

安全测试概念

一、什么是安全性测试

安全性测试(security testing)是有关验证应用程序的安全服务和识别潜在安全性**缺陷**的过程。

注意: 安全性测试并不最终证明应用程序是安全的, 而是用于验证所设立策略的有效性, 这些对策是基于威胁分析阶段所做的假设而选择的。

WEB 安全性测试

一个完整的 WEB 安全性测试可以从部署与基础结构、输入验证、身份验证、授权、**配置管理**、敏感数据、会话管理、加密。参数操作、异常管理、审核和日志记录等几个方面入手。

1. 安全体系测试

1) 部署与基础结构

1 **网络**是否提供了安全的通信

1 部署拓扑结构是否包括内部的防火墙

1 部署拓扑结构中是否包括远程应用程序**服务器**

1 基础结构安全性**需求**的限制是什么

1 目标环境支持怎样的信任级别

2) 输入验证

1 如何验证输入

A. 是否清楚入口点

- B. 是否清楚信任边界
- C. 是否验证 Web 页输入
- D. 是否对传递到组件或 Web 服务的参数进行验证
- E. 是否验证从**数据库**中检索的数据
- F. 是否将方法集中起来
- G. 是否依赖客户端的验证
- H. 应用程序是否易受 SQL 注入攻击
- I. 应用程序是否易受 XSS 攻击

Web 应用的安全性测试入门

介绍

随着越来越多的重要数据都存储在 **web** 应用上，以及网络事务数量的增长，适当的基于网络应用程序的安全性**测试**也变得相当重要。安全性测试是指机密的数据确保其机密性（例如，不是将其暴露给不恰当的不被授权的个人或用户实体）以及用户只能在其被授权的范围进行操作（例如，一个用户不应该能够单方面有权限屏蔽掉网站的某一功能，一个用户不应该能够在某种无心的状态下改变网络应用程序的功能）的这样一个过程。

一些在安全性测试中运用到的重要的术语

在我们说的更深入之前，了解一些网络应用程序的安全性测试中使用频繁的术语会很有帮助：

1、什么是“易受攻击性”？

这是网络应用程序的一个软肋。造成这个“软肋”的原因，可能是程序中的 **bug**，一种注入（SQL/脚本代码）或者已存在的病毒。

2、什么是“URL 处理”？

一些网络应用程序通过 URL 在客户端（浏览器）和服务器端之间进行额外信息的传递。有时在 URL 中改变信息可能会导致服务器不可预期的结果。

3、什么是“SQL 注入”？

是指通过网络应用的用户接口插入 SQL 指令到服务器所执行的查询中去的过程。

4、什么是“XSS（跨站式脚本攻击）”？

当一个用户通过 Web 应用接口插入 HTML 代码，而这种嵌入其中的代码对于其他用户是可见的，被称做 XSS。

5、什么是“欺骗”？

完全仿造网站或电子邮件的赝品被称为欺骗。

安全性测试方法：

为了对 Web 应用提供一次有效的安全性测试，安全测试人员应当对 HTTP 协议有很好的认知。对于客户端(浏览器)和服务器端之间如何运用 HTTP 通信有很好的了解是非常重要的。除此之外，测试人员需了解最基本的 SQL 注入以及跨站式脚本攻击。如果运气好的话，在 Web 应用上发现的安全漏洞的数目不会很多。不管怎样，能够对所发现的问题精确具体地描述安全漏洞能提供相当大的帮助。

1. 密码破解：

Web 应用上的安全测试可由“密码破解”开始。为了登录应用程序的非公开领域，可以通过猜测用户名/密码或者利用一些密码破解工具来达到相同的目的。常用用户名和密码列表与开源密码破解工具一样有可用价值。如果一个 web 应用不强制要求复杂的密码（例如，包含字母，数字和特殊字符，包含最小字符长度要求），那么这个用户名和密码不用花费很长的时间就可以被破解。

.....

查看全文请点击下载：<http://www.51testing.com/html/29/n-120029.html>

3. SQL 注入：

第二件应该在 SQL 注入中检查的是，在应用程序的任何一个文本框输入一个单引号应当是非法的。反之，如果测试人员遇到一个数据库错误，表示用户输入插入了一些查询并被应用程序所执行。这样的话，该应用容易受到 SQL 注入的攻击。

SQL 注入攻击是非常危险的，因为攻击者可以从服务器数据库获取一些重要的信息。检查你在 Web 应用上的 SQL 注入点，找出你代码库中那些能够直接在数据库上接收用户输入而执行的 MySQL 查询的代

码。

如果用户输入数据被用作查询数据库，攻击者可以注入 SQL 语句或者伪装 SQL 语句为用户输入从数据库提取重要信息。即使攻击者成功的摧毁了应用，在浏览器上显示了 SQL 查询错误，他们仍可以获取他们所找寻的信息。鉴于这种情况，用户输入的特殊字符应当被适当的处理或者避免。

4. 跨站式脚本攻击(XSS):

测试人员应当额外的检查对 Web 应用的跨站式脚本攻击。任何 HTML，比如<HTML>或者任何脚本，比如<SCRIPT>，应用程序都不应当接受。不然，该应用很容易受到跨站式脚本的攻击。

攻击者可以利用此方法在被攻击者的浏览器上执行恶意代码或者 URL。经此跨站式脚本攻击，攻击者能够利用像 JavaScript 之类的脚本来盗取用户 Cookie 以及存储在 Cookie 中的信息。

很多 Web 应用从不同页面的一些变量里传递或提取用户信息。

安全测试之维护服务器安全评测的七大技巧

安全测试之维护服务器安全评测的七大技巧

你的服务器上是否存有一些不能随意公开的重要数据呢?当然有吧?而最近,偏偏服务器遭受的风险又特别大,越来越多的病毒、心怀不轨的黑客,以及那些商业间谍都将服务器当作目标。很显然,服务器的安全问题一刻都忽视不得。

不可能在一篇文章中谈遍电脑安全问题,毕竟,市面上的已有许多这方面的书籍,不过,我倒是可以告诉你七个维护服务器安全的技巧。

技巧一：从基本做起

从基本做起是最保险的方式。你必须将服务器上含有机密数据的区域通通转换成 NTFS 格式;同理，防毒程序也必须按时更新。建议同时在服务器和桌面电脑上安装防毒软件。这些软件还应该设定成每天自动下载最新的病毒定义文件。另外，Exchange Server (邮件服务

器)也应该安装防毒软件,这类软件可扫描所有寄进来的电子邮件,寻找被病毒感染的附件,若发现病毒,邮件马上会被隔离,减低使用者被感染的机会。

另一个保护**网络**的好方法是依员工上班时间来限定使用者登录网络的权限。例如,上白天班的员工不该有权限在三更半夜登录网络。

最后,存取网络上的任何数据皆须通过密码登录。强迫大家在设定密码时,必须混用大小写字母、数字和特殊字符。在 Windows NT Server Resource Kit 里就有这样的工具软件。你还应该设定定期更新密码,且密码长度不得少于八个字符。若你已经做了这些措施,但还是担心密码不安全,你可以试试从网络下载一些黑客工具,然后测试一下这些密码到底有多安全。

技巧二: 保护备份

大多数人都没有意识到,备份本身就是一个巨大的安全漏洞,怎么说呢?试想,大多数的备份工作多在晚上 10 点或 11 点开始,依数据多寡,备份完成后大概也是夜半时分了。现在,想像一下,现在是凌晨四点,备份工作已经结束。有心人士正好可趁此时偷走备份磁盘,并在自己家中或是你竞争对手办公室里的服务器上恢复。不过,你可以阻止这种事情发生。首先,你可利用密码保护你的磁盘,若你的备份程序支持加密功能,你还可以将数据进行加密。其次,你可以将备份完成的时间定在你早上进办公室的时间,这样的话,即使有人半夜想溜进来偷走磁盘的话也无法了,因为磁盘正在使用中;如果窃贼强行把磁盘拿走,他一样无法读取那些损毁的数据。

技巧三: 使用 RAS 的回拨功能

Windows NT 最酷的功能之一就是支持服务器远端存取 (RAS)，不幸的是，RAS 服务器对黑客来说实在太方便了，他们只需要一个电话号码、一点耐心，然后就能通过 RAS 进入主机。不过你可以采取一些方法来保护 RAS 服务器的安全。

你所采用的技术主要端赖于远端存取者的工作方式。如果远端用户经常是从家里或是固定的地方上网，建议你使用回拨功能，它允许远端用户登录后即挂断，然后 RAS 服务器会拨出预设的电话号码接通用户，因为此一电话号码已经预先在程序中了，黑客也就没有机会指定服务器回拨的号码了。

另一个办法是限定远端用户只能存取单一服务器。你可以将用户经常使用到的数据复制到 RAS 服务器的一个特殊共用点上，再将远端用户的登录限制在一台服务器上，而非整个网络。如此一来，即使黑客入侵主机，他们也只能在单一机器上作怪，间接达到减少破坏的程度。

最后还有一个技巧就是在 RAS 服务器上使用“另类”网络协议。很都以 TCP/IP 协议当作 RAS 协议。利用 TCP/IP 协议本身的性质与接受程度，如此选择相当合理，但是 RAS 还支持 IPX/SPX 和 NetBEUI 协议，如果你使用 NetBEUI 当作 RAS 协议，黑客若一时不察铁定会被搞得晕头转向。

技巧四：考虑工作站的安全问题

在服务器安全的文章里提及工作站安全感觉似乎不太搭边，但是，工作站正是进入服务器的大门，加强工作站的安全能够提高整体网络的安全性。对于初学者，建议在所有工作站上使用 Windows 2000。Windows 2000 是一个非常安全操作系统，如果你没有 Windows 2000，那至少使用 Windows NT。如此你便能将工作站锁定，若没有权限，一般人将很难取得网络配置信息。

另一个技巧是限制使用者只能从特定工作站登录。还有一招是将工作站当作简易型的终端机(dumb terminal)或者说, 智慧型的简易终端机。换言之, 工作站上不会存有任何数据或软件, 当你将电脑当作 dumb terminal 使用时, 服务器必须执行 Windows NT 终端服务程序, 而且所有应用程序都只在服务器上运作, 工作站只能被动接收并显示数据而已。这意味着工作站上只有安装最少的 Windows 版本, 和一份微软 Terminal Server Client。这种方法应该是最安全的网络设计方案。

技巧五: 执行最新修补程序

微软内部有一组人力专门检查并修补安全漏洞, 这些修补程序(补丁)有时会被收集成 service pack(服务包)发布。服务包通常有两种不同版本: 一个任何人都可以使用的 40 位的版本, 另一个是只能在美国和加拿大发行的 128 位版本。128 位的版本使用 128 位的加密算法, 比 40 位的版本要安全得多。

一个服务包有时得等上好几个月才发行一次, 但要是严重点的漏洞被发现, 你当然希望立即进行修补, 不想苦等姗姗来迟的服务包。好在你并不需要等待, 微软会定期将重要的修补程序发布在它的 FTP 站上, 这些最新修补程序都尚未收录到最新版的服务包里, 我建议你经常去看看最新修补程序, 记住, 修补程序一定要按时间顺序来使用, 若使用错乱的话, 可能导致一些文件的版本错误, 也可能造成 Windows 当机。

技巧六: 颁布严格的安全政策

另一个提高安全性的方式就是制定一强有力的安全策略, 确保每一个人都了解, 并强制执行。若你使用 Windows 2000 Server, 你可以将部分权限授权给特定代理人, 而无须将全部的网管权利交出。即使你核定代理人某些权限, 你依然可县制其权限大小, 例如无法开设新的使用者帐号, 或改变权限等。

技巧七：防火墙，检查，再检查

最后一个技巧是仔细检查防火墙的设置。防火墙是网络规划中很重要的一部份，因为它能使公司电脑不受外界恶意破坏。

首先，不要公布非必要的 IP 地址。你至少要有个对外的 IP 地址，所有的网络通讯都必须经由此地址。如果你还有 DNS 注册的 Web 服务器或是电子邮件服务器，这些 IP 地址也要穿过防火墙对外公布。但是，工作站和其他服务器的 IP 地址则必须隐藏。

你还可以查看所有的通讯端口，确定不常用的已经全数关闭。例如，TCP/IP port 80 是用于 HTTP 流量，因此不能堵掉这个端口，也许 port 81 应该永远都用不着吧，所以就应该关掉。你可以在网络上查到每个端口的详细用途。

服务器安全问题是大大议题，你总不希望重要数据遭病毒/黑客损毁，或被人偷走做为不利你的用途，本文介绍了 7 个重要的安全检查关卡，你不妨试试看。

软件测试中日常测试中融入的安全测试

软件测试中日常测试中融入的安全测试

1. 用户权限测试

(1) 用户权限控制

- 1) 用户权限控制主要是对一些有权限控制的功能进行验证
- 2) 用户 A 才能进行的操作，B 是否能够进行操作（可通过窜 session，将在下面介绍）
- 3) 只能有 A 条件的用户才能查看的页面，是否 B 能够查看（可直接敲 URL 访问）

(2) 页面权限控制

- 1) 必须有登陆权限的页面，是否能够在不登陆情况下进行访问
- 2) 必须经过 A——B——C 的页面，是否能够直接由 A——C?

