String filter(String value) { if (value == null) { return null; } StringBuffer result = new StringBuffer(value.length()); for (int i=0; i<value.length(); ++i) { switch (value.charAt(i)) { case '<': result.append("<"); break; case '>': result.append(">"); break; case '"': result.append("""); break; case '\'': result.append("'"); break; case '%': result.append("%"); break; case ';': result.append(";"); break; case '(': result.append("("); break; case ')': result.append(")"); break; case '&': result.append("&"); break; case '+': result.append("+"); break; default: result.append(value.charAt(i)); break; } return result; } ... } ... // Filter the HTTP response using Validator.filter PrintWriter out = response.getWriter(); // set output response out.write(Validator.filter(response)); out.close(); Java Servlet API 2.3 引進了「過濾器」，它支援截取及轉換 HTTP 要求或回應。利用「Servlet 過濾器」，以 Validator.filter 來消毒回應的範例： // Example to filter all sensitive characters in the HTTP response using a Java Filter. // This example is for illustration purposes since it will filter all content in the response, including HTML tags! public class SensitiveCharsFilter implements Filter { ... public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException { PrintWriter out = response.getWriter(); ResponseWrapper wrapper = new ResponseWrapper((HttpServletResponse)response); chain.doFilter(request, wrapper); CharArrayWriter caw = new CharArrayWriter(); caw.write(Validator.filter(wrapper.toString())); response.setContentType("text/html"); response.setContentLength(caw.toString().length()); out.write(caw.toString()); out.close(); } ... public class CharResponseWrapper extends HttpServletResponseWrapper { private CharArrayWriter output; public String toString() { return output.toString(); } public CharResponseWrapper(HttpServletResponse response){ super(response); output = new CharArrayWriter(); } public PrintWriter getWriter(){ return new PrintWriter(output); } } } } [8-2] 維護 Cookie 安全當機密資料儲存在 Cookie 中，請務必利用 Cookie.setSecure（布林旗標），在 HTTP 回應中設定 Cookie 的安全旗標，以指示瀏覽器利用 HTTPS 或 SSL 之類的安全通訊協定來傳送 Cookie。維護 "user" Cookie 安全的範例： // Example to secure a cookie, i.e. instruct the browser to // send the cookie using a secure protocol Cookie cookie = new Cookie("user", "sensitive"); cookie.setSecure(true); response.addCookie(cookie); 建議的 Java 工具適用於伺服器端驗證的兩個主要的 Java 架構如下：[1] Jakarta 一般驗證器（整合 Struts 1.1）「Jakarta 一般驗證器」實作上述所有資料驗證需求，是一個功能強大的架構。這些規則配置在定義表單欄位輸入驗證規則的 XML 檔中。依預設，Struts 支援在利用 Struts 'bean:write' 標籤撰寫的所有資料上，過濾 [8] HTTP 回應中輸出的危險字元。設定 'filter=false' 旗標可以停用這個過濾。Struts 定義下列基本輸入驗證器，但也可以定義自訂驗證器：required：如果欄位含有空格以外的任何字元，便告成功。mask：如果值符合 mask 屬性給定的正規表示式，便告成功。range：如果值在 min 和 max 屬性給定的值範圍內（(value >= min) 且 (value <= max)），便告成功。maxLength：如果欄位長度小於或等於 max 屬性，便告成功。minLength：如果欄位長度大於或等於 min 屬性，便告成功。byte、short、integer、long、float、double：如果值可以轉換成對應的基本類型，便告成功。date：如果值代表有效日期，便告成功。可提供日期型樣。creditCard：如果值可能是有效的信用卡號碼，便告成功。e-mail：如果值可能是有效的電子郵件位址，便告成功。利用「Struts 驗證器」驗證 loginForm 之 userName 欄位的範例： <form-validation> <global> ... <validator name="required" classname="org.apache.struts.validator.FieldChecks" method="validateRequired" msg="errors.required"> </validator> <validator name="mask" classname="org.apache.struts.validator.FieldChecks" method="validateMask" msg="errors.invalid"> </validator> ... </global> <formset> <form name="loginForm"> <!-- userName is required and is alpha-numeric case insensitive --> <field property="userName" depends="required,mask"> <!-- message resource key to display if validation fails --> <msg name="mask" key="login.userName.maskmsg"/> <arg0 key="login.userName.displayname"/> <var> <var-name>mask</var-name> <var-value>^[a-zA-Z0-9]\*$</var-value> </var> </field> ... </form> ... </formset> </form-validation> [2] JavaServer Faces 技術「JavaServer Faces 技術」是一組代表使用者介面元件、管理元件狀態、處理事件及輸入驗證的 Java API (JSR 127)。JavaServer Faces API 實作下列基本驗證器，但可以定義自訂驗證器：validate\_doublerange：登錄元件的 DoubleRangeValidatorvalidate\_length：登錄元件的 LengthValidatorvalidate\_longrange：登錄元件的 LongRangeValidatorvalidate\_required：登錄元件的 RequiredValidatorvalidate\_stringrange：登錄元件的 StringRangeValidatorvalidator：登錄元件的自訂 ValidatorJavaServer Faces API 定義下列 UIInput 和 UIOutput 展現器（標籤）：input\_date：接受以 java.text.Date 實例格式化的 java.util.Dateoutput\_date：顯示以 java.text.Date 實例格式化的 java.util.Dateinput\_datetime：接受以 java.text.Time 實例格式化的 java.util.Dateoutput\_datetime：顯示以 java.text.Time 實例格式化的 java.util.Dateinput\_number：顯示以 java.text.NumberFormat 格式化的數值資料類型（java.lang.Number 或基本資料類型）output\_number：顯示以 java.text.NumberFormat 格式化的數值資料類型（java.lang.Number 或基本資料類型）input\_text：接受單行字串。output\_text：顯示單行字串。input\_time：接受以 java.text.DateFormat 時間實例格式化的 java.util.Dateoutput\_time：顯示以 java.text.DateFormat 時間實例格式化的 java.util.Dateinput\_hidden：可讓頁面作者將隱藏變數併入頁面input\_secret：接受不含空格的單行文字，輸入之時，將它顯示成一組星號input\_textarea：接受多行文字output\_errors：顯示整個頁面的錯誤訊息，或指定之用戶端 ID 的相關錯誤訊息output\_label：將巢狀元件顯示為指定輸入欄位的標籤output\_message：顯示本地化訊息利用 JavaServer Faces 驗證 loginForm 之 userName 欄位的範例： <%@ taglib uri="https://docs.oracle.com/javaee/6/tutorial/doc/glxce.html" prefix="h" %> <%@ taglib uri="http://mrbool.com/how-to-create-a-login-validation-with-jsf-java-server-faces/27046" prefix="f" %> ... <jsp:useBean id="UserBean" class="myApplication.UserBean" scope="session" /> <f:use\_faces> <h:form formName="loginForm" > <h:input\_text id="userName" size="20" modelReference="UserBean.userName"> <f:validate\_required/> <f:validate\_length minimum="8" maximum="20"/> </h:input\_text> <!-- display errors if present --> <h:output\_errors id="loginErrors" clientId="userName"/> <h:command\_button id="submit" label="Submit" commandName="submit" /><p> </h:form> </f:use\_faces> 參照Java API 1.3 - https://www.oracle.com/java/technologies/java-archive-13docs-downloads.htmlJava API 1.4 - https://www.oracle.com/java/technologies/java-archive-142docs-downloads.htmlJava Servlet API 2.3 - https://mvnrepository.com/artifact/javax.servlet/javax.servlet-apiJava 正規表示式套件 - http://jakarta.apache.org/regexp/Jakarta 驗證器 - http://jakarta.apache.org/commons/validator/JavaServer Faces 技術 - http://www.javaserverfaces.org/\*\* 錯誤處理：許多 J2EE Web 應用程式架構都遵循「模型視圖控制器 (MVC)」型樣。在這個型樣中，Servlet 扮演控制器的角色。Servlet 將應用程式處理程序委派給 JavaBean，例如，EJB Session Bean（模型）。之後，Servlet 再將要求轉遞給 JSP（視圖）來呈現處理結果。Servlet 應該檢查所有輸入、輸出、回覆碼、錯誤碼及已知的異常狀況，以確保會實際進行預期的處理程序。資料驗證可以保護應用程式免於遭受惡意的資料竄改，有效的錯誤處理策略則是防止應用程式意外揭露內部錯誤訊息（例如：異常狀況堆疊追蹤）所不可或缺。好的錯誤處理策略會處理下列項目：[1] 定義錯誤[2] 報告錯誤[3] 呈現錯誤[4] 錯誤對映[1] 定義錯誤應該避免將錯誤訊息寫在應用程式層（例如：Servlet）。相反地，應用程式應該使用對映至已知之應用程式失敗的錯誤索引鍵。在實務中，好的做法是定義錯誤索引鍵來對映至 HTML 表單欄位或其他 Bean 內容的驗證規則。