PDF-XChange
Click to buy NOW!
www.docu-track.com

WEB 应用系统安全开发规范

文档 ID	tech/web security
版本编号	v1.0
相关文档	
作者	
最后更新时间	

版本更新摘要			
版本号	日期	审阅人	更新摘要
v1.0	2011/07/25	高宏亮	Web 应用系统安全开发规范

目 录

PDF-XChange Click to buy NOW! www.docu-track.com

1. 目的

保障中网彩 WEB 应用平台的安全性，保证 WEB 应用系统的可用性，完整性和保密性。从安全角度规范 WEB 应用系统开发人员，能够让 WEB 应用开发者树立很强的安全意识，在开发过程中编写安全代码，进行安全编程。

2. 范围

本规范从应用安全开发角度出发，给出 WEB 应用系统安全开发的规范。供中网彩技术部内部使用，适用各个 WEB 应用系统项目开发的工作。

本规范定义了 WEB 应用系统安全开发和 WEB 编码安全相关的技术要求。

与 WEB 编码安全相关的内容包括：输入验证和输出编码的处理方法,跨站脚本攻击及解决方法、SQL 注入攻击及解决方法、恶意文件执行及解决方法、不安全的直接对象引用及解决方法、信息泄露和错误处理不当及解决方法、残缺的认证和会话管理及解决方法、不安全的通信及解决方法、限制 URL 访问失效的解决方法。

3.规范概述

当今电子商务时代，WEB应用系统为架构设计人员，开发人员提出一系列复杂的安全问题。最安全、最有能力抵御攻击的Web应用程序是那些应用安全思想构建的应用程序。

在设计初始阶段，应该使用可靠的安全体系结构和设计方法，同时要结合考虑应用程序的部署以及企业的安全策略。如果不能做到这一点，将导致在现有基础结构上部署应用程序时，要不可避免地危及网站系统的安全性。

本规范提供初步的安全体系结构和设计指南，并按照常见的应用程序漏洞类别进行组织。这些指南是Web应用程序安全的重要方面，并且是经常发生错误的领域。

4.安全编码原则

- 程序只实现你指定的功能
- 永不要信任用户输入，对用户输入数据做有效性检查
- 必须考虑意外情况并进行处理
- 不要试图在发现错误之后继续执行
- 尽可能使用安全函数进行编程
- 小心、认真、细致地编程

5. 安全背景知识

本规范主要提供设计应用程序时应该遵循的一些指南和原则。为充分理解本规范内容，请：

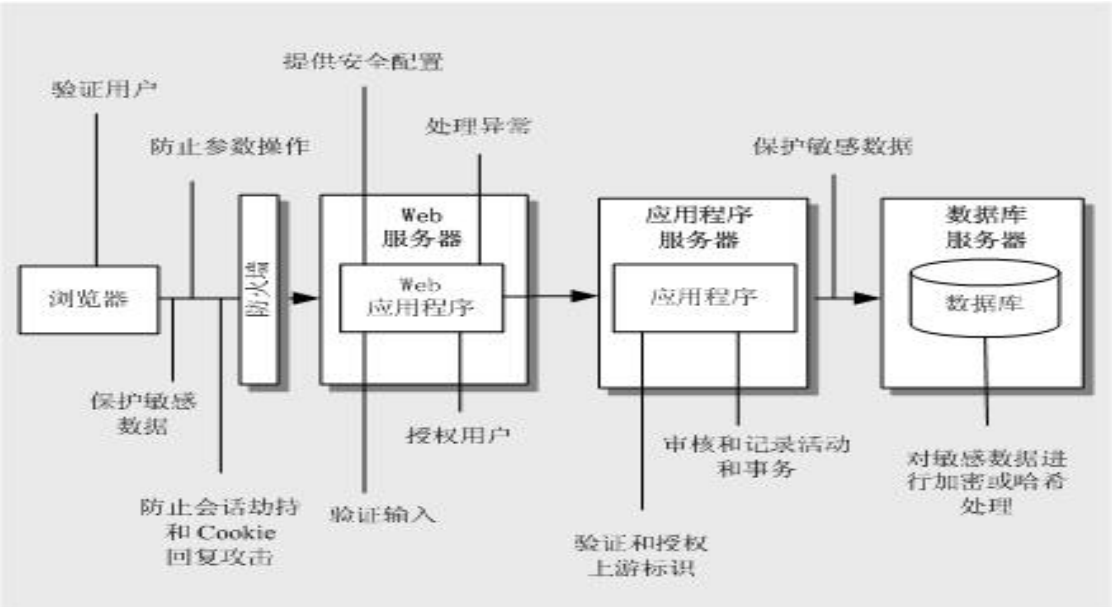
了解应用程序将会受到的威胁，以确保通过程序设计解决这些问题。了解需要考虑的威胁。在程序设计阶段应该考虑到这些威胁。