2. URL 安全测试

(1) 适用范围： URL 中含有参数，也就是通过 GET 方式传递的 HTTP 请求

(2) 什么叫 GET 方式？

HTTP 定义了与**服务器**交互的不同方法，最基本的方法是 GET 和 POST。

GET 方式在客户端通过 URL 提交数据，数据在 URL 中可以看到，例如在日常中订购服务：

`http://pay.daily.taobao.net/mysub/subdeal/order_sub_deal.htm?servId=2`

POST 方式，数据放置在 HTML HEADER 内提交，数据在 URL 中看不到

GET 只能传输比较少的数据，安全性较低，POST 传输数据较多，安全性也比 GET 高

(3) 测试关注点：

1) URL 参数检查：

A: 对 URL 中参数信息检查是否正确

如：URL 中的订单号、金额允许显示出来的话，需要验证其是否正确

B: 对于一些重要的参数信息，不应该在 URL 中显示出来

如：用户登陆时登录名、密码是否被显示出来了，

2) URL 参数值篡改

修改 URL 中的数据，看程序是否能识别：

如：对于以下 URL，修改其中 planId，看是程序是否可以识别：

`http://pay.daily.taobao.net/mysub/plan/subplan/confirmSubPlanInfo.htm?planId=878`

又如：对于 URL 中包含金额参数的，修改金额看是否能够提交成功（可能导致用户把 2 元金额改成 1 元金额能提交），还有修改订单号等重要信息看是否会报错

3) URL 中参数修改进行 XSS 注入：

什么是 XSS？

XSS 的全称是 Cross Site Script（跨站点脚本）

XSS 的原理很简单，即进行脚本注入，URL 执行时即把此脚本进行了执行，一般都是

JavaScript 脚本，如 `<script>alter(“abc”)</script>`

在 URL 中进行 XSS 注入，也就是把 URL 中的参数改成 JS 脚本。

4) URL 参数中进行 SQL 注入

什么是 SQL 注入？

SQL 注入全称是 SQL Injection，当应用程序使用输入内容来构造动态 sql 语句以访问数据库时，会发生 sql 注入攻击，如查询、插入数据时。

测试方法：URL 中写入 SQL 注入语句，看是否被执行，如：`' or 1=1;shutdown`

一般情况下要进行 SQL 注入攻击，需要对数据库类型、表名、判断逻辑、查询语句等比较清楚才能够写出有效的 SQL 注入语句。

3. 表单提交安全测试

适用范围：有表单提交的地方、有 HTTP 请求的地方(包括 GET、POST 请求)

测试关注点：

1) 表单中注入 XSS 脚本

什么是 XSS? 这已在上一节中说明。URL 中需要检测 XSS 注入，表单中更需要验证

测试方法：即在表单填写框中直接注入 JS 脚本

如在表单中输入 XSS 脚本，程序是不应该让脚本执行的

2) 表单中注入 SQL 脚本

与 URL 中参数进行 SQL 注入类似，就是在表单中写入 SQL 注入脚本提交看是否会有问题

4. Session 测试

(1) Session 是客户端与服务器端建立的会话，总是放在服务器上的，服务器会为每次会话建立一个 sessionId，每个客户会跟一个 sessionId 对应。

并不是关闭浏览器就结束了本次会话，通常是用户执行“退出”操作或者会话超时时才会结束。

(2) 测试关注点：

1) Session 互窜

Session 互窜即是用户 A 的操作被用户 B 执行了。

验证 Session 互窜，其原理还是基于权限控制，如某笔订单只能是 A 进行操作，或者只能是 A 才能看到的页面，但是 B 的 session 窜进来却能够获得 A 的订单详情等。

Session 互窜方法：

多 TAB 浏览器，在两个 TAB 页中都保留的是用户 A 的 session 记录，然后在其中一个 TAB 页执行退出操作，登陆用户 B，此时两个 TAB 页都是 B 的 session，然后在另一个 A 的页面执行操作，查看是否能成功。预期结果：有权限控制的操作，B 不能执行 A 页面的操作，应该报错，没有权限控制的操作，B 执行了 A 页面 操作后，数据记录是 B 的而不是 A 的。

2) Session 超时

基于 Session 原理，需要验证系统 session 是否有超时机制，还需要验证 session 超时后功能是否还能继续走下去。

测试方法：

1、打开一个页面，等着 10 分钟 session 超时时间到了，然后对页面进行操作，查看效果。

2、多 TAB 浏览器，在两个 TAB 页中都保留的是用户 A 的 session 记录，然后在其中一个 TAB 页执行退出操作，马上在另外一个页面进行要验证的操作，查看是能继续到下一步还是到登录页面。

软件 Web 安全性测试—SQL 注入

因为要对网站安全性进行测试，所以，学习了一些 sql 注入的知识。

在网上看一些 sql 注入的东东，于是想到了对网站的输入框进行一些测试，本来是想在输入框中输入<script>alter("abc")<script>,但是输入框有字符限制,只好输入<script>,结果网站出大问题了，呵呵，终于又出现了个 bug。

另一个就是在 URL 最后随意输入一些字符或数字，结果，新闻模块那出现了问题，暴露了网站的一些信息，又一个 bug，高兴下……

下面把今天看到的有关 SQL 注入方面的知识整理如下：

SQL 注入是一种攻击方式，在这种攻击方式中，恶意代码被插入到字符串中，然后将该字符串传递到 SQL Server 的实例以进行分析和执行。任何构成 SQL 语句的过程都应进行注入漏洞检查，因为 SQL Server 将执行其接收到的所有语法有效的查询。一个有经验的、坚定的攻击者甚至可以操作参数化数据。

SQL 注入的主要形式包括直接将代码插入到与 SQL 命令串联在一起并使其得以执行的用户输入变量。一种间接的攻击会将恶意代码注入要在表中存储或作为元数据存储的字符串。在存储的字符串随后串连到一个动态 SQL 命令中时，将执行该恶意代码。

注入过程的工作方式是提前终止文本字符串，然后追加一个新的命令。由于插入的命令可能在执行前追加其他字符串，因此攻击者将用注释标记“--”来终止注入的字符串。执行时，此后的文本将被忽略。

以下脚本显示了一个简单的 SQL 注入。此脚本通过串联硬编码字符串和用户输入的字符串而生成一个 SQL 查询：

```
var Shipcity;
```

```
ShipCity = Request.form. ("ShipCity");
```

```
var sql = "select * from OrdersTable where ShipCity = '" + ShipCity + "'";
```

用户将被提示输入一个市县名称。如果用户输入 Redmond，则查询将由与下面内容相似的脚本组成：

```
SELECT * FROM OrdersTable WHERE ShipCity = 'Redmond'
```

但是，假定用户输入以下内容：

```
Redmond'; drop table OrdersTable--
```

此时，脚本将组成以下查询：

```
SELECT * FROM OrdersTable WHERE ShipCity = 'Redmond';drop table OrdersTable--'
```

分号(;)表示一个查询的结束和另一个查询的开始。双连字符(--)指示当前行余下的部分是一个注释，应该忽略。如果修改后的代码语法正确，则**服务器**将执行该代码。SQL Server 处理该语句时，SQL Server 将首先选择 OrdersTable 中的所有记录（其中 ShipCity 为 Redmond）。然后，SQL Server 将删除 OrdersTable。

只要注入的 SQL 代码语法正确，便无法采用编程方式来检测篡改。因此，必须验证所有用户输入，并仔细检查在您所用的服务器中执行构造 SQL 命令的代码。本主题中的以下各部分说明了编写代码的最佳做法。

验证所有输入：

始终通过测试类型、长度、格式和范围来验证用户输入。实现对恶意输入的预防时，请注意应用程序的体系结构和部署方案。请注意，设计为在安全环境中运行的程序可能会被复制到不安全的环境中。以下建议应被视为最佳做法：

如果一个用户在需要邮政编码的位置无意中或恶意地输入了一个 10 MB 的 MPEG 文件，应用程序会做出什么反应？

如果在文本字段中嵌入了一个 DROP TABLE 语句，应用程序会做出什么反应？

测试输入的大小和数据类型，强制执行适当的限制。这有助于防止有意造成的缓冲区溢出。

输入字符 在 Transact-SQL 中的含义

； 查询分隔符。

’ 字符数据字符串分隔符。

-- 注释分隔符。

/* ... */ 注释分隔符。服务器不对/*和*/之间的注释进行处理。

xp_ 用于目录扩展存储过程的名称的开头，如 xp_cmdshell。

软件测试中 web 安全黑盒测试之保证测试的全面性

软件测试中 web 安全黑盒测试之保证测试的全面性

在目前的 WEB 安全黑盒**测试方法**中，一般是按照黑客攻击的手法进行测试，以达到准确性与全面性。那么，如何保证黑盒测试的全面性与准确性呢？总结一下，可以有以下几个方面：

1、对产品项目的熟悉程度。

测试之前，对项目进行了解跟踪，熟悉项目的所有功能、接口以及与其他项目的关联性（有时候 A 项目的功能会造成 B 项目存在安全风险）。

2、全面的技术知识。

由于每个项目的功能都不同，可能涉及到的应用就不同，有的项目是视频应用，就要了解 flash 脚本编写技术与前端配置知识，大多数 flash 蠕虫都是因为前端配置问题造成的。有些项目用到了 AJAX，那么**测试人员**就必须了解 AJAX 的知识。有的项目用到 ActiveX 插件，那么就要知道 ActiveX 可能造成的安全问题，等等。所以，安全测试人员要掌握全面技术知识，才可以对每个项目进行测试，并不因为新项目中包含新的技术而放弃测试。同时还要有不断的学习能力，遇到未知的技术要进行快速学习，然后对项目进行测试。

3、超强的漏洞挖掘能力，以及实战能力。

安全测试时，必须按照黑客攻击的手法进行测试，所以，这就要求 WEB 安全测试人员拥有超强的漏洞挖掘能力与漏洞认知度。同样还要拥有实战经验，一个没有实践经验的测试人员，不是一个好的安全测试人员，当安全测试人员并不知道安全 BUG 所造成的的方式与利用后所造成的影响，就不能全部的发现所有安全 BUG，同时又不能在各个安全 BUG 危险度上进行分级，这就造成一些安全 BUG 的疏漏。

4、黑盒测试标准

总结出一个黑盒测试标准文档，对所有可能影响的安全漏洞进行罗列，并详细描述黑盒测试的方法与步骤，在项目**测试过程**中对条目中的所有漏洞进行检测，并严格按照规定的方法进行测试。

5、细心+用心

一个很小的功能，就可能造成很大的安全漏洞，如果未测试到就进行上线，就可能造成黑客攻击，所有的用户帐户被盗取，或者应用瘫痪。

所谓工欲善其事, 必先利其器，这里列出了一些常用的**黑盒测试工具**：

1、扫描工具：

Web Vulnerability Scanner

Ratproxy

2、嗅探工具：

Wireshark

Fiddler2

WebScarab

burpsuite

SPIKEProxy

appsniiff

httpwatch

Paros

3、测试工具：

Web2Fuzz

pangolin

sqlmap

Firefox+插件:

FileEncrypter HASH 转换

<https://addons.mozilla.org/firefox/3208>

nftools HASH 转换

<http://www.net-force.nl/files/download/nftools/nftools.xpi>

Greasemonkey 在网页内实时插入脚本

<https://addons.mozilla.org/firefox/748/>

hackbar SQL 注入辅助

<https://addons.mozilla.org/firefox/3899/>

Add n Edit Cookies 编辑 COOKIES

<https://addons.mozilla.org/firefox/573/>

Poster 自定义构造 POST 的提交 包括文件上传

<https://addons.mozilla.org/fr/firefox/addon/2691>

RefControl 编辑 REF 来源等

<https://addons.mozilla.org/firefox/953/>

LiveHTTPHeaders 方便记录给次的 GET 和 POST 提交并可以 relpaly 修改 HTTP 请求包

<https://addons.mozilla.org/firefox/3829/>

Tamper Data 监视所有的 GET 和 POST 提交

<https://addons.mozilla.org/firefox/966/>

User Agent Switcher 可以伪造 User Agent

<https://addons.mozilla.org/firefox/59/>

View Dependencies 展示页面所有相关的文件

<https://addons.mozilla.org/firefox/2214/>

Web Developer 控制打开关闭 HTML 元素

<https://addons.mozilla.org/firefox/60/>

Jsview 查看整个页面的 JS

<https://addons.mozilla.org/firefox/2076/>

FormFox 自动识别提交表单指向的 URL

<https://addons.mozilla.org/fr/firefox/addon/1579>

FireBug 自动查找页面语法错误

<https://addons.mozilla.org/firefox/1843/>

httpfox http 协议分析器

<https://addons.mozilla.org/en-US/firefox/addon/6647>

总结:

只有对项目中所有的应用接口与功能进行全面测试,并对可能造成的风险进行一一审核,兼备完善的 WEB 安全技术,同时,由于未来情况下可能会存在新的攻击方式,并时刻关注业内安全事件,对于新的攻击手段进行跟踪学习,应用到安全测试中,才可以达到安全黑盒测试的全面性,保证应用的安全性。

安全性测试: SYN flood 网络攻击

1 SYN Flood 攻击介绍:

拒绝服务攻击(Denial of Service, DoS)是目前比较有效而又非常难于防御的一种网络攻击方式,它的目的就是使服务器不能够为正常访问的用户提供服务。所以,DoS 对一些紧密依靠互联网开展业务的企业和组织带来了致命的威胁。

SYN Flood 是最为有效和流行的一种 DoS 攻击形式。它利用 TCP 三次握手协议的缺陷,向目标主机发送大量的伪造源地址的 SYN 连接请求,消耗目标主机的资源,从而不能够为正常用户提供服务。

1.1 TCP 连接建立的过程

要掌握 SYN Flood 攻击的基本原理,必须先介绍 TCP 的三次握手机制。

TCP 三次握手过程如下:

- 1)客户端向服务器端发送一个 SYN 置位的 TCP 报文,包含客户端使用的端口号和初始序列号 x;

2) 服务器端收到客户端发送来的 SYN 报文后，向客户端发送一个 SYN 和 ACK 都置位的 TCP 报文，包含确认号为 $x+1$ 和服务器的初始序列号 y ；

3)

TCP 客户端

客户端端口

(1024—65535)

TCP 服务器端

服务器端口

(1—1023)

SYN

SYN/ACK

ACK

客户端收到服务器返回的 SYN+ACK 报文后，向服务器返回一个确认号为 $y+1$ 序号为 $x+1$ 的 ACK 报文，一个标准的 TCP 连接完成。如图 1 所示：

1.2 攻击原理

在 SYN Flood 攻击中，黑客机器向受害主机发送大量伪造源地址的 TCP SYN 报文，受害主机分配必要的资源，然后向源地址返回 SYN+ACK 包，并等待源端返回 ACK 包，如图 2 所示。由于源地址是伪造的，所以源端永远都不会返回 ACK 报文，受害主机继续发送 SYN+ACK

包，并将半连接放入端口的积压队列中，虽然一般的主机都有超时机制和默认的重传次数，但是由于端口的半连接队列的长度是有限的，如果不断的向受害主机发送大量的 TCP SYN 报文，半连接队列就会很快填满，服务器拒绝新的连接，将导致该端口无法响应其他机器进行的连接请求，最终使受害主机的资源耗尽。

TCP 客户端

客户端端口

(1024—65535)

TCP 服务器端

服务器端口

(1—1023)

SYN

SYN/ACK

伪造源地址

2 几种防御技术

SYN Flood 攻击给互联网造成重大影响后，针对如何防御 SYN Flood 攻击出现了几种比较有效的技术。

2.1 SYN—cookie 技术

一般情况下，当服务器收到一个 TCP SYN 报文后，马上为该连接请求分配缓冲区，然后返回一个 SYN+ACK 报文，这时形成一个半连接。SYN Flood 正是利用了这一点，发送大量的伪造源地址的 SYN 连接请求，而不完成连接。这样就大量的消耗的服务器的资源。

SYN-cookie 技术针对标准 TCP 连接建立过程资源分配上的这一缺陷，改变了资源分配的策略。当服务器收到一个 SYN 报文后，不立即分配缓冲区，而是利用连接的信息生成一个 cookie，并将这个 cookie 作为将要返回的 SYN+ACK 报文的初始序列号。当客户端返回一个 ACK 报文时，根据包头信息计算 cookie，与返回的确认序列号（初始的序列号+1）的前 24 位进行对比，如果相同，则是一个正常连接，然后，分配资源，建立连接。

该技术的巧妙之处在于避免了在连接信息未完全到达前进行资源分配，使 SYN Flood 攻击的资源消耗失效。实现的关键之处在于 cookie 的计算。cookie 的计算应该做到包含本次连接的状态信息，使攻击者不能伪造 cookie。cookie 的计算过程如下：

1) 服务器收到一个 SYN 包后，计算一个消息摘要 mac：

$$\text{mac} = \text{MAC} (A, k) ;$$

MAC 是密码学中的一个消息认证码函数，也就是满足某种安全性质的带密钥的 hash 函数，它能够提供 cookie 计算中需要的安全性。

A 为客户和服务双方 IP 地址和端口号以及参数 t 的串联组合：

$$A = \text{SOURCE_IP} \parallel \text{SOURCE_PORT} \parallel \text{DST_IP} \parallel \text{DST_PORT} \parallel t$$

K 为服务器独有的密钥；

时间参数 t 为 32 比特长的时间计数器，每 64 秒加 1；

2) 生成 cookie：

$$\text{cookie} = \text{mac} (0:24) ; \text{表示取 mac 值的第 0 到 24 比特位；}$$

3) 设置将要返回的 SYN+ACK 报文的初始序列号, 设置过程如下:

- i. 高 24 位用 cookie 代替;
- ii. 接下来的 3 比特位用客户要求的最大报文长度 MSS 代替;
- iii. 最后 5 比特位为 $t \bmod 32$ 。

客户端收到来自服务器 SYN+ACK 报文后, 返回一个 ACK 报文, 这个 ACK 报文将带一个 cookie(确认号为服务器发送过来的 SYN ACK 报文的初始序列号加 1, 所以不影响高 24 位), 在服务器端重新计算 cookie, 与确认号的前 24 位比较, 如果相同, 则说明未被修改, 连接合法, 然后, 服务器完成连接的建立过程。

SYN-cookie 技术由于在连接建立过程中不需要在服务器端保存任何信息, 实现了无状态的三次握手, 从而有效的防御了 SYN Flood 攻击。但是该方法也存在一些弱点。由于 cookie 的计算只涉及了包头的部分信息, 在连接建立过程中不在服务器端保存任何信息, 所以失去了协议的许多功能, 比如, 超时重传。此外, 由于计算 cookie 有一定的运算量, 增加了连接建立的延迟时间, 因此, SYN-cookie 技术不能作为高性能服务器的防御手段。通常采用动态资源分配机制, 当分配了一定的资源后再采用 cookie 技术, **Linux** 就是这样实现的。还有一个问题是, 当我们避免了 SYN Flood 攻击的同时, 同时也提供了另一种拒绝服务攻击方式, 攻击者发送大量的 ACK 报文, 使服务器忙于计算验证。尽管如此, 在预防 SYN Flood 攻击方面, SYN-cookie 技术仍然是一种有效的技术。

2.2 地址状态监控的解决方法

地址状态监控的解决方法是利用监控工具对网络中的有关 TCP 连接的数据包进行监控, 并对监听到的数据包进行处理。处理的主要依据是连接请求的源地址。

每个源地址都有一个状态与之对应，总共有四种状态：

初态：任何源地址刚开始的状态；

NEW 状态：第一次出现或出现多次也不能断定存在的源地址的状态；

GOOD 状态：断定存在的源地址所处的状态；

BAD 状态：源地址不存在或不可达时所处的状态。

具体的动作和状态转换根据 TCP 头中的位码值决定：

1) 监听到 SYN 包，如果源地址是第一次出现，则置该源地址的状态为 NEW 状态；如果是 NEW 状态或 BAD 状态；则将该包的 RST 位置 1 然后重新发出去，如果是 GOOD 状态不作任何处理。

2) 监听到 ACK 或 RST 包，如果源地址的状态为 NEW 状态，则转为 GOOD 状态；如果是 GOOD 状态则不变；如果是 BAD 状态则转为 NEW 状态；如果是 BAD 状态则转为 NEW 状态。

3) 监听到从服务器来的 SYN ACK 报文（目的地址为 addr），表明服务器已经为从 addr 发来的连接请求建立了一个半连接，为防止建立的半连接过多，向服务器发送一个 ACK 包，建立连接，同时，开始计时，如果超时，还未收到 ACK 报文，证明 addr 不可达，如果此时 addr 的状态为 GOOD 则转为 NEW 状态；如果 addr 的状态为 NEW 状态则转为 BAD 状态；如果为 addr 的状态为 BAD 状态则不变。

状态的转换图如图 3 所示：

初态

GOOD

NEW

BAD

ACK/RST

SYN

ACK/RST

ACK 包确认超时

ACK/RST

ACK 包确认超时

下面分析一下基于地址状态监控的方法如何能够防御 SYN Flood 攻击。

1) 对于一个伪造源地址的 SYN 报文，若源地址第一次出现，则源地址的状态为 NEW 状态，当监听到服务器的 SYN+ACK 报文，表明服务器已经为该源地址的连接请求建立了半连接。此时，监控程序代源地址发送一个 ACK 报文完成连接。这样，半连接队列中的半连接数不是很多。计时器开始计时，由于源地址是伪造的，所以不会收到 ACK 报文，超时后，监控程序发送 RST 数据包，服务器释放该连接，该源地址的状态转为 BAD 状态。之后，对于每一个来自该源地址的 SYN 报文，监控程序都会主动发送一个 RST 报文。

2) 对于一个合法的 SYN 报文，若源地址第一次出现，则源地址的状态为 NEW 状态，服务器响应请求，发送 SYN+ACK 报文，监控程序发送 ACK 报文，连接建立完毕。之后，来自客户端的 ACK 很快会到达，该源地址的状态转为 GOOD 状态。服务器可以很好的处理重复到达的 ACK 包。

从以上分析可以看出，基于监控的方法可以很好的防御 SYN Flood 攻击，而不影响正常用户的连接。

3 小结

本文介绍了 SYN Flood 攻击的基本原理，然后详细描述了两种比较有效和方便实施的防御方法：SYN-cookie 技术和基于监控的源地址状态技术。SYN-cookie 技术实现了无状态的握手，避免了 SYN Flood 的资源消耗。基于监控的源地址状态技术能够对每一个连接服务器的 IP 地址的状态进行监控，主动采取措施避免 SYN Flood 攻击的影响。这两种技术是目前所有的防御 SYN Flood 攻击的最为成熟和可行的技术。