比方說，如果需要 "user\_name" 欄位，它必須是英數字元，且在資料庫中必須是唯一，便應該定義下列錯誤索引鍵：(a) ERROR\_USERNAME\_REQUIRED：這個錯誤索引鍵用來顯示一則訊息，通知使用者需要 "user\_name" 欄位；(b) ERROR\_USERNAME\_ALPHANUMERIC：這個錯誤索引鍵用來顯示一則訊息，通知使用者 "user\_name" 欄位應該是英數字元；(c) ERROR\_USERNAME\_DUPLICATE：這個錯誤索引鍵用來顯示一則訊息，通知使用者 "user\_name" 值在資料庫中重複；(d) ERROR\_USERNAME\_INVALID：這個錯誤索引鍵用來顯示一則一般訊息，通知使用者 "user\_name" 值無效；在實務中，好的做法是定義下列架構 Java 類別來儲存及報告應用程式錯誤：- ErrorKeys：定義所有錯誤索引鍵 // Example: ErrorKeys defining the following error keys: // - ERROR\_USERNAME\_REQUIRED // - ERROR\_USERNAME\_ALPHANUMERIC // - ERROR\_USERNAME\_DUPLICATE // - ERROR\_USERNAME\_INVALID // ... public Class ErrorKeys { public static final String ERROR\_USERNAME\_REQUIRED = "error.username.required"; public static final String ERROR\_USERNAME\_ALPHANUMERIC = "error.username.alphanumeric"; public static final String ERROR\_USERNAME\_DUPLICATE = "error.username.duplicate"; public static final String ERROR\_USERNAME\_INVALID = "error.username.invalid"; ... } - Error：封裝個別錯誤 // Example: Error encapsulates an error key. // Error is serializable to support code executing in multiple JVMs. public Class Error implements Serializable { // Constructor given a specified error key public Error(String key) { this(key, null); } // Constructor given a specified error key and array of placeholder objects public Error(String key, Object[] values) { this.key = key; this.values = values; } // Returns the error key public String getKey() { return this.key; } // Returns the placeholder values public Object[] getValues() { return this.values; } private String key = null; private Object[] values = null; } - Errors：封裝錯誤的集合 // Example: Errors encapsulates the Error objects being reported to the presentation layer. // Errors are stored in a HashMap where the key is the bean property name and value is an // ArrayList of Error objects. public Class Errors implements Serializable { // Adds an Error object to the Collection of errors for the specified bean property. public void addError(String property, Error error) { ArrayList propertyErrors = (ArrayList)errors.get(property); if (propertyErrors == null) { propertyErrors = new ArrayList(); errors.put(property, propertyErrors); } propertyErrors.put(error); } // Returns true if there are any errors public boolean hasErrors() { return (errors.size > 0); } // Returns the Errors for the specified property public ArrayList getErrors(String property) { return (ArrayList)errors.get(property); } private HashMap errors = new HashMap(); } 以下是利用上述架構類別來處理 "user\_name" 欄位驗證錯誤的範例： // Example to process validation errors of the "user\_name" field. Errors errors = new Errors(); String userName = request.getParameter("user\_name"); // (a) Required validation rule if (!Validator.validateRequired(userName)) { errors.addError("user\_name", new Error(ErrorKeys.ERROR\_USERNAME\_REQUIRED)); } // (b) Alpha-numeric validation rule else if (!Validator.matchPattern(userName, "^[a-zA-Z0-9]\*$")) { errors.addError("user\_name", new Error(ErrorKeys.ERROR\_USERNAME\_ALPHANUMERIC)); } else { // (c) Duplicate check validation rule // We assume that there is an existing UserValidationEJB session bean that implements // a checkIfDuplicate() method to verify if the user already exists in the database. try { ... if (UserValidationEJB.checkIfDuplicate(userName)) { errors.addError("user\_name", new Error(ErrorKeys.ERROR\_USERNAME\_DUPLICATE)); } } catch (RemoteException e) { // log the error logger.error("Could not validate user for specified userName: " + userName); errors.addError("user\_name", new Error(ErrorKeys.ERROR\_USERNAME\_DUPLICATE); } } // set the errors object in a request attribute called "errors" request.setAttribute("errors", errors); ... [2] 報告錯誤報告 Web 層應用程式錯誤的方式有兩種：(a) Servlet 錯誤機制(b) JSP 錯誤機制[2-a] Servlet 錯誤機制Servlet 報告錯誤的可能方式如下：- 轉遞至輸入 JSP（已將錯誤儲存在要求屬性中），或- 利用 HTTP 錯誤碼引數來呼叫 response.sendError，或- 擲出異常狀況在實務中，好的做法是處理所有已知的應用程式錯誤（依照 [1] 區段所說明），將它們儲存在要求屬性中，再轉遞給輸入 JSP。輸入 JSP 應該顯示錯誤訊息，並提示使用者重新輸入資料。下列範例說明如何轉遞到輸入 JSP (userInput.jsp)： // Example to forward to the userInput.jsp following user validation errors RequestDispatcher rd = getServletContext().getRequestDispatcher("/user/userInput.jsp"); if (rd != null) { rd.forward(request, response); } 如果 Servlet 無法轉遞到已知的 JSP 頁面，第二個選項是利用 response.sendError 方法，設定 HttpServletResponse.SC\_INTERNAL\_SERVER\_ERROR 引數（狀態碼 500）來報告錯誤。請參閱 javax.servlet.http.HttpServletResponse javadoc，以取得各種 HTTP 狀態碼的詳細資訊。傳回 HTTP 錯誤的範例： // Example to return a HTTP error code RequestDispatcher rd = getServletContext().getRequestDispatcher("/user/userInput.jsp"); if (rd == null) { // messages is a resource bundle with all message keys and values response.sendError(HttpServletResponse.SC\_INTERNAL\_SERVER\_ERROR, messages.getMessage(ErrorKeys.ERROR\_USERNAME\_INVALID)); } 作為最後的手段，Servlet 可以擲出異常狀況，它必須是下列類別之一的子類別：- RuntimeException- ServletException- IOException[2-b] JSP 錯誤機制 JSP 頁面依照下列範例所示來定義 errorPage 指引，從而提供了執行時期異常狀況的處理機制： <%@ page errorPage="/errors/userValidation.jsp" %> 未捕捉的 JSP 異常狀況是轉遞到指定的 errorPage，而原始異常狀況則設在稱為 javax.servlet.jsp.jspException 的要求參數中。錯誤頁面必須包含 isErrorPage 指引，如下所示： <%@ page isErrorPage="true" %> isErrorPage 指引會使 "exception" 變數起始設定為所擲出的異常狀況物件。[3] 呈現錯誤J2SE Internationalization API 提供用來提出應用程式資源以及將訊息格式化的公用程式類別，其中包括：(a) 資源組(b) 訊息格式化[3-a] 資源組資源組會將本地化的資料與使用它的原始碼分開，從而支援國際化。每個資源組都會儲存特定語言環境之鍵值配對的對映。通常是利用或延伸 java.util.PropertyResourceBundle，它會將內容儲存在外部內容檔中，如下列範例所示： ################################################ # ErrorMessages.properties ################################################ # required user name error message error.username.required=User name field is required # invalid user name format error.username.alphanumeric=User name must be alphanumeric # duplicate user name error message error.username.duplicate=User name {0} already exists, please choose another one ... 定義多重資源可以支援不同的語言環境（因此稱為資源組）。例如，定義 ErrorMessages\_fr.properties 可以支援資源組系列的法國成員。如果要求之語言環境的資源成員不存在，便會使用預設成員。在上述範例中，預設資源是 ErrorMessages.properties。依使用者的語言環境而定，應用程式（JSP 或 Servlet）會從適當的資源擷取內容。[3-b] 訊息格式化J2SE 標準類別 java.util.MessageFormat 提供以取代位置保留元來建立訊息的一般方式。MessageFormat 物件包含內嵌了格式指定元的型樣字串，如下所示： // Example to show how to format a message using placeholder parameters String pattern = "User name {0} already exists, please choose another one"; String userName = request.getParameter("user\_name"); Object[] args = new Object[1]; args[0] = userName; String message = MessageFormat.format(pattern, args); 以下是利用 ResourceBundle 和 MessageFormat 來呈現錯誤訊息的更綜合性的範例： // Example to render an error message from a localized ErrorMessages resource (properties file) // Utility class to retrieve locale-specific error messages public Class ErrorMessageResource { // Returns the error message for the specified error key in the environment locale public String getErrorMessage(String errorKey) { return getErrorMessage(errorKey, defaultLocale); } // Returns the error message for the specified error key in the specified locale public String getErrorMessage(String errorKey, Locale locale) { return getErrorMessage(errorKey, null, locale); } // Returns a formatted error message for the specified error key in the specified locale public String getErrorMessage(String errorKey, Object[] args, Locale locale) { // Get localized ErrorMessageResource ResourceBundle errorMessageResource = ResourceBundle.getBundle("ErrorMessages", locale); // Get localized error message String errorMessage = errorMessageResource.getString(errorKey); if (args != null) { // Format the message using the specified placeholders args return MessageFormat.format(errorMessage, args); } else { return errorMessage; } } // default environment locale private Locale defaultLocale = Locale.getDefaultLocale(); } ... // Get the user's locale Locale userLocale = request.getLocale(); // Check if there were any validation errors Errors errors = (Errors)request.getAttribute("errors"); if (errors != null && errors.hasErrors()) { // iterate through errors and output error messages corresponding to the "user\_name" property ArrayList userNameErrors = errors.getErrors("user\_name"); ListIterator iterator = userNameErrors.iterator(); while (iterator.hasNext()) { // Get the next error object Error error = (Error)iterator.next(); String errorMessage = ErrorMessageResource.getErrorMessage(error.getKey(), userLocale); output.write(errorMessage + "\r\n"); } } 建議您定義自訂 JSP 標籤（如 displayErrors）來疊代處理及呈現錯誤訊息，如上述範例所示。