通用漏洞测试工具设计

现在信息**安全**已经成为热门行业，而程序漏洞又在信息安全里面占有很重要的地位，因此我们有必要对如何研究漏洞、找出程序中的臭虫进行研究。

在我写的《漏洞研究方法总结》里面总结了一些经验，建立安全模型，源代码、二进制代码分析等的一些研究方法，但总体上还是人工去找漏洞，这对于研究的人员要求也比较高。现在大量的服务软件出来，而基本上都存在着漏洞，完全靠人员去一个个的分析代码**测试**显然满足不了**需求**，所以我们需要设计一个比较自动化能够适合多数**服务器**的**测试工具**。原来也编写过测试工具，利用测试工具辅助研究。但原来编写的 getiisfile 还是具有很强的针对性，基本上只能用于 WEB 服务等的一些研究，也不够自动化。

那我们如何设计一个比较通用的测试工具呢？不同的服务有不同的协议，而很多漏洞研究也是协议相关性很大，所以我们研究一个服务的漏洞，往往先研究其协议、熟悉其软件等，然后研究其漏洞。这显然是一个漫长的过程，所以我们的测试软件不能走这套路子。如果我

们的测试软件里面也需要“掌握”每个服务的协议知识，那显然就不能很好的做到通用，所以我们需要提炼一些服务的共性。仔细想想，现在的大多服务模型就是一个服务端，一个客户端，两者利用通信协议通信，所以服务的共性就是数据包的通信，具体每个数据包里面什么内容怎么解释，那就是看每个服务采用的协议，我们的测试程序不能涉及到具体协议。再想想，像 IIS 的 .htr 的溢出包，就是请求“GET /aaa.htr HTTP/1.1 \r\n HOST:host”的“aaa.htr”这个域比较长，IIS 的 .ipp 的溢出包，就是请求“GET /a.printer HTTP/1.1 \r\n HOST:host”的“host”这个域比较长而又没有超过一个范围。很多服务的漏洞不就是包里面的某个域的数据过长或者数值不对的问题吗？所以我们的漏洞测试工具的主要任务就是构造、自动发送这些某个域数据有问题的包。怎么构造这样的包呢？我们可以采用脚本的方法，脚本指定怎么构造、构造什么样的包。测试不同的服务，只要编写针对这个服务的简单脚本，这样程序就能够通用了。不同服务采用不同的协议，他们接收的包要有一定的协议要求，那怎么解决协议的问题呢。我们可以采用样本包的办法，就是脚本里面提供一个样本包，构造包都在这个样本包的基础上，这样就可以满足包的协议要求，也可以避免测试每个服务我们需要掌握其协议的要求。

其实我们的设计主要两点好的思路，一个就是脚本，这个使得我们的程序编写好后比较通用，而又能够比较灵活。再一个就是样本包的解决办法，这个屏蔽了具体协议，这样我们的测试已经基本上是协议无关的了。比如我们要测试一个新的**数据库**软件的漏洞，不需要先分析好它的协议、包格式后才能开始了。

好了，现在基本设计方案已经明朗了，剩下的就是需要考虑脚本包含一些什么内容，以能够更有效的进行测试。

我们可以把脚本提供的一些相关参数放一块，组成一个节，这样方便我们的程序读取脚本参数，也方便我们自己编写、阅读脚本。还有就是最好支持注释。

一、测试目标；

- 1、目标 IP；
- 2、协议；这儿的协议当然是 UDP，TCP、ICMP 等；
- 3、端口；有些协议没有端口的概念；
- 4、可能这些项通过样本包直接提供。

二、样本包；

1、样本包提供的方式；为了更有效的进行测试，我们有必要通过多种途径提供样本包。如脚本里面直接提供样本包，这个需要处理包的输入问题，比如我们的脚本是以回车换行分隔各个脚本变量等，那么样本包显然需要处理回车换行等，还有 0 的问题，不可显示字符的输入，所以基本上需要一个 16 进制的输入和直接字符输入的混合输入方法。通过一个样本包数据文件提供样本包。通过抓包提供样本包。这个抓包提供样本包，可以方便我们到具体应用环境测试，因为这个测试工具的测试效果显然主要靠样本包要能够是有问题包的一个基准，所以样本包越丰富，测试得也会越多。再一个可以第一项的测试目标也可以通过样本包提供。

三、测试包组装；

1、发包前固定发送的信息；这个像有些 FTP 的测试，可能需要认证，而是需要**测试认证**后的命令处理。可以每次发包前固定发送认证内容，这样就可以通过认证了。

2、样本包的分隔符;用于把样本包的各个域分隔出来,分隔符可以指定多个。比如请求“GET /a.htr?a=var1&b=var2 HTTP/1.1\r\nHOST:host\r\n\r\n”,分割符就可能有空格” ”、“?”、“=”、“&”、“\r”、“\n”、“:”等,当然也要处理 16 进制的输入问题。

3、测试域范围;包分隔后从第几到第几个域进行测试,或者从样本包的哪个偏移到哪个偏移进行测试。

4、测试的类型;服务软件处理一个包里面常见的错误一个是包字符串的处理错误,一个是一些数值的错误。

对于字符串测试:

5、添加的域前缀、后缀;比如测试 CGI 的问题,可能需要 URL 域后面要有固定串”.cgi”。

6、域串测试开始长度、结束长度、步长;

7、填充所用的字符;一般测试可能就用”a”填充,但如果要测试那个域的处理是否有格式串的问题,可能就需要填充”%s”、“%n”等这样的串了。

对于数值测试:

8、数值字段单位(字节、字、双字)、开始、结束、步长;

四、测试包发送;

1、是否采用阻塞模式;

2、每次发包是否重新连接;

3、两个包发送之间等待时间;

4、每个包发送次数;

五、返回信息处理;

1、 是否接收返回信息;

2、 返回信息是否记录;

3、 返回信息特征串的搜索, 报警处理; 比如 IIS 返回 "The remote procedure call failed", 就是 RPC 服务出现了错误, 一般就是溢出或者有漏洞什么的了, 可以进行报警或者 LOG 里面进行记录。

4、 返回信息的分析处理, 判断服务有问题; 这点可能一时还比较难。

比如:

```
getiisfile www.iis.com 80 "GET /" 1 a .ida
```

返回 " 找不到 IDQ 文件 c:\inetpub\wwwroot\a.ida。 "

```
getiisfile www.iis.com 80 "GET /" 10 a .ida
```

返回 " 找不到 IDQ 文件 c:\inetpub\wwwroot\aaaaaaaaa.ida。 "

```
getiisfile 61.132.55.67 80 "GET /" 239 a .ida
```

返回 " c:\inetpub\wwwroot\aaa

aaa

a

aa

a

aaaaaaaaaaaaaaaaaaaaaaaaaaaaa. i???? ?r。系统找不到指定的路径。”

显然此程序已经发生了错误，239 个” a” 的请求已经与 1 个、10 个” a” 的请求不一样了，多了后面一个尾巴” ????? ?r”。其实是 strncpy 调用没有串结尾的一个问题，这样导致泄露了堆栈的内容。要根据返回信息自动判断这些问题，可能算法就比较难做了，这种问题我们还是采用人工识别可能好做点。

六、LOG;

1、 测试状态的 LOG;方便测试停止后的继续测试。

2、 每个测试的状态、返回信息的 LOG;

这样我们可以构造发送各种包，显然还不能算完，我们还得知道这些包的效果，服务软件到底出问题没有。现在判断问题主要根据返回信息，这显然不够，这只能收集到很少部分信息，主要信息显然都在服务器上，所以我们还需要有服务端运行的软件配合检测服务软件的状态。这个大致可以从下面一些方面入手：

一、 内存使用监视;这个可以用现有软件或者写软件监视，可以判断一些内存不释放的漏洞。

二、 异常的监视;像一些溢出、数值非法导致的内存非法访问等往往会导致一些异常，这个保护模式的 CPU 就会产生一个异常中断，在 **WINDOWS** 下程序就会返回到 NTDLL.DLL 的引出函数” KiUserExceptionDispatcher” 的入口，所以可以编写软件拦截这个入口或者在

SOFT-ICE 下设置这个断点，再根据调用这个入口提供的数据结构，得到发生异常时的 EIP 等寄存器内容，判断是否程序真正有问题。*NIX 系统下可以采用可加载内核模块 (LKM) 拦截系统的异常中断，然后进行综合判断。如果写程序，难点就主要在这个判断是否程序错误、漏洞的算法，不过看来应该还比较乐观，差不多出现异常就是有问题了。*NIX 系统由于基本上没有结构异常这个概念，所以一般程序发生错误基本上就会进程死掉等，所以很好判断。内存交换出去了等异常 WINDOWS 自己识别了不会返回到这个入口，再就是 WINDOWS 结束一个线程等可能会进入这个入口，但基本上得到的发生异常的 EIP 是比较固定的，所以很好排除。但 WINDOWS 编程有结构异常的概念，所以如果不拦截异常，很多程序的错误可能就不会暴露出来。

三、服务进程状态的监视；像很多服务程序出现问题后，系统可能会杀死服务进程，IIS5 等系统会自动重新启动此服务进程，*NIX 等系统可能会产生进程 DUMP 的操作，所以我们都可以根据这些信息判断服务软件是否产生了错误。

手工注入猜解语句

自己在学习过程中总结的一些东西，在这发出来，希望对看到的人能有所帮助吧！

猜解表名：

```
and exists (select * from 表名)
```

猜解列名：

```
and exists (select 字段 from 表名)
```

UNION 法：

联合查询：

```
select name,password,id from user union select user,pwd,uid from
```

爆指定表名内容:

```
and 1=1 union select 1,2,3,4,5 from 表名
```

ASCII 逐字解码法:

1、猜解列长度

猜解语句:

```
and (select top 1 len(列名)from 表名)>N and (select top 1 len(列名)from 表名)=N
```

其中 N 是数字, 变换这个 N 的值猜解列长度, 例如:

```
and (select top 1 len(列名)from 表名)>1 and (select top 1 len(列名)from 表名)>6
```

如果一直猜到 6 都显示正常页面, 猜到 7 的时候返回错误 (大于 6 并且小于等于 7), 那么该列的长度为 7。因为 “top 1” 的意思是把最靠前的 1 条记录给提取出来, 所以如果要猜解第二条记录就该使用:

```
select top 1 len(列名) from 表名
```

```
where 列名 not in (select top 1 列名 from 表名)
```

2、ASCII 码分析法猜解用户和密码

ASC() 函数和 Mid 函数

例如: mid(列名,N,1)

ASC(mdi(列名,N,1))得到 “列名” 第 N 位字符 ASCII 码

猜解语句为:

and (select top 1 asc(mid(字段,1,1)) from 数据库名)=ASC 码(通过转换工具换)

区间判断语句:

“.....between.....and.....”

中文处理法:

当 ASCII 转换后为“负数”使用 abs() 函数取绝对值。

例: and (select top 1 abs(asc(mid(字段,1,1))) from 数据库名)=ASC 码(通过转换工具换)

ASCII 逐字解码法的应用:

1、猜解表名:

and (select count(*) from admin)<>0

如何做好软件安全测试

近来,在我负责的公司某软件产品的最后测试工作,常常被问到这样一个问题:在做**测试过程**中,我们的软件产品在安全性方面考虑了多少?应该如何测评一个软件到底有多安全?

这个软件因为涉及客户商业上重要的信息资料,因此用户关心的核心问题始终围绕“这个软件安全吗”。一个由于设计导致的安全漏洞和一个由于实现导致的安全漏洞,对用户的最终影响都是巨大的。我的任务就是确保这个软件在安全性方面能满足客户期望。

什么是软件安全性测试

(1) 什么是软件安全

软件安全属于软件领域里一个重要的子领域。在以前的单机时代，安全问题主要是操作系统容易感染病毒，单机应用程序软件安全问题并不突出。但是自从互联网普及后，软件安全问题愈加显加突显，使得软件安全性测试的重要性上升到一个前所未有的高度。

软件安全一般分为两个层次，即应用程序级别的安全性和操作系统级别的安全性。应用程序级别的安全性，包括对数据或业务功能的访问，在预期的安全性情况下，操作者只能访问应用程序的特定功能、有限的的数据等。操作系统级别的安全性是确保只有具备系统平台访问权限的用户才能访问，包括对系统的登录或远程访问。

本文所讲的软件安全主要是应用程序层的安全，包括两个层面：①是应用程序本身的安全性。一般来说，应用程序的安全问题主要是由软件漏洞导致的，这些漏洞可以是设计上的**缺陷**或是编程上的问题，甚至是**开发人员**预留的后门。②是应用程序的数据安全，包括数据存储安全和数据传输安全两个方面。

(2) 软件安全性测试

一般来说，对安全性要求不高的软件，其安全性测试可以混在**单元测试**、**集成测试**、**系统测试**里一起做。但对安全性有较高需求的软件，则必须做专门的安全性测试，以便在破坏之前预防并识别软件的安全问题。

安全性测试 (Security Testing) 是指有关验证应用程序的安全等级和识别潜在安全性缺陷的过程。应用程序级安全测试的主要目的是查找软件自身程序设计中存在的安全隐患，并检查应用程序对非法侵入的防范能力，根据安全指标不同测试策略也不同。注意：安全性测试并不最终证明应用程序是安全的，而是用于验证所设立策略的有效性，这些对策是基于

威胁分析阶段所做的假设而选择的。例如，测试应用软件在防止非授权的内部或外部用户的访问或故意破坏等情况时的运作。

软件安全性测试过程

(1) 安全性测试方法

有许多的测试手段可以进行安全性测试，目前主要安全测试方法有：

①静态的代码安全测试：主要通过对源代码进行安全扫描，根据程序中数据流、控制流、语义等信息与其特有软件安全规则库进行比对，从中找出代码中潜在的安全漏洞。静态的源代码安全测试是非常有用的方法，它可以在编码阶段找出所有可能存在安全风险的代码，这样开发人员可以在早期解决潜在的安全问题。而正因为如此，静态代码测试比较适用于早期的代码开发阶段，而不是测试阶段。

②动态的渗透测试：渗透测试也是常用的安全测试方法。是使用自动化工具或者人工的方法模拟黑客的输入，对应用系统进行攻击性测试，从中找出运行时刻所存在的安全漏洞。这种测试的特点就是真实有效，一般找出来的问题都是正确的，也是较为严重的。但渗透测试一个致命的缺点是模拟的测试数据只能到达有限的测试点，覆盖率很低。

③程序数据扫描。一个有高安全性需求的软件，在运行过程中数据是不能遭到破坏的，否则就会导致缓冲区溢出类型的攻击。数据扫描的手段通常是进行内存测试，内存测试可以发现许多诸如缓冲区溢出之类的漏洞，而这类漏洞使用除此之外的测试手段都难以发现。例如，对软件运行时的内存信息进行扫描，看是否存在一些导致隐患的信息，当然这需要专门的工具来进行验证，手工做是比较困难的。

(2) 反向安全性测试过程

大部分软件的安全测试都是依据缺陷空间反向设计原则来进行的,即事先检查哪些地方可能存在安全隐患,然后针对这些可能的隐患进行测试。因此,反向测试过程是从缺陷空间出发,建立缺陷威胁模型,通过威胁模型来寻找入侵点,对入侵点进行已知漏洞的扫描测试。好处是可以对已知的缺陷进行分析,避免软件里存在已知类型的缺陷,但是对未知的攻击手段和方法通常会无能为力。

①建立缺陷威胁模型。建立缺陷威胁模型主要是从已知的安全漏洞入手,检查软件中是否存在已知的漏洞。建立威胁模型时,需要先确定软件牵涉到哪些专业领域,再根据各个专业领域所遇到的攻击手段来进行建模。

②寻找和扫描入侵点。检查威胁模型里的哪些缺陷可能在本软件中发生,再将可能发生的威胁纳入入侵点矩阵进行管理。如果有成熟的漏洞扫描工具,那么直接使用漏洞扫描工具进行扫描,然后将发现的可疑问题纳入入侵点矩阵进行管理。

③入侵矩阵的验证测试。创建好入侵矩阵后,就可以针对入侵矩阵的具体条目设计对应的**测试用例**,然后进行测试验证。

(3) 正向安全性测试过程

为了规避反向设计原则所带来的测试不完备性,需要一种正向的测试方法来对软件进行比较完备的测试,使测试过的软件能够预防未知的攻击手段和方法。

①先标识测试空间。对测试空间的所有的可变数据进行标识,由于进行安全性测试的代价高昂,其中要重点对外部输入层进行标识。例如,**需求分析**、概要设计、详细设计、编码这几个阶段都要对测试空间进行标识,并建立测试空间跟踪矩阵。

②精确定义设计空间。重点审查需求中对设计空间是否有明确定义，和需求牵涉到的数据是否都标识出了它的合法取值范围。在这个步骤中，最需要注意的是精确二字，要严格按照安全性原则来对设计空间做精确的定义。

③标识安全隐患。根据找出的测试空间和设计空间以及它们之间的转换规则，标识出哪些测试空间和哪些转换规则可能存在安全隐患。例如，测试空间愈复杂，即测试空间划分越复杂或可变数据组合关系越多也越不安全。还有转换规则愈复杂，则出问题的可能性也愈大，这些都属于安全隐患。

④建立和验证入侵矩阵。安全隐患标识完成后，就可以根据标识出来的安全隐患建立入侵矩阵。列出潜在安全隐患，标识出存在潜在安全隐患的可变数据，和标识出安全隐患的等级。其中对于那些安全隐患等级高的可变数据，必须进行详尽的**测试用例设计**。

(4) 正向和反向测试的区别

正向测试过程是以测试空间为依据寻找缺陷和漏洞，反向测试过程则是以已知的缺陷空间为依据去寻找软件中是否会发生同样的缺陷和漏洞，两者各有其优缺点。反向测试过程主要的一个优点是成本较低，只要验证已知的可能发生的缺陷即可，但缺点是测试不完善，无法将测试空间覆盖完整，无法发现未知的攻击手段。正向测试过程的优点是测试比较充分，但工作量相对来说较大。因此，对安全性要求较低的软件，一般按反向测试过程来测试即可，对于安全性要求较高的软件，应以正向测试过程为主，反向测试过程为辅。

常见的软件安全性缺陷和漏洞

软件的安全有很多方面的内容，主要的安全问题是由软件本身的漏洞造成的，下面介绍常见的软件安全性缺陷和漏洞。

(1) 缓冲区溢出

缓冲区溢出已成为软件安全的头号公敌，许多实际中的安全问题都与它有关。造成缓冲区溢出问题通常有以下两种原因。①设计空间的转换规则的校验问题。即缺乏对可测数据的校验，导致非法数据没有在外部输入层被检查出来并丢弃。非法数据进入接口层和实现层后，由于它超出了接口层和实现层的对应测试空间或设计空间的范围，从而引起溢出。②局部测试空间和设计空间不足。当合法数据进入后，由于程序实现层内对应的测试空间或设计空间不足，导致程序处理时出现溢出。

(2) 加密弱点

这几种加密弱点是不安全的：①使用不安全的加密算法。加密算法强度不够，一些加密算法甚至可以用穷举法破解。②加密数据时密码是由伪随机算法产生的，而产生伪随机数的方法存在缺陷，使密码很容易被破解。③身份验证算法存在缺陷。④客户机和**服务器**时钟未同步，给攻击者足够的时间来破解密码或修改数据。⑤未对加密数据进行签名，导致攻击者可以篡改数据。所以，对于加密进行测试时，必须针对这些可能存在的加密弱点进行测试。

(3) 错误处理

一般情况下，错误处理都会返回一些信息给用户，返回的出错信息可能会被恶意用户利用来进行攻击，恶意用户能够通过分析返回的错误信息知道下一步要如何做才能使攻击成功。如果错误处理时调用了一些不该有的功能，那么错误处理的过程将被利用。错误处理属于异常空间内的处理问题，异常空间内的处理要尽量简单，使用这条原则来设计可以避免这个问题。但错误处理往往牵涉到易用性方面的问题，如果错误处理的提示信息过于简单，用户可能会一头雾水，不知道下一步该怎么操作。所以，在考虑错误处理的安全性的同时，需要和易用性一起进行权衡。

(4) 权限过大

如果赋予过大的权限,就可能导致只有普通用户权限的恶意用户利用过大的权限做出危害安全的操作。例如没有对能操作的内容做出限制,就可能导致用户可以访问超出规定范围的其他资源。进行安全性测试时必须测试应用程序是否使用了过大的权限,重点要分析在各种情况下应该有的权限,然后检查实际中是否超出了给定的权限。权限过大问题本质上属于设计空间过大问题,所以在设计时要控制好设计空间,避免设计空间过大造成权限过大的问题。

做好安全性测试的建议

许多软件安全测试经验告诉我们,做好软件安全性测试的必要条件是:一是充分了解软件安全漏洞,二是评估安全风险,三是拥有高效的软件安全测试技术和工具。

(1) 充分了解软件安全漏洞

评估一个软件系统的安全程度,需要从设计、实现和部署三个环节同时着手。我们先看一下 Common Criteria 是如何评估软件系统安全的。首先要确定软件产品对应的 Protection Profile (PP)。一个 PP 定义了一类软件产品的安全特性模板。例如数据库的 PP、防火墙的 PP 等。然后,根据 PP 再提出具体的安全功能需求,如用户的身份认证实现。最后,确定安全对象以及是如何满足对应的安全功能需求的。因此,一个安全软件的三个环节,哪个出问题都不行。

(2) 安全性测试的评估

当做完安全性测试后,软件是否能够达到预期的安全程度呢?这是安全性测试人员最关心的问题,因此需要建立对测试后的安全性评估机制。一般从以下两个方面进行评估。①安

全性缺陷数据评估。如果发现软件的安全性缺陷和漏洞越多，可能遗留的缺陷也越多。进行这类评估时，必须建立基线数据作为参照，否则评估起来没有依据就无法得到正确的结论。

②采用漏洞植入法来进行评估。漏洞植入法和**可靠性测试**里的故障插入测试是同一道理，只不过这里是在软件里插入一些有安全隐患的问题。采用漏洞植入法时，先让不参加安全测试的特定人员在软件中预先植入一定数量的漏洞，最后测试完后看有多少植入的漏洞被发现，以此来评估软件的安全性测试做得是否充分。

(3) 采用安全测试技术和工具

可使用专业的具有特定功能的安全扫描软件来寻找潜在的漏洞，将已经发生的缺陷纳入缺陷库，然后通过**自动化测试**方法来使用自动化缺陷库进行轰炸测试。例如，使用一些能够模拟各种攻击的软件来进行测试。

安全测试是用来验证集成在软件内的保护机制是否能够在实际中保护系统免受非法的侵入。一句通俗的话说：软件系统的安全当然必须能够经受住正面的攻击——但是它也必须能够经受住侧面的和背后的攻击。

识别常见 Web 漏洞 有效防止入侵

在 Internet 大众化及 Web 技术飞速演变的今天，在线**安全**所面临的挑战日益严峻。伴随着在线信息和服务的可用性的提升，以及基于 Web 的攻击和破坏的增长，安全风险达到了前所未有的高度。由于众多安全工作集中在**网络**本身上面，Web 应用程序几乎被遗忘了。也许这是因为应用程序过去常常是在一台计算机上运行的独立程序，如果这台计算机安全的话，那么应用程序就是安全的。如今，情况大不一样了，Web 应用程序在多种不同的机器上运行：

客户端、Web **服务器**、**数据库**服务器和应用服务器。而且，因为他们一般可以让所有的人使用，所以这些应用程序成为了众多攻击活动的后台旁路。

由于 Web 服务器提供了几种不同的方式将请求转发给应用服务器，并将修改过的或新的网页发回给最终用户，这使得非法闯入网络变得更加容易。

而且，许多**程序员**不知道如何开发安全的应用程序。他们的经验也许是开发独立应用程序或 Intranet Web 应用程序，这些应用程序没有考虑到在安全**缺陷**被利用时可能会出现灾难性后果。