[4] 錯誤對映一般而言，「Servlet 儲存器」會傳回對應於回應狀態碼或異常狀況的預設錯誤頁面。您可以利用自訂錯誤頁面來指定狀態碼或異常狀況與 Web 資源之間的對映。在實務中，好的做法是開發不揭露內部錯誤狀態的靜態錯誤頁面（依預設，大部分 Servlet 儲存器都會報告內部錯誤訊息）。這項對映是依照下列範例所指定，配置在「Web 部署描述子 (web.xml)」中： <!-- Mapping of HTTP error codes and application exceptions to error pages --> <error-page> <exception-type>UserValidationException</exception-type> <location>/errors/validationError.html</error-page> </error-page> <error-page> <error-code>500</exception-type> <location>/errors/internalError.html</error-page> </error-page> <error-page> ... </error-page> ... 建議的 Java 工具適用於伺服器端驗證的兩個主要的 Java 架構如下：[1] Jakarta 一般驗證器（整合 Struts 1.1）「Jakarta 一般驗證器」是依照上述說明來定義錯誤處理機制的 Java 架構。驗證規則配置在 XML 檔中，檔案定義了表單欄位的輸入驗證規則以及對應的驗證錯誤索引鍵。Struts 提供國際化支援，供您利用資源組和訊息格式化來建置本地化應用程式。利用「Struts 驗證器」驗證 loginForm 之 userName 欄位的範例： <form-validation> <global> ... <validator name="required" classname="org.apache.struts.validator.FieldChecks" method="validateRequired" msg="errors.required"> </validator> <validator name="mask" classname="org.apache.struts.validator.FieldChecks" method="validateMask" msg="errors.invalid"> </validator> ... </global> <formset> <form name="loginForm"> <!-- userName is required and is alpha-numeric case insensitive --> <field property="userName" depends="required,mask"> <!-- message resource key to display if validation fails --> <msg name="mask" key="login.userName.maskmsg"/> <arg0 key="login.userName.displayname"/> <var> <var-name>mask</var-name> <var-value>^[a-zA-Z0-9]\*$</var-value> </var> </field> ... </form> ... </formset> </form-validation> Struts JSP 標籤庫定義了有條件地顯示一組累計錯誤訊息的 "errors" 標籤，如下列範例所示： <%@ page language="java" %> <%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %> <%@ taglib uri="/WEB-INF/struts-bean.tld" prefix="bean" %> <html:html> <head> <body> <html:form action="/logon.do"> <table border="0" width="100%"> <tr> <th align="right"> <html:errors property="username"/> <bean:message key="prompt.username"/> </th> <td align="left"> <html:text property="username" size="16"/> </td> </tr> <tr> <td align="right"> <html:submit><bean:message key="button.submit"/></html:submit> </td> <td align="right"> <html:reset><bean:message key="button.reset"/></html:reset> </td> </tr> </table> </html:form> </body> </html:html> [2] JavaServer Faces 技術「JavaServer Faces 技術」是一組代表使用者介面元件、管理元件狀態、處理事件、驗證輸入，以及支援國際化的 Java API (JSR 127)。JavaServer Faces API 定義了 "output\_errors" UIOutput 展現器，以顯示整個頁面的錯誤訊息，或指定之用戶端 ID 的相關錯誤訊息。利用 JavaServer Faces 驗證 loginForm 之 userName 欄位的範例： <%@ taglib uri="https://docs.oracle.com/javaee/6/tutorial/doc/glxce.html" prefix="h" %> <%@ taglib uri="http://mrbool.com/how-to-create-a-login-validation-with-jsf-java-server-faces/27046" prefix="f" %> ... <jsp:useBean id="UserBean" class="myApplication.UserBean" scope="session" /> <f:use\_faces> <h:form formName="loginForm" > <h:input\_text id="userName" size="20" modelReference="UserBean.userName"> <f:validate\_required/> <f:validate\_length minimum="8" maximum="20"/> </h:input\_text> <!-- display errors if present --> <h:output\_errors id="loginErrors" clientId="userName"/> <h:command\_button id="submit" label="Submit" commandName="submit" /><p> </h:form> </f:use\_faces> 參照Java API 1.3 - https://www.oracle.com/java/technologies/java-archive-13docs-downloads.htmlJava API 1.4 - https://www.oracle.com/java/technologies/java-archive-142docs-downloads.htmlJava Servlet API 2.3 - https://mvnrepository.com/artifact/javax.servlet/javax.servlet-apiJava 正規表示式套件 - http://jakarta.apache.org/regexp/Jakarta 驗證器 - http://jakarta.apache.org/commons/validator/JavaServer Faces 技術 - http://www.javaserverfaces.org/\*\* 過濾使用者輸入將任何資料傳給 SQL 查詢之前，一律應該先利用白名單技術來適當過濾。這無論如何強調都不為過。過濾使用者輸入可讓許多注入缺失，在抵達資料庫之前便得到更正。\*\* 引用使用者輸入不論任何資料類型，只要資料庫允許，便用單引號括住所有使用者資料，始終是好的觀念。MySQL 接受這個格式化技術。\*\* 跳出資料值如果您使用 MySQL 4.3.0 或更新的版本，您應該用 mysql\_real\_escape\_string() 來跳出所有字串。如果使用舊版的 MySQL，便應該使用 mysql\_escape\_string() 函數。如果未使用 MySQL，您可以選擇使用特定資料庫的特定跳出函數。如果不知道跳出函數，您可以選擇使用較一般的跳出函數，例如，addslashes()。如果使用 PEAR DB 資料庫抽象層，您可以使用 DB::quote() 方法或使用 ? 之類的查詢位置保留元，它會自動跳出取代位置保留元的值。參照http://ca3.php.net/mysql\_real\_escape\_stringhttp://ca.php.net/mysql\_escape\_stringhttp://ca.php.net/addslasheshttp://pear.php.net/package-info.php?package=DB\*\* 驗證輸入資料：雖然為了使用者的方便，可以在用戶端層提供資料驗證，但一律必須在伺服器層執行資料驗證。用戶端驗證原本就不安全，因為它們可以輕易略過，例如：停用 JavaScript。好的設計通常需要 Web 應用程式架構提供伺服器端公用程式常式來驗證下列項目：[1] 必要欄位[2] 欄位資料類型（依預設，所有 HTTP 要求參數都是 String）[3] 欄位長度[4] 欄位範圍[5] 欄位選項[6] 欄位型樣[7] Cookie 值[8] HTTP 回應在實務中，好的做法是實作一或多個驗證各個應用程式參數的函數。下列各節說明一些檢查範例。[1] 必要欄位一律檢查確認欄位不是空值，欄位長度大於零，且不含在前端及尾端的空格。如何驗證必要欄位的範例： // PHP example to validate required fields function validateRequired($input) { ... $pass = false; if (strlen(trim($input))>0){ $pass = true; } return $pass; ... } ... if (validateRequired($fieldName)) { // fieldName is valid, continue processing request ... } [2] 欄位資料類型在 Web 應用程式中，所輸入的輸入參數設定不良。例如，所有 HTTP 要求參數或 Cookie 值都是 String 類型。開發人員負責確認輸入的資料類型正確。[3] 欄位長度一律確定輸入參數（HTTP 要求參數或 Cookie 值）在長度下限及/或長度上限的範圍內。[4] 欄位範圍一律確定輸入參數在函數需求定義的範圍內。[5] 欄位選項Web 應用程式通常會向使用者呈現一組可供選擇的選項（例如：使用 SELECT HTML 標籤），但無法執行伺服器端驗證來確保所選的值是容許的選項之一。請記住，惡意的使用者可以輕易修改任何選項值。請一律對照函數需求所定義容許的選項來驗證所選的使用者值。[6] 欄位型樣一律檢查使用者輸入是否符合功能需求所定義的型樣。比方說，如果 userName 欄位只應接受英數字元，且不區分大小寫，請使用下列正規表示式：^[a-zA-Z0-9]+$[7] Cookie 值依應用程式需求而定，相同的驗證規則（說明如上）也適用於 Cookie 值，例如：驗證必要的值、驗證長度，等等。[8] HTTP 回應[8-1] 過濾使用者輸入如果要保護應用程式免於遭受跨網站指令碼，開發人員應該將機密字元轉換成對應的字元實體來消毒 HTML。以下是 HTML 區分字元：< > " ' % ; ) ( & +PHP 包括一些 htmlentities() 之類的自動化消毒公用程式函數： $input = htmlentities($input, ENT\_QUOTES, 'UTF-8'); 此外，為了避免「跨網站指令碼」的 UTF-7 變式，您應該明確定義回應的 Content-Type 標頭，例如： <?php header('Content-Type: text/html; charset=UTF-8'); ?> [8-2] 維護 Cookie 安全當機密資料儲存在 Cookie 中，且透過 SSL 來傳輸它時，請務必先在 HTTP 回應中設定 Cookie 的安全旗標。這會指示瀏覽器只透過 SSL 連線來使用這個 Cookie。您可以利用下列程式碼範例來維護 Cookie 安全： <$php $value = "some\_value"; $time = time()+3600; $path = "/application/"; $domain = ".example.com"; $secure = 1; setcookie("CookieName", $value, $time, $path, $domain, $secure, TRUE); ?> 此外，我們建議您使用 HttpOnly 旗標。當 HttpOnly 旗標設為 TRUE 時，只能透過 HTTP 通訊協定來存取 Cookie。這表示無法用 JavaScript 之類的 Scripting 語言存取 Cookie。這個設定有助於實際減少透過 XSS 攻擊來盜用身分的情況（不過，並非所有瀏覽器都支援）。PHP 5.2.0 新增 HttpOnly 旗標。參照[1] 利用「HTTP 專用 Cookie」緩和「跨網站指令碼」： http://msdn2.microsoft.com/en-us/library/ms533046.aspx[2] PHP 安全聯盟： http://phpsec.org/[3] PHP & Web 應用程式安全部落格 (Chris Shiflett)： http://shiflett.org/