其次，许多 Web 应用程序容易受到通过服务器、应用程序和内部已开发的代码进行的攻击。这些攻击行动直接通过了周边防火墙安全措施，因为端口 80 或 443（SSL，安全套接字协议层）必须开放，以便让应用程序正常运行。Web 应用程序攻击包括对应用程序本身的 DoS（拒绝服务）攻击、改变网页内容以及盗走企业的关键信息或用户信息等。

总之，Web 应用攻击之所以与其他攻击不同，是因为它们很难被发现，而且可能来自任何在线用户，甚至是经过验证的用户。迄今为止，该方面尚未受到重视，因为企业用户主要使用防火墙和**入侵检测解决方案**来保护其网络的安全，而防火墙和入侵检测解决方案发现不了 Web 攻击行动。

常见的 Web 应用安全漏洞

下面将列出一系列通常会出现的安全漏洞并且简单解释一下这些漏洞是如何产生的。

已知弱点和错误配置

已知弱点包括 Web 应用使用的操作系统和第三方应用程序中的所有程序错误或者可以被利用的漏洞。这个问题也涉及到错误配置，包含有不安全的默认设置或管理员没有进行安

全配置的应用程序。一个很好的例子就是你的 Web 服务器被配置成可以让任何用户从系统上的任何目录路径通过，这样可能会导致泄露存储在 Web 服务器上的一些敏感信息，如口令、源代码或客户信息等。

隐藏字段

在许多应用中，隐藏的 HTML 格式字段被用来保存系统口令或商品价格。尽管其名称如此，但这些字段并不是很隐蔽的，任何在网页上执行“查看源代码”的人都能看见。许多 Web 应用允许恶意的用户修改 HTML 源文件中的这些字段，为他们提供了以极小成本或无需成本购买商品的机会。这些攻击行动之所以成功，是因为大多数应用没有对返回网页进行验证；相反，它们认为输入数据和输出数据是一样的。

后门和调试漏洞

开发人员常常建立一些后门并依靠调试来排除应用程序的故障。在开发过程中这样做可以，但这些安全漏洞经常被留在一些放在 Internet 上的最终应用中。一些常见的后门使用户不用口令就可以登录或者访问允许直接进行应用配置的特殊 URL。

跨站点脚本编写

一般来说，跨站点编写脚本是将代码插入由另一个源发送的网页之中的过程。利用跨站点编写脚本的一种方式是通过 HTML 格式，将信息帖到公告牌上就是跨站点脚本编写的一个很好范例。恶意的用户会在公告牌上帖上包含有恶意的 JavaScript 代码的信息。当用户查看这个公告牌时，服务器就会发送 HTML 与这个恶意的用户代码一起显示。客户端的浏览器会执行该代码，因为它认为这是来自 Web 服务器的有效代码。

参数篡改

参数篡改包括操纵 URL 字符串，以检索用户以其他方式得不到的信息。访问 Web 应用的后端数据库是通过常常包含在 URL 中的 **SQL** 调用来进行的。恶意的用户可以操纵 SQL 代码，以便将来有可能检索一份包含所有用户、口令、信用卡号的清单或者储存在数据库中的任何其他数据。

更改 cookie

更改 cookie 指的是修改存储在 cookie 中的数据。网站常常将一些包括用户 ID、口令、帐号等的 cookie 存储到用户系统上。通过改变这些值，恶意的用户就可以访问不属于他们的帐户。攻击者也可以窃取用户的 cookie 并访问用户的帐户，而不必输入 ID 和口令或进行其他验证。

输入信息控制

输入信息检查包括能够通过控制由 CGI 脚本处理的 HTML 格式中的输入信息来运行系统命令。例如，使用 CGI 脚本向另一个用户发送信息的形式可以被攻击者控制来将服务器的口令文件邮寄给恶意的用户或者删除系统上的所有文件。

缓冲区溢出

缓冲区溢出是恶意的用户向服务器发送大量数据以使系统瘫痪的典型攻击手段。该系统包括存储这些数据的预置缓冲区。如果所收到的数据量大于缓冲区，则部分数据就会溢出到堆栈中。如果这些数据是代码，系统随后就会执行溢出到堆栈上的任何代码。Web 应用缓冲区溢出攻击的典型例子也涉及到 HTML 文件。如果 HTML 文件上的一个字段中的数据足够的大，它就能创建一个缓冲器溢出条件。

直接访问浏览

直接访问浏览指直接访问应该需要验证的网页。没有正确配置的 Web 应用程序可以让恶意的用户直接访问包括有敏感信息的 URL 或者使提供收费网页的公司丧失收入。

Web 应用安全两步走

Web 应用攻击能够给企业的财产、资源和声誉造成重大破坏。虽然 Web 应用增加了企业受攻击的危险，但有许多方法可以帮助减轻这一危险。首先，必须教育开发人员了解安全编码方法。仅此项步骤就会消除大部分 Web 应用的安全问题。其次，坚持跟上所有厂商的最新安全补丁程序。如果不对已知的缺陷进行修补，和特洛伊木马一样，攻击者就能很容易地利用你的 Web 应用程序穿过防火墙访问 Web 服务器、数据库服务器、应用服务器等等。将这两项步骤结合起来，就会大大减少 Web 应用受到攻击的风险。同时管理人员必须采取严格措施，以保证不让任何东西从这些中溜过去

Web 安全测试计划

任何一个测试的开始都要制定一个完整的**测试计划**，现在我们就从 **web 安全测试** 的测试计划开始

要做一个测试计划首先要明确**测试需求**。在写测试计划之前必须要明确测试需求，

暗含的要求：例如很少看到这样明确话的文档要求：“入侵这相应手册中不许有拼写错误”但同时有些组织是允许拼写错误存在的。这样暗含的要求我们就要明确，可以通过和主管部门或是用户沟通来明确这样的要求。

不完全的或模糊的要求

比如：“所有的网站都应该安装 **SP3** 补丁”这样的要求就是模糊不清的，因为没有指明是操作系统还是网站服务软件，或是某些具体的系统软件。这样的需求就应该有需求提出的人来明确，

确定在什么系统上安装 sp3 补丁。

未指明的要求

如：“必须使用强密码”看起来还向没什么问题，但从测试观点看，设么是强密码呢？是常超过 7 个字符的，还是应该有大小写的。这样的要求我们就应该根据密码要求标准具体化，比如：要求密码加强必须大于 7 个字符。

笼统的要求

比如“站点必须是安全的”尽管每个人都会同意这个要求，但展点能够彻底安全的唯一方法就是，断开展点的所有连接，内网的外网的，然后锁在一个加了封条的屋子里。但是这并不是要求的本意。这样的要求应该具体化，制定要求达到的安全程度。

好了要求明确了，下面就说一下就话的结构。呵呵我也是学来的，照别人的说吧，也有我的体会

测试计划的结构

测试计划可以依照工业标准（例如软件文档标准——Std.829）组织，也可以基于内部模版，甚至可以用创献礼的全新个是编排。但大家一定要注意一点，测试计划重要的不是个是而是创建测试计划的过程一定要获得测试组的认可。但有的测试必须用规格的测试计划格式，行业内部模版或行业标准，这样的测试如：政府机构、保险承销商等。

测试计划可以长达几百页，也可以简单的只有一张纸，关键测试计划必须实用，也不必要把大量的人力和物理花费在测试计划上，要根据具体情况来确定。

根据 IEEE Std.829-1998(软件测试文档标准 1998 年修订版)来介绍测试计划的内容

1.测试计划标题

就是说没个测试计划和每个测试计划的版本都应该有一个公司内部的独一无二标示,这也是文档控制和**版本控制**的基本要求，我觉得在正规公司的同仁们都应该明白。

2.介绍

这一部分适度测试的一个总的概括,通过这一部分一该让读者明白此项目的准确目标和测试组如何达到这些目标。根据情况也可以做一些基本概念的解释,比如为什么要做安全测试等等。

3 项目范围

在这一部分中明确项目的测试目标,如果在介绍中已经描述的测试目标的话在这一部分应该详尽的介绍测试目标。同时在这一部分可以列出在测试中不设计的测试项。

4 变动控制过程

这一部分主要是解决再测试中如果有需要变动的测试项应做如何处理,可参考 **CCB**(变动控制委员会)的意见进行适当的变动。

5 待测的特性(还没吃饭呢,同志们现在不写了好吗,等明天在写吧,好了写玩这一部分!坚持)

这一部分应该是对测试对象的描述,测试组应该对则是对象进行研究,对测试对象进行可行性检查。因该根据具体情况对测试项进行删改,比如:应该确定是否有足够的时间和资金来测试每一样特性。测试组极有可能没有得到想要的足够资源,在这种情况下,必须做出决定应重点测试哪些方面,而那些方面可以相对简单。达成这一点的方式是使用风险分析。(好了不行了,饿晕了,等有时间在写吧)未完待续

6 不测的特性

这一部分主要说明因为测试项目多,可能测试的过程中有的重要的测是项目可能会被忽略。

“因为在测试计划的各自测试范围拼合的不是很紧密,可能出现系统的某个特性就完全没有测试,因为企业里的每个人都以为其他人会测试系统的这个方面。”解决的办法就是:不仅在某个测试计划中记录下什么项将被测试,也纪录下这些项的哪些方面将会测试而那些将是在测试计划范围之外的。从而明确澄清各个测试计划的范围会涉及到什么和不涉及什么。一句话就是在测试计划重要把这些都确定下来,不能含糊不清。

7 方法

测试计划的这部分一般用来阐明测试组达成选前确定的测试目标所用的策略。无需深入到每个测试策略决定的详尽细节。但是应该明确主要的决定。例如：将执行什么层级的测试和在系统生命周期内何时执行测试。下面对一些概念进行介绍：

```
<!--[if !supportLists]-->1) ant: normal; font-weight: normal; font-size: 7pt; line-height: normal; font-size-adjust: none; font-stretch: normal;"> <!--[endif]-->测试的层级
```

测试分为多测试阶段（或测试层）的一个策略是按照某个测试可运行前系统必须完成的测试程度来划分测试。比如一个测试可以分为**单元测试**（在系统一个组件上单独进行的测试也可称为模块测试。）、整合级、串级或连级测试（设计用来测试系统的两个或个多组件见的通信的测试。）、**系统测试**等。根据测试的具体情况将一个或几个层写到一个测试计划中。

```
<!--[if !supportLists]-->2) <!--[endif]-->何时测试
```

这一部分解决测试应该在何时进行，反对以前的那种当软件设计完成后才进行的安全测试，软件测试应该贯穿整个软件开发周期。

```
<!--[if !supportLists]-->3) <!--[endif]-->何时再测试
```

这个概念是说只要系统在运行安全测试就应该不断地进行测试，测试的频率由执行这些测试的资源的可利用性及系统随时间变动的程度来决定。因为系统完成开发运行后可能出现新的安全隐患如：以前所用的操作系统的一个未知的漏洞现在被黑客团体知道了；系统增加了一些复述设备（防火墙，**服务器**，路由器）以使之符合更高的使用要求等等。

```
<!--[if !supportLists]-->4) <!--[endif]-->再测试什么
```

这里的再测试就是软件测试中的**回归测试**，测试在前期**测试过程**中出现的 **bug** 是否已经修复。测试是否出现新的安全风险等等。

8 通过/未通过的标准

在安全测试中通过/未同过的结果比较难下，建议在测试之前先对测试的预期结果过进行文档化操作。最好这个结果应该由负责做出决策的人来定，而不是测试组，测试组需要做的应该是怎么把测试结果提交给负责做出决策的人。（建议测试组提交一份通过测试结果评估出来的系统运行后可能出现的安全风险的报告，而不是自动化检测工具检测出来的安全漏洞的列表）。如果必须有测试组指明通过不同过的标准最好用这样的标示方法：“有信息安全经理决定网站中全部探测到的/或严重部分是否需要返工或再测试”而不应该用“通过 95%的**测试用例**系统可视为系统通过”

9 暂停标准或重测

测试计划这一部分可用来标明在何种情况下可以谨慎的暂停整个（或部分）测试工作及因此要达到什么样要求才能恢复暂停测试。例如：建议在主要网站服务器上的操作系统计划要安装最新的补丁包之前，不要运行渗透性测试。相反如果暂停等到服务器上的操作系统安装了最新的补丁，并重新进行了配置，就可以重新进行测试了。