### 反映型跨網站指令碼

#### 弱點原因

有可能竊取或操作客戶階段作業和 Cookie，並可能用來假冒合法的使用者，讓駭客可以檢視或變更使用者記錄，以及用該使用者的身分來執行交易

#### 弱點說明

XSS 攻擊可以公開使用者的階段作業 Cookie，容許攻擊者強制存取使用者的階段作業以及存取使用者的帳戶，進而導致模擬使用者。攻擊者可以修改和檢視使用者記錄，以及以這些使用者的身分執行交易。攻擊者可能可以代表使用者來執行特許作業，或存取任何屬於使用者的機密資料。如果使用者具有管理員許可權，則此情況特別危險。攻擊者甚至可以在受害者的瀏覽器上執行惡意 Script，而此 Script 會將使用者重新導向至其他頁面或網站、修改內容呈現，或是甚至可能執行惡意軟體或加密採礦程式。

#### 修補建議

完整編碼來自插入至網頁之不受信任來源的所有動態資料，以確保將其視為純文字，而非可執行的 Script 或可呈現的標記。請考慮要在其中使用資料的環境定義，並且盡可能依環境定義將資料編碼為接近實際輸出：例如適用於 HTML 內容的 HTML 編碼；適用於屬性值之資料輸出的 HTML 屬性編碼；適用於動態產生之 JavaScript 的 JavaScript 編碼。例如，HTML 將非英數字元編碼為 HTML 實體時，`<` 和 `>` 將會變成 `&lt;` 和 `&gt;`。額外的防範措施是不論來源為何，都會驗證伺服器上的所有外部輸入。請依據定義資料類型、大小、範圍、格式和可接受值的嚴密正面規格（容許清單），仔細檢查每個輸入參數。在部分情況下，正規表示式或架構控制項可能十分有用，但這不是輸出編碼的取代項目。無論來源為何，都必須完成所有不受信任資料的輸出編碼和資料驗證：例如表單欄位、URL 參數、Web 服務引數、Cookie、任何來自網路的資料、環境變數、反向 DNS 查閱、查詢結果、要求標頭、URL 元件、電子郵件、檔案和檔名、資料庫，以及所有向應用程式提供資料的外部系統。請記住，可以透過 API 呼叫間接取得這類輸入。針對伺服器所傳回的每個網頁，明確地設定 `Content-Type` HTTP 回應標頭。此標頭值應該定義特定字元編碼（字集）。例如 `ISO-8859-1` 或 `UTF-8`。未指定編碼時，Web 瀏覽器可能會猜測網頁實際使用的編碼以選擇不同的編碼，但這樣容許潛在的攻擊者略過 XSS 保護。此外，請在階段作業 Cookie 上設定 `httpOnly` 旗標，防止任何 XSS 惡意探索竊取使用者的 Cookie。偏好使用可根據環境定義自動編碼所有動態輸出以防止此弱點的架構或標準程式庫，或至少提供的建構可更輕鬆地避免弱點。針對伺服器所傳回的每個網頁，明確地設定 `Content-Security-Policy` HTTP 回應標頭，讓攻擊者更難實際惡意探索 XSS 攻擊。

### 擷取毒害空值位元組 Windows 檔案

#### 弱點原因

有可能檢視 Web 伺服器上任何檔案（例如：資料庫、使用者資訊或配置檔）的內容，但受 Web 伺服器使用者的許可權限制

#### 弱點說明

攻擊者可能會修改、刪除或讀取 Web 伺服器上應用程式可存取之任何檔案（例如，資料庫、使用者資訊或配置檔）的內容。攻擊者可能嘗試毀損或移除應用程式或作業系統所使用的重要檔案進而導致 DoS 攻擊，甚至將資料新增至檔案。攻擊也可以用來刪除日誌檔案，內含可用來追蹤攻擊者的資訊。

#### 修補建議

若要防止此類型的攻擊，您應該先消毒由使用者提供且將用來建置路徑的所有資料 。此作業的最佳方式是使用容許清單。容許清單是應用程式將接受的已接受值清單。例如，由使用者提供且要插入至路徑字串之資料的容許清單將會只有可能容許底線 (\_) 和橫線 (-) 字元的英數字元。除了使用容許清單技術之外，伺服器應禁止存取檔案系統的所有區域，但特別用於 Web 應用程式的區域除外。這將防止使用者嘗試中斷 Web 應用程式目錄以及存取檔案系統之其他區域中的檔案。強制只能開啟特定副檔名移除使用者輸入中的特殊字元（中繼字元），例如垂直線 (|) 字元先標準化路徑名稱，再對其進行驗證

### 整數溢位

#### 弱點原因

有可能收集機密性除錯資訊

#### 弱點說明

發生整數溢位時，解譯的值似乎已回歸超過最大值，並重設回最小值。值可以非預期地變為零或負值。如果使用該值來控制迴圈、管理資源（例如記憶體配置）或進行商業邏輯決策，則這可能會有安全隱患。例如，交易完成時，除了客戶所購買的東西之外，整數溢位可能還會把錢給客戶。特別是，如果數學運算導致可能大於整數類型最大值的數字，則值會回歸，而且變數會設定為零或負值。i=UINT\_MAX+1; // 不帶正負號整數類型之變數的最大值 - 4294967295 (0xffffffff)。結果是：i=0

#### 修補建議

先驗證所有輸入都在預期範圍內以及正負號，再依賴其值或將它們用於算數計算。請務必檢查上下限，包括帶正負號之整數的負下限（整數溢位也可能具有極大的負數）。可能的話，請使用不帶正負號的整數。請考慮使用安全整數處理程式庫（例如 C/C++ SafeInt 或 IntegerLib）。請考慮啟用可防止部分緩衝區溢位類別的編譯器延伸。

### 預存的跨網站指令碼

#### 弱點原因

有可能竊取或操作客戶階段作業和 Cookie，並可能用來假冒合法的使用者，讓駭客可以檢視或變更使用者記錄，以及用該使用者的身分來執行交易

#### 弱點說明

XSS 攻擊可以公開使用者的階段作業 Cookie，容許攻擊者強制存取使用者的階段作業以及存取使用者的帳戶，進而導致模擬使用者。攻擊者可以修改和檢視使用者記錄，以及以這些使用者的身分執行交易。攻擊者可能可以代表使用者來執行特許作業，或存取任何屬於使用者的機密資料。如果使用者具有管理員許可權，則此情況特別危險。攻擊者甚至可以在受害者的瀏覽器上執行惡意 Script，而此 Script 會將使用者重新導向至其他頁面或網站、修改內容呈現，或是甚至可能執行惡意軟體或加密採礦程式。

#### 修補建議

完整編碼來自插入至網頁之不受信任來源的所有動態資料，以確保將其視為純文字，而非可執行的 Script 或可呈現的標記。請考慮要在其中使用資料的環境定義，並且盡可能依環境定義將資料編碼為接近實際輸出：例如適用於 HTML 內容的 HTML 編碼；適用於屬性值之資料輸出的 HTML 屬性編碼；適用於動態產生之 JavaScript 的 JavaScript 編碼。例如，HTML 將非英數字元編碼為 HTML 實體時，`<` 和 `>` 將會變成 `&lt;` 和 `&gt;`。額外的防範措施是不論來源為何，都會驗證伺服器上的所有外部輸入。請依據定義資料類型、大小、範圍、格式和可接受值的嚴密正面規格（容許清單），仔細檢查每個輸入參數。在部分情況下，正規表示式或架構控制項可能十分有用，但這不是輸出編碼的取代項目。無論來源為何，都必須完成所有不受信任資料的輸出編碼和資料驗證：例如表單欄位、URL 參數、Web 服務引數、Cookie、任何來自網路的資料、環境變數、反向 DNS 查閱、查詢結果、要求標頭、URL 元件、電子郵件、檔案和檔名、資料庫，以及所有向應用程式提供資料的外部系統。請記住，可以透過 API 呼叫間接取得這類輸入。針對伺服器所傳回的每個網頁，明確地設定 `Content-Type` HTTP 回應標頭。此標頭值應該定義特定字元編碼（字集）。例如 `ISO-8859-1` 或 `UTF-8`。未指定編碼時，Web 瀏覽器可能會猜測網頁實際使用的編碼以選擇不同的編碼，但這樣容許潛在的攻擊者略過 XSS 保護。此外，請在階段作業 Cookie 上設定 `httpOnly` 旗標，防止任何 XSS 惡意探索竊取使用者的 Cookie。偏好使用可根據環境定義自動編碼所有動態輸出以防止此弱點的架構或標準程式庫，或至少提供的建構可更輕鬆地避免弱點。針對伺服器所傳回的每個網頁，明確地設定 `Content-Security-Policy` HTTP 回應標頭，讓攻擊者更難實際惡意探索 XSS 攻擊。

### Microsoft FrontPage 目錄清單

#### 弱點原因

有可能檢視及下載某些 Web 應用程式虛擬目錄的內容，其可能包含受限的檔案

#### 弱點說明

有可能檢視及下載某些 Web 應用程式虛擬目錄的內容，其可能包含受限的檔案Web 伺服器通常配置成不允許目錄清單含有 Script 或文字內容。部分第三方產品會建立含有正式作業檔案之副本的輔助目錄。如果存取這些目錄並不受限，攻擊者便可以下載這些目錄中的檔案。Microsoft 的 FrontPage 便是會建立這類輔助目錄的產品。

#### 修補建議

請確定 Web 伺服器用戶端無法直接存取這類產品所建立的目錄（如 CVS、RCS 或 FrontPage 目錄）。請將 Web 伺服器配置成這些目錄都不可存取（所有一般 Web 伺服器都有拒絕存取特定目錄的選項）。

### SameSite 屬性不安全、不適當或遺漏的 Cookie

#### 弱點原因

將 Cookie 限制為第一方或相同網站環境定義以預防 Cookie 資訊洩漏。如果沒有設置額外的保護措施（如反 CSRF 記號），攻擊者可以延伸為偽造跨網站要求 (CSRF) 攻擊。

#### 弱點說明

將 Cookie 限制為第一方或相同網站環境定義，藉此預防 Cookie 資訊洩漏。如果沒有設置額外的保護措施（如反 CSRF 記號），攻擊者可以延伸為偽造跨網站要求 (CSRF) 攻擊。SameSite 屬性控制跨網域要求的 Cookie 傳送方式。屬性的值有三個：「Lax」、「Strict」或「None」。如果您使用「None」，網站可以建立與其他網站之間的跨網域 POST HTTP 要求，而瀏覽器會自動將 Cookie 新增到該要求中。如果沒有設置額外的保護措施（如反 CSRF 記號），可能會引發偽造跨網站要求 (CSRF) 攻擊。模式與其用法：「Lax」模式：Cookie 只會連同最上層 GET 要求一同傳送。「Strict」模式：即使使用者遵循其他網站的鏈結，Cookie 也不會連同任何跨網站用法一同傳送。「None」模式：Cookie 將連同跨網站要求一同傳送。擁有「Lax」或「None」的屬性必須設定「Secure」旗標，而且必須透過 https 傳輸。範例 - Set-Cookie: key=value; SameSite=Lax;Secure建議選項是將屬性設定為「Strict」。範例 - Set-Cookie: key=value; SameSite=Strict

#### 修補建議

[1] 檢閱將 SameSite Cookie 屬性配置為建議值的可能解決方案。[2] 將 Cookie 限制為第一方或相同網站環境定義。[3] 將 Cookie 的 SameSite 屬性設定為 Strict 並加以驗證，確保 Cookie 只能在第一方環境定義中傳送。[4] 或者，如果您想要放鬆第一方環境定義的限制，請將 Cookie 的 SameSite 屬性設定為 Lax 並啟用 Secure 旗標，再透過 HTTPS 傳輸，同時加以驗證。

### Windows 檔案參數變更

#### 弱點原因

有可能檢視 Web 伺服器上任何檔案（例如：資料庫、使用者資訊或配置檔）的內容，但受 Web 伺服器使用者的許可權限制

#### 弱點說明

有可能檢視 Web 伺服器上任何檔案（例如：資料庫、使用者資訊或配置檔）的內容，但受 Web 伺服器使用者的許可權限制CGI Script 通常含有一些參數，用來指定顯示的檔案或作為範本的檔案。如果應用程式未驗證提供給 Script 的檔名，攻擊者就可以操作這個參數，然後要求在伺服器上的其他檔案。範例：[原始 HTML 表單] <FORM METHOD=POST ACTION="/cgi-bin/vulnerable\_script.cgi"> ... <INPUT TYPE=HIDDEN NAME="template" VALUE="/dir1/dir2/template.txt"> ... </FORM> [操作的 HTML 表單] <FORM METHOD=POST ACTION="http://target/cgi-bin/vulnerable\_script.cgi"> ... <INPUT TYPE=HIDDEN NAME="template" VALUE="../../../../../boot.ini"> ... </FORM> 如此一來，當應用程式將表單提交回伺服器時，會向攻擊者提供 boot.ini 檔。附註：這項攻擊有可能改為利用 "\"，或與 "/" 交換，以形成變式。因此，可能使用 ../../../../../boot.ini, ..\..\..\..\..\boot.ini 和 ..\../../..\../boot.ini。