10 测试交付物

在这一部分应该记录在安全测试工作中应该交付的测试结果文 件。在安全测试结果中应该交付如下这些文件

```
<!--[if !supportLists]-->1) <!--[endif]-->测试日志
```

以时间顺序记录测试执行中发生的事件。

```
<!--[if !supportLists]-->2) <!--[endif]-->测试事件报告
```

测试事件报告是在测试过程中出现的一些问题的报告，比如测试过程中重出现的一些系统的错误信息等等，在测试过程中可以由测试成员以观察报告的形式记录下来，由最终出报告的人来进行分析。

```
<!--[if !supportLists]-->3) <!--[endif]-->漏洞追查报告
```

就是用自动化工具检查出来的漏洞结果。

```
<!--[if !supportLists]-->4) <!--[endif]-->度量
```

就是衡量一个事物的单位，比如每天出现 15 个漏洞等等。

```
<!--[if !supportLists]-->5) <!--[endif]-->测试总结报告
```

就是总结在测试工作中所找到的一切东西。

11 环境需求

在测试计划的这一部分主要描述测试所需要的环境。和在环境中需要一些什么设施。

12 配置管理

配置管理是及时的以离散点标识（什么）、控制（库管理）、追查（谁和何时）和报告（谁需要知道）系统组件的过程，其主要目的是为了维护系统的完整性。

13 责任

在测试计划中这一部分主要是确定，某些事情应该由谁来负责协调，或某些人应该负责什么等。就是说在这一部分应该确定部门责任。如公司层 负责镜网站的物理**安全评估**；部门层：负责安装配置**测试环境**等等。

14 提供人员和**培训**需要

测试计划在这一部分要求计划制定者考虑自己测试组的能力，如果需要可以进行人员培训计划。

15 测试进度

在测试计划的这一部分主要是制定测试每一个阶段需要花费的时间，进度控制等。对于打得测试项目可以考虑单独的作为一个文档来做这个测试进度可以应用一种进度控制工具(Easy Schedule Maker 等) 进行管理，进度的细节可以在其他的计划文档中详细描述。

16 预计风险和应急措施

在测试计划的这一部分应该有计划的制定者考虑到在测试中可能遇到的风险，以及遇到这种风险的解决办法，以条款的形式列出。做一个风险百分比图，通过这个图可以知道在测试中什么风险占的比例最大，再测试中应该进行避免这种风险的出现。

17 审批

在测试计划这一部分就是确定评审组的成员，既由那些人来评审你的测试计划，有哪些人来评审你的测文档等。可能不同的公司需要的评审程度不一样，根据自己公司的具体需要来定。但前面提到的粮店是必不可上的。

小结：

无论选择测试计划的格式是基于工业标准， 还是内部模版，或是为具体项目配置的独特模版，其测试计划和相关文档一定要经过修订以确保其充分说明了下表测试计划要考虑的因素：

测试计划考虑因素列表

是	否	描述
		是否系统的安全要求以被澄清并做了文档化操作
		是否已经明确确定以了测试工作的目标（及其范围）
		是否标明了所有待测项（及其版本）
		是否未列出重要的不测项
		是否定以了变动控制过程并标明了那些有权限批准测试范围变动的人
		是否标明了所有待册特性
		是否未列出重要的不测特性
		是否以对 测试方法 （策略）进行文档化操作
		是否以将把系统视为通过的标准（如果有）进行文档化操作
		是否以将测试终止（和恢复）的标准（如果有）进行文档化操作
		是否以将测试工作要产生的文档进行文档化操作

		是否研究过将测试工作所有的环境需要进行文档化操作
		是否将待测项的配置管理策略进行文档化操作
		是否以将 测试脚本 和测试数据（测试套件）的配置管理策略进行文档化操作
		是否以指派了所有测试工作的责任
		是否以指派了所有测试工作所依赖的责任
		是否标明了人事需要的来源
		是否标明了培训需要的来源
		是否创建了测试进度
		是否以考虑了项目圆满结束的必须步骤
		是否以标明了预计的最严重的风险
		是否以设计并通过了预计的最严重的风险的规避风险措施
		是否以将所有的问题、假设、约束和依赖进行文档化操作
		是否以定义了所有的不常见的简写和术语
		是否以标明并互相引用了支持文档

		是否以标明了负责批准计划的人
		是否以标明了负责接收测试结果的人
		是否以标明了需要通报测试工作进展的人



Web 安全测试的步骤

安全测试方面应该参照 spi 的 web 安全 top 10 来进行。

目前做软件测试人员可能对安全性测试了解不够，测试结果不是很好。如果经验不足，测试过程中可以采用一些较专业的 web 安全测试工具，如 WebInspect、Acunetix.Web.Vulnerability.Scanner 等，不过自动化 web 安全测试的最大缺陷就是误报太多，需要认为审核测试结果，对报告进行逐项手工检测核对。

对于 web 安全的测试用例，可以参照 top 10 来写，如果写一个详细的测试用例，还是比较麻烦的，建议采用安全界常用的 web 渗透报告结合 top10 来写就可以了。

现在有专门做系统和网站安全检测的公司，那里做安全检测的人的技术都很好，大多都是红客。

再补充点，网站即使站点不接受信用卡支付，安全问题也是非常重要的。Web 站点收集的用户资料只能在公司内部使用。如果用户信息被黑客泄露，客户在进行交易时，就不会有安全感。

目录设置

Web 安全的第一步就是正确设置目录。每个目录下应该有 index.html 或 main.html 页面，这样就不会显示该目录下的所有内容。我服务的一个公司没有执行这条规则。我选中一幅图片，单击鼠标右键，找到该图片所在的路径"...com/objects/images".然后在浏览器地址栏中手工输入该路径，发现该站点所有图片的列表。这可能没什么关系。我进入下一级目录 "...com/objects"，点击 jackpot.在该目录下有很多资料，其中引起我注意的是已过期页面。该公司每个月都要更改产品价格，并且保存过期页面。我翻看了一下这些记录，就可以估计他们的边际利润以及他们为了争取一个合同还有多大的降价空间。如果某个客户在谈判之前查看了这些信息，他们在谈判桌上肯定处于上风。

SSL

很多站点使用 **SSL** 进行安全传送。你知道你进入一个 **SSL** 站点是因为浏览器出现了警告消息，而且在地址栏中的 **HTTP** 变成 **HTTPS**。如果开发部门使用了 **SSL**，测试人员需要确定是否有相应的替代页面（适用于 **3.0** 以下版本的浏览器，这些浏览器不支持 **SSL**。当用户进入或离开安全站点的时候，请确认有相应的提示信息。是否有连接时间限制？超过限制时间后出现什么情况？

登录

有些站点需要用户进行登录，以验证他们的身份。这样对用户是方便的，他们不需要每次都输入个人资料。你需要验证系统阻止非法的用户名/口令登录，而能够通过有效登录。用户登录是否有次数限制？是否限制从某些 **IP** 地址登录？如果允许登录失败的次数为 **3**，你在第三次登录的时候输入正确的用户名和口令，能通过验证吗？口令选择有规则限制吗？

日志文件

在后台，要注意验证服务器日志工作正常。日志是否记所有的事务处理？是否记录失败的注册企图？是否记录被盗信用卡的使用？是否在每次事务完成的时候都进行保存？记录 **IP** 地址吗？记录用户名吗？

脚本语言脚本语言是常见的安全隐患。每种语言的细节有所不同。有些脚本允许访问根目录。其他只允许访问邮件服务器，但是经验丰富的黑客可以将服务器用户名和口令发送给他们自己。找出站点使用。

实战 Web 安全测试之 HTTP 截断(走私漏洞篇)

在本文中，我们将详细为读者介绍针对 **HTTP** 截断和 **HTTP** 走私攻击的安全测试技术。我们将通过实例演示如何利用 **HTTP** 协议的某些特性，或者利用 **Web** 应用程序的弱点或者不同代理对 **HTTP** 消息的解释也不相同的特点来发动这两种攻击。

一、HTTP 截断/走私漏洞概述

本文遏制，我们将分析针对特定的 **HTTP** 报头的两种不同的攻击技术：**HTTP** 截断和 **HTTP** 走私攻击。对于 **HTTP** 截断攻击而言，它是利用了缺乏输入消毒措施的漏洞，该漏洞允许入侵者向应用程序响应的头部插入 **CR** 和 **LF** 字符，从而将响应分割为两个不同的 **HTTP** 消息。该攻击的目标既不同于缓存投毒，也区别于跨站脚本攻击。对于第二种攻击方法，攻击者利用了这样的一个事实，即一些专门精心制作的 **HTTP** 消息可能会随着接收它们的代

理的不同，而作不同的解析和解释。**HTTP** 走私技术要求对处理 **HTTP** 消息的各种代理相当熟悉，否则无法发动这种攻击。

下面我们介绍这些漏洞的黑盒测试和灰盒测试技术。

二、HTTP 截断攻击黑盒测试

一些 **web** 应用程序会使用部分用户输入来生成它们的响应头部的某些值，这方面最简单的例子就是重定向了，因为目标 **URL** 依赖于用户提交的某些值。举例来说，假如用户被要求在标准 **web** 接口和高级 **web** 接口之间做出选择，然后，选择的结果将作为一个参数传递，并且这个参数将用于触发重定向到相应的页面的应答头中。更确切地说，如果该参数 **interface** 的值是 **advanced**，那么应用程序将响应下列内容：

```
HTTP/1.1 302 Moved Temporarily
Date: Sun, 03 Dec 2009 16:22:19 GMT
Location: http://victim.com/main.jsp?interface=advanced
<snip>
```


 技术成就梦想

图 1

收到这个消息后，浏览器会把用户引向 **Location** 头部规定的页面。然而，如果应用程序没有对用户输入进行过滤的话，攻击者就可以在参数 **interface** 中插入序列 **%0d%0a**，而该序列代表的则是用于分割各行的 **CRLF**(回车换行)序列。这样一来，攻击者将能够触发一个响应，重要的是任何解析器(例如介于用户和 **Web** 应用之间的 **web** 缓存)都会把这个响应会被解释为两个不同的响应。所以，攻击者就可以通过给这个 **web** 缓存“投毒”以使它为后续的请求中提供虚假的内容。例如，在我们前面的例子中，假设攻击者将下列内容作为参数 **interface** 进行传递：

```
advanced%0d%0aContent-Length:%20%0d%0a%0d%0aHTTP/1.1%20200%20OK%0d%0aContent-
Type:%20text/html%0d%0aContent-Length:%2035%0d%0a%0d%0a<html>Sorry,%20System%20Down</html>
```

图 2

从存在漏洞的软件(也就是没有对用户输入进行严格消毒的应用程序)中得到的响应将是下面的内容:

```
HTTP/1.1 302 Moved Temporarily
Date: Sun, 03 Dec 2005 16:22:19 GMT
Location: http://victim.com/main.jsp?interface=advanced
Content-Length: 0

HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 35

<html>Sorry,%20System%20Down</html>
<other data>
```

图 3

Web 缓存将看到两个不同的响应,因此如果攻击者发送第一个请求之后立即发送对 /index.html 页面的请求的话,web 缓存会认为这个请求与第二个响应相匹配,并缓存它的内容,这样一来后面经由 web 缓存的所有指向 victim.com/index.html 的请求都会收到系统故障消息,即“system down”。通过这种方式,一个攻击者将能有效涂改站点在使用 web 缓存的用户心中的形象,如果该 web 缓存是该 Web 应用程序的一个反向代理的话,那么这个 Web 应用程序在整个因特网中的用户都会受到影响。另外,攻击者还可以向这些用户传输发动跨站脚本攻击攻击的 JavaScript 代码片断,例如窃取 cookies 等。需要注意的是,虽然该安全漏洞位于应用程序中,但是攻击针对的对象却是使用该应用程序的用户。

因此,为了查找这个安全漏洞,渗透测试人员需要识别所有能够影响响应的一个或多个头部的用户输入,并检查用户是否能够在其中注入一个 **CR+LF** 序列。与这个攻击关系最密切的两个头部是:

Location 和 Set-Cookie

需要注意的是,现实中要想成功利用这个安全漏洞可能是件非常复杂的事情,因为有多种因素必须考虑到:

- 1.攻击者要想让伪造的响应被缓存的话,必须正确设置其中的各个头部,例如 **Last-Modified** 头部的值必须设为将来的一个时间。此外,攻击者还必须破坏目标页面先前的缓存版本,方法是提交一个请求头部中带有“**Pragma: no-cache**”的前导请求来防止页面被缓存。
2. 即使应用程序没有过滤 **CR+LF** 序列,但是仍可能过滤了发动该攻击所需的其他字符,例如字符<和>等。这时候,攻击者可以尝试使用其他编码,例如 **UTF-7** 编码等。
3. 某些攻击目标(例如 **ASP**)会对 **Location** 头部(例如 **www.victim.com/redirect.asp**)中的路径部分进行 **URL** 编码处理,这样就使得 **CRLF** 序列不起作用了。然而,它们却不能对查询部分(例如**?interface=advanced**)进行这样的编码处理,这意味着放置一个前导问号就能够绕过这种过滤技术。

三、HTTP 截断攻击灰盒测试

对于 **Web** 应用程序即攻击目标的深入了解,在发动 **HTTP** 截断攻击时是极其有帮助的。例如,不同的攻击目标可能使用不同的方法来判定第一个 **HTTP** 消息在何时终止,第二个 **HTTP** 消息从什么时候开始。当然,有时候可以使用消息边界来进行判定,如上面的例子就是这样。但是,其他的攻击目标可能使用不同的数据包来携带不同的消息。此外,有些目标会为每个消息分配指定组块的数量,这种情况下,第二个消息必须从特定的块开始,这就要求攻击者在两个消息之间填充必要的块。不过,当时有 **URL** 发送有弱点的参数的时候,这么做可能会引起一些麻烦,因为过长的 **URL** 很可能被截断或者过滤掉。灰盒测试可以帮助攻击者找到解决办法:一些应用程序服务器允许使用 **POST** 而非 **GET** 来发送请求。

四、HTTP 走私攻击灰盒测试

之前讲过，HTTP 走私攻击所利用的漏洞是，某些精心构造的 HTTP 消息会随不同的代理(浏览器、web 缓存、应用程序防火墙)而做不同的解释。这种攻击方法是由 Chaim Linhart、Amit Klein、Ronen Heled 和 Steve Orrin 于 2005 年发现的。这种攻击有多种用途，我们这里仅介绍最雷人的一种：绕过应用程序防火墙。

下面，我们详细介绍利用 HTTP 走私攻击绕过应用程序防火墙的详细方法。有许多应用防火墙都能根据一些已知的嵌入请求的恶意模式来检测和阻止怀有恶意的 web 请求。举例来说，针对 IIS 服务器的 Unicode 目录遍历攻击能够通过提交一个类似下面所示的一个请求发动攻击：

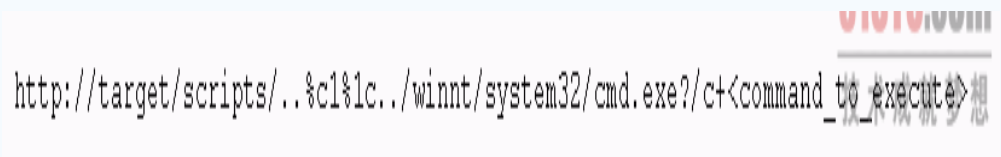


图 4

当然，通过检查 URL 是否存在类似“..”和“cmd.exe”的字符串可以很容易地检测和过滤掉这种攻击。然而，IIS 5.0 对于 POST 请求主体的长度是有要求的，最多为 48K 字节——当 Content-Type 头部不同于 application/x-www-form-urlencoded 时，超出该限制的部分会被全部截断。攻击者可以利用这一点来创建一个很大的请求，如下所示：

```

POST /target.asp HTTP/1.1      <-- Request #1
Host: target
Connection: Keep-Alive
Content-Length: 49225
<CRLF>
<49152 bytes of garbage>
POST /target.asp HTTP/1.0      <-- Request #2
Connection: Keep-Alive
Content-Length: 33
<CRLF>
POST /target.asp HTTP/1.0      <-- Request #3
xxxx: POST /scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir HTTP/1.0  <-- Request #4
Connection: Keep-Alive
<CRLF>

```

图 5

这里，Request #1 由 49223 字节内容组成，所以 Request #1 还包含了 Request #2 的几行内容。因此，防火墙(或者其他任何代理)会看到 Request #1，但是却无法看到 Request #2(它的数据正好是#1 请求的一部分)，能够看到 Request #3 却会遗漏 Request #4(因为该 POST 正好是伪造的头部 xxxx 部分)。现在，IIS 5.0 将会发生什么情况？它将在 49152 字节无用信息之后停止对 Request #1 的解析，因为这已经达到了 48K=49152 字节的限制，并将 Request #2 解析为一个新的、单独的请求。Request #2 声称它的内容为 33 字节，包括 xxxx: 之前的所有内容，这使得 IIS 会漏掉 Request #3，因为 Request #3 被解释为 Request #2 的一部分，但是 IIS 会认出 Request #4，因为它的 POST 是从 Request #2 的第 33 个字节之后开始的。当然，这看起来有些复杂，但是却很好的解释了为什么该攻击性 URL 不会被防火墙发现，却能被 IIS 正确解析和执行的原因所在。

在上面的情形中，虽然我们利用的是 web 服务器中的安全漏洞，但是在其他情形中，我们可以通过利用各种支持 HTTP 的设备在解析不兼容 1005 RFC 的消息时所采取的方式各不相同来发动攻击。举例来说，HTTP 协议只允许一个 Content-Length 头部，但是却规定如何处理具有两个 Content-Length 头部的消息。一些实现将使用第一个，而另一些实现则使用第二个，这种情况下就很容易遭到 HTTP 走私的攻击。另一个例子是 GET 消息中 Content-Length 头部的使用。

五、小结

在本文中,我们详细为读者介绍了针对 HTTP 截断和 HTTP 走私攻击的安全测试技术。我们通过实例演示了如何利用 HTTP 协议的某些特性,或者利用 Web 应用程序的弱点或者不同代理对 HTTP 消息的解释也不相同的特点来发动这两种攻击。希望本文对读者的安全测试工作有所帮助。

安全测试

即使站点不接受信用卡支付,安全问题也是非常重要的。Web 站点收集的用户资料只能在公司内部使用。如果用户信息被黑客泄露,客户在进行交易时,就不会有安全感。

5.1 目录设置

Web 安全的第一步就是正确设置目录。每个目录下应该有 index.html 或 main.html 页面,这样就不会显示该目录下的所有内容。我服务的一个公司没有执行这条规则。我选中一幅图片,单击鼠标右键,找到该图片所在的路径“...com/objects/images”。然后在浏览器地址栏中手工输入该路径,发现该站点所有图片的列表。这可能没什么关系。我进入下一级目录“...com/objects”,点击 jackpot。在该目录下有很多资料,其中引起我注意的是已过期页面。该公司每个月都要更改产品价格,并且保存过期页面。我翻看了一下这些记录,就可以估计他们的边际利润以及他们为了争取一个合同还有多大的降价空间。如果某个客户在谈判之前查看了这些信息,他们在谈判桌上肯定处于上风。

5.2 SSL

很多站点使用 SSL 进行安全传送。你知道你进入一个 SSL 站点是因为浏览器出现了警告消息,而且在地址栏中的 HTTP 变成 HTTPS。如果开发部门使用了 SSL,测试人员需要确定是否有相应的替代页面(适用于 3.0 以下版本的浏览器,这些浏览器不支持 SSL。当用户进入或离开安全站点的时候,请确认有相应的提示信息。是否有连接时间限制?超过限制时间后出现什么情况?

5.3 登录

有些站点需要用户进行登录,以验证他们的身份。这样对用户是方便的,他们不需要每次都输入个人资料。你需要验证系统阻止非法的用户名/口令登录,而能够通过有效登录。用户登录是否有次数限制?是否限制从某些 IP 地址登录?如果允许登录失败的次数为 3,你在第三次登录的时候输入正确的用户名和口令,能通过验证吗?口令选择有规则限制吗?是否可以不登陆而直接浏览某个页面?

Web 应用系统是否有超时的限制，也就是说，用户登陆后在一定时间内（例如 15 分钟）没有点击任何页面，是否需要重新登陆才能正常使用。

5.4 日志文件

在后台，要注意验证服务器日志工作正常。日志是否记录所有的事务处理？是否记录失败的注册企图？是否记录被盗信用卡的使用？是否在每次事务完成的时候都进行保存？记录 IP 地址吗？记录用户名吗？

5.5 脚本语言

脚本语言是常见的安全隐患。每种语言的细节有所不同。有些脚本允许访问根目录。其他只允许访问邮件服务器，但是经验丰富的黑客可以将服务器用户名和口令发送给他们自己。找出站点使用了哪些脚本语言，并研究该语言的缺陷。还要需要测试没有经过授权，就不能在服务器端放置和编辑脚本的问题。**最好的办法是订阅一个讨论站点使用的脚本语言安全性的新闻组。**

WEB 安全测试通常要考虑的测试点

1、问题：没有被验证的输入

测试方法：

数据类型（字符串，整型，实数，等）

允许的字符集

最小和最大的长度

是否允许空输入

参数是否是必须的

重复是否允许

数值范围

特定的值（枚举型）

特定的模式（正则表达式）

2、问题：有问题的访问控制

测试方法：

主要用于需要验证用户身份以及权限的页面，复制该页面的 url 地址，关闭该页面以后，查看是否可以直接进入该复制好的地址

例：从一个页面链到另一个页面的间隙可以看到 URL 地址

直接输入该地址，可以看到自己没有权限的页面信息，

3、错误的认证和会话管理

例：对 Grid、Label、Tree view 类的输入框未作验证，输入的内容会按照 html 语法解析出来

4、缓冲区溢出

没有加密关键数据

例：view-source: http 地址可以查看源代码

在页面输入密码，页面显示的是 *****, 右键，查看源文件就可以看见刚才输入的密码，

5、拒绝服务

分析：攻击者可以从一个主机产生足够多的流量来耗尽很多应用程序，最终使程序陷入瘫痪。需要做负载均衡来对付。

6、不安全的配置管理

分析：Config 中的链接字符串以及用户信息，邮件，数据存储信息都需要加以保护

程序员应该作的：配置所有的安全机制，关掉所有不使用的服务，设置角色权限帐号，使用日志和警报。

分析：用户使用缓冲区溢出来破坏 web 应用程序的栈，通过发送特别编写的代码到 web 程序中，攻击者可以让 web 应用程序来执行任意代码。

7、注入式漏洞

例：一个验证用户登陆的页面，

如果使用的 sql 语句为：

```
Select * from table A where username=''+username+''' and pass word .....
```

Sql 输入 ‘or 1=1 —— 就可以不输入任何 password 进行攻击

或者是半角状态下的用户名与密码均为：‘or’‘=’

8、不恰当的异常处理

分析：程序在抛出异常的时候给出了比较详细的内部错误信息，暴露了不应该显示的执行细节，网站存在潜在漏洞，

9、不安全的存储

分析：帐号列表：系统不应该允许用户浏览到网站所有的帐号，如果必须要一个用户列表，推荐使用某种形式的假名（屏幕名）来指向实际的帐号。

浏览器缓存：认证和会话数据不应该作为 GET 的一部分来发送，应该使用 POST，

10、问题：跨站脚本（XSS）

分析：攻击者使用跨站脚本来发送恶意代码给没有发觉的用户，窃取他机器上的任意资料

测试方法：

- [HTML](#) 标签： <...>...</...>
- 转义字符： &(&); <(<); >(>); (空格) ;
- 脚本语言：

```
<script. language='javascript'>
```

```
...Alert('')
```

```
</script>
```

- 特殊字符： ‘ ’ < > /
- 最小和最大的长度
- 是否允许空输入