#### 修補建議

[1] 確定要求的檔案在 Web 伺服器的虛擬路徑中。[2] 確定只能開啟特定副檔名[3] 從使用者輸入中移除特殊字元（meta 字元），例如：垂直線 ("|") 字元。[4] 在 Perl CGI Script 中，檔案使用「明確開啟」模式。ASP.NET 提供了一些先驗證檔名再開啟檔案的方法。例如：[1] Server.MapPath() 方法：這個方法會接收一個路徑（字串），再將指定的相對或虛擬路徑對映到伺服器上相對應的實體目錄。附註：AspEnableParentPaths 內容（Metabase 內容）的預設值是 FALSE，以確保 Script 不會有應用程式根目錄以外的檔案存取權。將這個內容值改成 TRUE 會構成潛在的安全風險。[2] Path.GetFileName() 方法：這個方法會有效移除給定檔案路徑的最後一個元素，傳回由檔案路徑中最後一個分隔字元之前的所有字元所組成的字串（不包括最後一個分隔字元）。附註：如果這個方法的引數包括如下 'InvalidPathChars'，便會發出 'ArgumentException'：a. 引號 (")b. 小於 (<)c. 大於 (>)d. 垂直線 (|)e. 倒退鍵 (\b)f. 空值 (\0)g. Unicode 字元 16 至 18 和 20 至 25。基於安全考量，我們建議您先使用 Server.MapPath() 方法，再執行檔名的任何輸入驗證。\*\* 檔案路徑驗證：您可以利用多種方法從「Servlet 儲存器」存取檔案系統。不過，部分方法有危險，因為它們支援在 Web 根目錄之外存取檔案。給定虛擬路徑在伺服器的文件樹狀結構中，有兩種安全的 Servlet API 方法可用來存取 Web 資源。當給定檔名解析為在 Web 根目錄之外的值時，這些 API 會傳回空值。應該利用下列 API 來存取儲存在 Web 根目錄之下的配置檔或其他檔案：[1] ServletContext.getResource（或 ServletContext.getResourceAsStream）[2] ServletContext.getRealPath[1] ServletContext.getResource（或 ServletContext.getResourceAsStream）ServletContext.getResource 或 ServletContext.getResourceAsStream 可用來存取位於虛擬路徑（相對於伺服器的 Web 根目錄）的資源。資源可以在本端或遠端檔案系統上，在資料庫中，或在 .war 檔中。ServletContext.getResource 會傳回對映至指定路徑之資源的 URL。如果沒有資源對映至指定的路徑，它會傳回空值。如果指定路徑的形式不正確，它會擲出 MalFormedURLException。ServletContext.getResourceAsStream 會傳回對映至指定路徑之資源的 InputStream 物件。如果沒有有效資源對映至指定的路徑，它會傳回空值。下列範例說明如何載入在應用程式 WEB-INF 目錄中，名稱為 servlet.xml 的 Servlet 配置檔： // Example to load the /WEB-INF/servlet.xml configuration file URL url = getServletContext().getResource("/WEB-INF/servlet.xml"); // Acquire an input stream to the config resource InputStream configInput = url.openStream(); ... 另外，您也可以依照下列範例所說明，利用 ServletContext.getResourceAsStream 方法來直接獲得配置檔的輸入串流： // Example to acquire an input stream to a resource InputStream configInput = getServletContext().getResourceAsStream("/WEB-INF/servlet.xml"); ... [2] ServletContext.getRealPath給定資源的虛擬路徑（相對於伺服器的 Web 根目錄），這個方法會傳回資源的真實路徑。真實路徑以執行 Servlet 儲存器的電腦和作業系統所適用的形式傳回，其中包括適當的路徑分隔字元。這個方法的通行程度低於 ServletContext.getResource（或 ServletContext.getResourceAsStream），因為它不允許存取未儲存在本端檔案的資源。如果 Servlet 儲存器為了任何原因（例如：從 .war 保存檔啟用內容），無法將虛擬路徑轉換成真實路徑，它會傳回空值。請注意，Servlet 引擎負責實作 ServletContext.getRealPath 方法。在所有 Servlet 引擎中，這項實作有可能不一致或不正確。請一律依照下列範例所說明來驗證傳回的路徑： // Example to access a resource using ServletContext.getRealPath // Get the virtual path parameter from the http request String virtualPath = request.getParameter("virtual\_path"); String realPath = getServletContext().getRealPath(virtualPath); if (realPath != null) { // verify that realPath is valid File file = new File(realPath); if (!file.exists()) { // oops, invalid path ... } } 建議的 Java 工具N/A.參照https://www.programcreek.com/java-api-examples/?class=javax.servlet.ServletContext&method=getResourcehttps://www.programcreek.com/java-api-examples/?class=javax.servlet.ServletContext&method=getResourceAsStreamhttps://www.programcreek.com/java-api-examples/?class=javax.servlet.ServletContext&method=getRealPath\*\*驗證輸入資料：雖然為了使用者的方便，可以在用戶端層提供資料驗證，但必須利用 Servlet，在伺服器層執行資料驗證。用戶端驗證原本就不安全，因為它們可以輕易略過，例如：停用 JavaScript。好的設計通常需要 Web 應用程式架構提供伺服器端公用程式常式來驗證下列項目：[1] 必要欄位[2] 欄位資料類型（依預設，所有 HTTP 要求參數都是 String）[3] 欄位長度[4] 欄位範圍[5] 欄位選項[6] 欄位型樣[7] Cookie 值[8] HTTP 回應在實務中，好的做法是在 "Validator" 公用程式類別中將上述常式當作靜態方法來實作。下列各節說明範例驗證器類別[1] 必要欄位一律檢查確認欄位不是空值，欄位長度大於零，且不含在前端及尾端的空格。如何驗證必要欄位的範例： // Java example to validate required fields public Class Validator { ... public static boolean validateRequired(String value) { boolean isFieldValid = false; if (value != null && value.trim().length() > 0) { isFieldValid = true; } return isFieldValid; } ... } ... String fieldValue = request.getParameter("fieldName"); if (Validator.validateRequired(fieldValue)) { // fieldValue is valid, continue processing request ... } [2] 欄位資料類型在 Web 應用程式中，所輸入的輸入參數設定不良。例如，所有 HTTP 要求參數或 Cookie 值都是 String 類型。開發人員負責確認輸入的資料類型正確。請利用 Java 基本封套類別來檢查欄位值是否能安全轉換成所需要的基本資料類型。如何驗證數值欄位（int 類型）的範例： // Java example to validate that a field is an int number public Class Validator { ... public static boolean validateInt(String value) { boolean isFieldValid = false; try { Integer.parseInt(value); isFieldValid = true; } catch (Exception e) { isFieldValid = false; } return isFieldValid; } ... } ... // check if the HTTP request parameter is of type int String fieldValue = request.getParameter("fieldName"); if (Validator.validateInt(fieldValue)) { // fieldValue is valid, continue processing request ... } 在實務中，好的做法是將所有 HTTP 要求參數轉換成它們的相關資料類型。請將要求參數的 "integerValue" 儲存在要求屬性中，再依照下列範例所示來使用它： // Example to convert the HTTP request parameter to a primitive wrapper data type // and store this value in a request attribute for further processing String fieldValue = request.getParameter("fieldName"); if (Validator.validateInt(fieldValue)) { // convert fieldValue to an Integer Integer integerValue = Integer.getInteger(fieldValue); // store integerValue in a request attribute request.setAttribute("fieldName", integerValue); } ... // Use the request attribute for further processing Integer integerValue = (Integer)request.getAttribute("fieldName"); ... 應用程式應該處理的主要 Java 資料類型：- Byte- Short- Integer- Long- Float- Double- Date[3] 欄位長度一律確定輸入參數（HTTP 要求參數或 Cookie 值）在長度下限及/或長度上限的範圍內。驗證 userName 欄位的長度是否在 8-20 個字元之間的範例： // Example to validate the field length public Class Validator { ... public static boolean validateLength(String value, int minLength, int maxLength) { String validatedValue = value; if (!validateRequired(value)) { validatedValue = ""; } return (validatedValue.length() >= minLength && validatedValue.length() <= maxLength); } ... } ... String userName = request.getParameter("userName"); if (Validator.validateRequired(userName)) { if (Validator.validateLength(userName, 8, 20)) { // userName is valid, continue further processing ... } } [4] 欄位範圍一律確定輸入參數在函數需求定義的範圍內。驗證輸入 numberOfChoices 是否在 10-20 之間的範例： // Example to validate the field range public Class Validator { ... public static boolean validateRange(int value, int min, int max) { return (value >= min && value <= max); } ... } ... String fieldValue = request.getParameter("numberOfChoices"); if (Validator.validateRequired(fieldValue)) { if (Validator.validateInt(fieldValue)) { int numberOfChoices = Integer.parseInt(fieldValue); if (Validator.validateRange(numberOfChoices, 10, 20)) { // numberOfChoices is valid, continue processing request ... } } } [5] 欄位選項Web 應用程式通常會向使用者呈現一組可供選擇的選項（例如：使用 SELECT HTML 標籤），但無法執行伺服器端驗證來確保所選的值是容許的選項之一。請記住，惡意的使用者可以輕易修改任何選項值。請一律對照函數需求所定義容許的選項來驗證所選的使用者值。對照容許的選項清單來驗證使用者選項的範例： // Example to validate user selection against a list of options public Class Validator { ... public static boolean validateOption(Object[] options, Object value) { boolean isValidValue = false; try { List list = Arrays.asList(options); if (list != null) { isValidValue = list.contains(value); } } catch (Exception e) { } return isValidValue; } ... } ... // Allowed options String[] options = {"option1", "option2", "option3"); // Verify that the user selection is one of the allowed options String userSelection = request.getParameter("userSelection"); if (Validator.validateOption(options, userSelection)) { // valid user selection, continue processing request ... } [6] 欄位型樣一律檢查使用者輸入是否符合功能需求所定義的型樣。比方說，如果 userName 欄位只應接受英數字元，且不區分大小寫，請使用下列正規表示式：^[a-zA-Z0-9]\*$Java 1.3 或更早的版本不包括任何正規表示式套件。建議搭配 Java 1.3 來使用「Apache 正規表示式套件」（請參閱下列「資源」），以解決這項支援欠缺。執行正規表示式驗證的範例： // Example to validate that a given value matches a specified pattern // using the Apache regular expression package import org.apache.regexp.RE; import org.apache.regexp.RESyntaxException; public Class Validator { ... public static boolean matchPattern(String value, String expression) { boolean match = false; if (validateRequired(expression)) { RE r = new RE(expression); match = r.match(value); } return match; } ... } ... // Verify that the userName request parameter is alpha-numeric String userName = request.getParameter("userName"); if (Validator.matchPattern(userName, "^[a-zA-Z0-9]\*$")) { // userName is valid, continue processing request ... } Java 1.4 引進了新的正規表示式套件 (java.util.regex)。以下是 Validator.matchPattern 的修正版，使用新的 Java 1.4 正規表示式套件： // Example to validate that a given value matches a specified pattern // using the Java 1.4 regular expression package import java.util.regex.Pattern; import java.util.regexe.Matcher; public Class Validator { ... public static boolean matchPattern(String value, String expression) { boolean match = false; if (validateRequired(expression)) { match = Pattern.matches(expression, value); } return match; } ... } [7] Cookie 值請利用 javax.servlet.http.Cookie 物件來驗證 Cookie 值。依應用程式需求而定，相同的驗證規則（說明如上）也適用於 Cookie 值，例如：驗證必要的值、驗證長度，等等。驗證必要 Cookie 值的範例： // Example to validate a required cookie value // First retrieve all available cookies submitted in the HTTP request Cookie[] cookies = request.getCookies(); if (cookies != null) { // find the "user" cookie for (int i=0; i<cookies.length; ++i) { if (cookies[i].getName().equals("user")) { // validate the cookie value if (Validator.validateRequired(cookies[i].getValue()) { // valid cookie value, continue processing request ... } } } } [8] HTTP 回應[8-1] 過濾使用者輸入如果要保護應用程式免於遭受跨網站指令碼，請將機密字元轉換成對應的字元實體來消毒 HTML。以下是 HTML 區分字元：< > " ' % ; ) ( & +將機密字元轉換成對應的字元實體來過濾指定字串的範例： // Example to filter sensitive data to prevent cross-site scripting public Class Validator { ... public static String filter(String value) { if (value == null) { return null; } StringBuffer result = new StringBuffer(value.length()); for (int i=0; i<value.length(); ++i) { switch (value.charAt(i)) { case '<': result.append("<"); break; case '>': result.append(">"); break; case '"': result.append("""); break; case '\'': result.append("'"); break; case '%': result.append("%"); break; case ';': result.append(";"); break; case '(': result.append("("); break; case ')': result.append(")"); break; case '&': result.append("&"); break; case '+': result.append("+"); break; default: result.append(value.charAt(i)); break; } return result; } ... } ... // Filter the HTTP response using Validator.filter PrintWriter out = response.getWriter(); // set output response out.write(Validator.filter(response)); out.close(); Java Servlet API 2.3 引進了「過濾器」，它支援截取及轉換 HTTP 要求或回應。利用「Servlet 過濾器」，以 Validator.filter 來消毒回應的範例： // Example to filter all sensitive characters in the HTTP response using a Java Filter. // This example is for illustration purposes since it will filter all content in the response, including HTML tags! public class SensitiveCharsFilter implements Filter { ... public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException { PrintWriter out = response.getWriter(); ResponseWrapper wrapper = new ResponseWrapper((HttpServletResponse)response); chain.doFilter(request, wrapper); CharArrayWriter caw = new CharArrayWriter(); caw.write(Validator.filter(wrapper.toString())); response.setContentType("text/html"); response.setContentLength(caw.toString().length()); out.write(caw.toString()); out.close(); } ... public class CharResponseWrapper extends HttpServletResponseWrapper { private CharArrayWriter output; public String toString() { return output.toString(); } public CharResponseWrapper(HttpServletResponse response){ super(response); output = new CharArrayWriter(); } public PrintWriter getWriter(){ return new PrintWriter(output); } } } } [8-2] 維護 Cookie 安全當機密資料儲存在 Cookie 中，請務必利用 Cookie.setSecure（布林旗標），在 HTTP 回應中設定 Cookie 的安全旗標，以指示瀏覽器利用 HTTPS 或 SSL 之類的安全通訊協定來傳送 Cookie。維護 "user" Cookie 安全的範例： // Example to secure a cookie, i.e. instruct the browser to // send the cookie using a secure protocol Cookie cookie = new Cookie("user", "sensitive"); cookie.setSecure(true); response.addCookie(cookie); 建議的 Java 工具適用於伺服器端驗證的兩個主要的 Java 架構如下：[1] Jakarta 一般驗證器（整合 Struts 1.1）「Jakarta 一般驗證器」實作上述所有資料驗證需求，是一個功能強大的架構。這些規則配置在定義表單欄位輸入驗證規則的 XML 檔中。依預設，Struts 支援在利用 Struts 'bean:write' 標籤撰寫的所有資料上，過濾 [8] HTTP 回應中輸出的危險字元。設定 'filter=false' 旗標可以停用這個過濾。Struts 定義下列基本輸入驗證器，但也可以定義自訂驗證器：required：如果欄位含有空格以外的任何字元，便告成功。mask：如果值符合 mask 屬性給定的正規表示式，便告成功。range：如果值在 min 和 max 屬性給定的值範圍內（(value >= min) 且 (value <= max)），便告成功。maxLength：如果欄位長度小於或等於 max 屬性，便告成功。minLength：如果欄位長度大於或等於 min 屬性，便告成功。byte、short、integer、long、float、double：如果值可以轉換成對應的基本類型，便告成功。date：如果值代表有效日期，便告成功。可提供日期型樣。creditCard：如果值可能是有效的信用卡號碼，便告成功。e-mail：如果值可能是有效的電子郵件位址，便告成功。利用「Struts 驗證器」驗證 loginForm 之 userName 欄位的範例： <form-validation> <global> ... <validator name="required" classname="org.apache.struts.validator.FieldChecks" method="validateRequired" msg="errors.required"> </validator> <validator name="mask" classname="org.apache.struts.validator.FieldChecks" method="validateMask" msg="errors.invalid"> </validator> ... </global> <formset> <form name="loginForm"> <!-- userName is required and is alpha-numeric case insensitive --> <field property="userName" depends="required,mask"> <!-- message resource key to display if validation fails --> <msg name="mask" key="login.userName.maskmsg"/> <arg0 key="login.userName.displayname"/> <var> <var-name>mask</var-name> <var-value>^[a-zA-Z0-9]\*$</var-value> </var> </field> ... </form> ... </formset> </form-validation> [2] JavaServer Faces 技術「JavaServer Faces 技術」是一組代表使用者介面元件、管理元件狀態、處理事件及驗證輸入的 Java API (JSR 127)。JavaServer Faces API 實作下列基本驗證器，但可以定義自訂驗證器：validate\_doublerange：登錄元件的 DoubleRangeValidator。validate\_length：登錄元件的 LengthValidator。validate\_longrange：登錄元件的 LongRangeValidator。validate\_required：登錄元件的 RequiredValidator。validate\_stringrange：登錄元件的 StringRangeValidator。validator：登錄元件的自訂 Validator。JavaServer Faces API 定義下列 UIInput 和 UIOutput 展現器（標籤）：input\_date：接受以 java.text.Date 實例格式化的 java.util.Date。output\_date：顯示以 java.text.Date 實例格式化的 java.util.Date。input\_datetime：接受以 java.text.Time 實例格式化的 java.util.Date。output\_datetime：顯示以 java.text.Time 實例格式化的 java.util.Date。input\_number：顯示以 java.text.NumberFormat 格式化的數值資料類型（java.lang.Number 或基本資料類型）。output\_number：顯示以 java.text.NumberFormat 格式化的數值資料類型（java.lang.Number 或基本資料類型）。input\_text：接受單行字串。output\_text：顯示單行字串。input\_time：接受以 java.text.DateFormat 時間實例格式化的 java.util.Date。output\_time：顯示以 java.text.DateFormat 時間實例格式化的 java.util.Date。input\_hidden：可讓頁面作者將隱藏變數併入頁面。input\_secret：接受不含空格的單行文字，並在輸入時，將它顯示成一組星號。input\_textarea：接受多行文字。output\_errors：顯示整個頁面的錯誤訊息，或指定之用戶端 ID 的相關錯誤訊息。output\_label：將巢狀元件顯示為指定輸入欄位的標籤。output\_message：顯示本地化訊息。利用 JavaServer Faces 驗證 loginForm 之 userName 欄位的範例： <%@ taglib uri="https://docs.oracle.com/javaee/6/tutorial/doc/glxce.html" prefix="h" %> <%@ taglib uri="http://mrbool.com/how-to-create-a-login-validation-with-jsf-java-server-faces/27046" prefix="f" %> ... <jsp:useBean id="UserBean" class="myApplication.UserBean" scope="session" /> <f:use\_faces> <h:form formName="loginForm" > <h:input\_text id="userName" size="20" modelReference="UserBean.userName"> <f:validate\_required/> <f:validate\_length minimum="8" maximum="20"/> </h:input\_text> <!-- display errors if present --> <h:output\_errors id="loginErrors" clientId="userName"/> <h:command\_button id="submit" label="Submit" commandName="submit" /><p> </h:form> </f:use\_faces> 參照Java API 1.3 - https://www.oracle.com/java/technologies/java-archive-13docs-downloads.htmlJava API 1.4 - https://www.oracle.com/java/technologies/java-archive-142docs-downloads.htmlJava Servlet API 2.3 - https://mvnrepository.com/artifact/javax.servlet/javax.servlet-apiJava 正規表示式套件 - http://jakarta.apache.org/regexp/Jakarta 驗證器 - http://jakarta.apache.org/commons/validator/JavaServer Faces 技術 - http://www.javaserverfaces.org/\*\* 檔案路徑驗證：PHP 提供了一些先驗證檔名再開啟檔案的方法。例如：[1] file\_exists() 方法：這個方法檢查檔案或目錄是否存在。如果檔案或目錄存在，它會傳回 True，否則，傳回 False。 <?php $filename = '/path/to/foo.txt'; if (file\_exists($filename)) { echo "The file $filename exists"; } else { echo "The file $filename does not exist"; } ?> [2] is\_file() 方法：這個方法指出檔名是否為一般檔案，若是，便傳回 True。 <?php $filename = '/path/to/foo.txt'; if (is\_file($filename)) { echo "The file $filename is a regular file"; } else { echo "There is a problem with the file: $filename"; } ?> [3] 當執行含有使用者輸入的檔案作業時，您應該一律檢查指定的檔名和路徑是否符合特定限制。具體地說，您應該確定只允許在預定的目錄樹狀結構內執行檔案作業。檢查給定的路徑名稱是否開始於所要的目錄名稱並不足夠， 因為攻擊者可以利用路徑遍訪攻擊來欺騙應用程式，讓它在所要的目錄之外提供檔案；比方說，利用 "../../../../"（附註：這個字串也可以用許多不同形式來編碼，以略過字串相符演算法）。以下是試圖驗證使用者提供的檔名時，可能會有用的兩個 PHP 函數：realpath() - 傳回標準化絕對路徑名稱basename() - 傳回路徑的檔名元件參照[1] PHP 手冊 - realpath(): http://www.php.net/realpath[2] PHP 手冊 - basename(): http://www.php.net/basename[3] PHP 手冊 - file\_exists(): http://www.php.net/file\_exists[4] PHP 手冊 - is\_file(): http://www.php.net/is\_file\*\*檔案路徑驗證：PHP 提供了一些先驗證檔名再開啟檔案的方法。例如：[1] file\_exists() 方法：這個方法檢查檔案或目錄是否存在。如果檔案或目錄存在，它會傳回 True，否則，傳回 False。 <?php $filename = '/path/to/foo.txt'; if (file\_exists($filename)) { echo "The file $filename exists"; } else { echo "The file $filename does not exist"; } ?> [2] is\_file() 方法：這個方法指出檔名是否為一般檔案，若是，便傳回 True。 <?php $filename = '/path/to/foo.txt'; if (is\_file($filename)) { echo "The file $filename is a regular file"; } else { echo "There is a problem with the file: $filename"; } ?> [3] 當執行含有使用者輸入的檔案作業時，您應該一律檢查指定的檔名和路徑是否符合特定限制。具體地說，您應該確定只允許在預定的目錄樹狀結構內執行檔案作業。檢查給定的路徑名稱是否開始於所要的目錄名稱並不足夠， 因為攻擊者可以利用路徑遍訪攻擊來欺騙應用程式，讓它在所要的目錄之外提供檔案；比方說，利用 "../../../../"（附註：這個字串也可以用許多不同形式來編碼，以略過字串相符演算法）。以下是試圖驗證使用者提供的檔名時，可能會有用的兩個 PHP 函數：realpath() - 傳回標準化絕對路徑名稱basename() - 傳回路徑的檔名元件參照[1] PHP 手冊 - realpath(): http://www.php.net/realpath[2] PHP 手冊 - basename(): http://www.php.net/basename[3] PHP 手冊 - file\_exists(): http://www.php.net/file\_exists[4] PHP 手冊 - is\_file(): http://www.php.net/is\_file\*\* 驗證輸入資料：雖然為了使用者的方便，可以在用戶端層提供資料驗證，但一律必須在伺服器層執行資料驗證。用戶端驗證原本就不安全，因為它們可以輕易略過，例如：停用 JavaScript。好的設計通常需要 Web 應用程式架構提供伺服器端公用程式常式來驗證下列項目：[1] 必要欄位[2] 欄位資料類型（依預設，所有 HTTP 要求參數都是 String）[3] 欄位長度[4] 欄位範圍[5] 欄位選項[6] 欄位型樣[7] Cookie 值[8] HTTP 回應在實務中，好的做法是實作一或多個驗證各個應用程式參數的函數。下列各節說明一些檢查範例。[1] 必要欄位一律檢查確認欄位不是空值，欄位長度大於零，且不含在前端及尾端的空格。如何驗證必要欄位的範例： // PHP example to validate required fields function validateRequired($input) { ... $pass = false; if (strlen(trim($input))>0){ $pass = true; } return $pass; ... } ... if (validateRequired($fieldName)) { // fieldName is valid, continue processing request ... } [2] 欄位資料類型在 Web 應用程式中，所輸入的輸入參數設定不良。例如，所有 HTTP 要求參數或 Cookie 值都是 String 類型。開發人員負責確認輸入的資料類型正確。[3] 欄位長度一律確定輸入參數（HTTP 要求參數或 Cookie 值）在長度下限及/或長度上限的範圍內。[4] 欄位範圍一律確定輸入參數在函數需求定義的範圍內。[5] 欄位選項Web 應用程式通常會向使用者呈現一組可供選擇的選項（例如：使用 SELECT HTML 標籤），但無法執行伺服器端驗證來確保所選的值是容許的選項之一。請記住，惡意的使用者可以輕易修改任何選項值。請一律對照函數需求所定義容許的選項來驗證所選的使用者值。[6] 欄位型樣一律檢查使用者輸入是否符合功能需求所定義的型樣。比方說，如果 userName 欄位只應接受英數字元，且不區分大小寫，請使用下列正規表示式：^[a-zA-Z0-9]+$[7] Cookie 值依應用程式需求而定，相同的驗證規則（說明如上）也適用於 Cookie 值，例如：驗證必要的值、驗證長度，等等。[8] HTTP 回應[8-1] 過濾使用者輸入如果要保護應用程式免於遭受跨網站指令碼，開發人員應該將機密字元轉換成對應的字元實體來消毒 HTML。以下是 HTML 區分字元：< > " ' % ; ) ( & +PHP 包括一些 htmlentities() 之類的自動化消毒公用程式函數： $input = htmlentities($input, ENT\_QUOTES, 'UTF-8'); 此外，為了避免「跨網站指令碼」的 UTF-7 變式，您應該明確定義回應的 Content-Type 標頭，例如： <?php header('Content-Type: text/html; charset=UTF-8'); ?> [8-2] 維護 Cookie 安全當機密資料儲存在 Cookie 中，且透過 SSL 來傳輸它時，請務必先在 HTTP 回應中設定 Cookie 的安全旗標。這會指示瀏覽器只透過 SSL 連線來使用這個 Cookie。您可以利用下列程式碼範例來維護 Cookie 安全： <$php $value = "some\_value"; $time = time()+3600; $path = "/application/"; $domain = ".example.com"; $secure = 1; setcookie("CookieName", $value, $time, $path, $domain, $secure, TRUE); ?> 此外，我們建議您使用 HttpOnly 旗標。當 HttpOnly 旗標設為 TRUE 時，只能透過 HTTP 通訊協定來存取 Cookie。這表示無法用 JavaScript 之類的 Scripting 語言存取 Cookie。這個設定有助於實際減少透過 XSS 攻擊來盜用身分的情況（不過，並非所有瀏覽器都支援）。PHP 5.2.0 新增 HttpOnly 旗標。參照[1] 利用「HTTP 專用 Cookie」緩和「跨網站指令碼」： http://msdn2.microsoft.com/en-us/library/ms533046.aspx[2] PHP 安全聯盟： http://phpsec.org/[3] PHP & Web 應用程式安全部落格 (Chris Shiflett)： http://shiflett.org/

### 有弱點的元件

#### 弱點原因

攻擊者有可能使用 Web 伺服器來攻擊其他網站，讓其身分更加隱密

#### 弱點說明

有弱點的元件可能導致應用程式出現各種弱點

#### 修補建議

升級到該組件的最新版本。如果該組件已達到其生命終結（EOL），請用最近支持的替代品替換該組件。我們強烈建議聯繫該產品的供應商，看看是否最近有補丁或修復可用。