在应用程序易受攻击的重要环节应用系统的方法。将重点放在程序部署、输入验证、身份验证和授权、加密及数据敏感度、配置、会话、异常管理以及适当的审核和记录策略上，以确保应用程序的安全可靠性。

6.Web 应用程序的体系结构和设计问题

Web 应用程序向设计人员和开发人员提出了许多挑战。HTTP 是无国界的，这意味着跟踪每位用户的会话状态将成为应用程序的责任。作为先导者，应用程序必须能够通过某种形式的身份验证来识别用户。由于所有后续授权决策都要基于用户的标识，因此，身份验证过程必须是安全的，同样必须很好地保护用于跟踪已验证用户的会话处理机制。设计安全的身份验证和会话管理机制仅仅是Web 应用程序设计人员和开发人员所面临的众多问题中的两个方面。由于输入和输出数据要在公共网络上进行传输，因此还会存在其他挑战。防止参数操作和敏感数据泄漏是另外一些重要问题。

下图列出了安全设计方法中必须解决的其他重要问题。



Web 应用程序设计问题

下面的安全问题是根据应用程序漏洞类别描述的。实际经验表明，如果这些领域的设计存在薄弱环节，将会导致安全漏洞。下表列出了漏洞的类别，每个类别都突出显示了由于设计不当可能会导致的潜在问题。

漏洞类别	由于设计不当而引起的潜在问题
输入验证	嵌入到查询字符串、表单字段、cookie 和 HTTP 头中的恶意字符串的攻击。这些攻击包括命令执行、跨站点脚本(XSS)、SQL 注入和缓冲区溢出攻击。
身份验证	标识欺骗、密码破解、特权提升和未经授权的访问。
授权	访问保密数据或受限数据、篡改数据以及执行未经授权的操作。
配置管理	对管理界面进行未经授权的访问、具有更新配置数据的能力以及对用户帐户和帐户配置文件进行未经授权的访问。
敏感数据	泄露保密信息以及篡改数据。
会话管理	捕捉会话标识符，从而导致会话劫持及标识欺骗。
加密	访问保密数据或帐户凭据，或二者均能访问。
参数操作	路径遍历攻击、命令执行以及绕过访问控制机制，从而导致信息泄漏、特权提升和拒绝服务。
异常管理	拒绝服务和敏感的系统级详细信息的泄漏。
审核和记录	不能发现入侵迹象、不能验证用户操作，以及在诊断问题时出现困难。

7.输入验证

7.1 什么是输入

WEB 应用程序从各个方面获取输入,例如所有用户发送的，或者 web 应用程序与用户交互往返的数据 （用户提交的数据, cookie 信息,查询字符串参数等），以及后台数据 （数据库、配置数据和其他数据来源）。所有输入的数据都会在某种情况下影响请求的处理。

输入验证是一个很复杂的问题，并且是应用程序开发人员需要解决的首要问题。然而，正确的输入验证是防御目前应用程序攻击的最有效方法之一。正确的输入验证是防止 XSS、SQL 注入、缓冲区溢出和其他输入攻击的有效对策。

输入验证非常复杂，因为对于应用程序之间甚至应用程序内部的输入，其有效构成没有一个统一的答案。同样，对于恶意的输入也没有一个统一的定义。更困难的是， 应用程序对如何处理此输入将会影响应用的风险。例如，您是否存储用于其他应用程序的数据，或者应用程序是否接受来自其他应用程序所创建的数据源的输入?

以下做法可以增强 Web 应用程序的输入验证:

- 假定所有输入都是恶意的
- 集中方法

- 不要依赖客户端验证
- 注意标准化问题
- 限制，拒绝和净化输入

假定所有输入都是恶意的

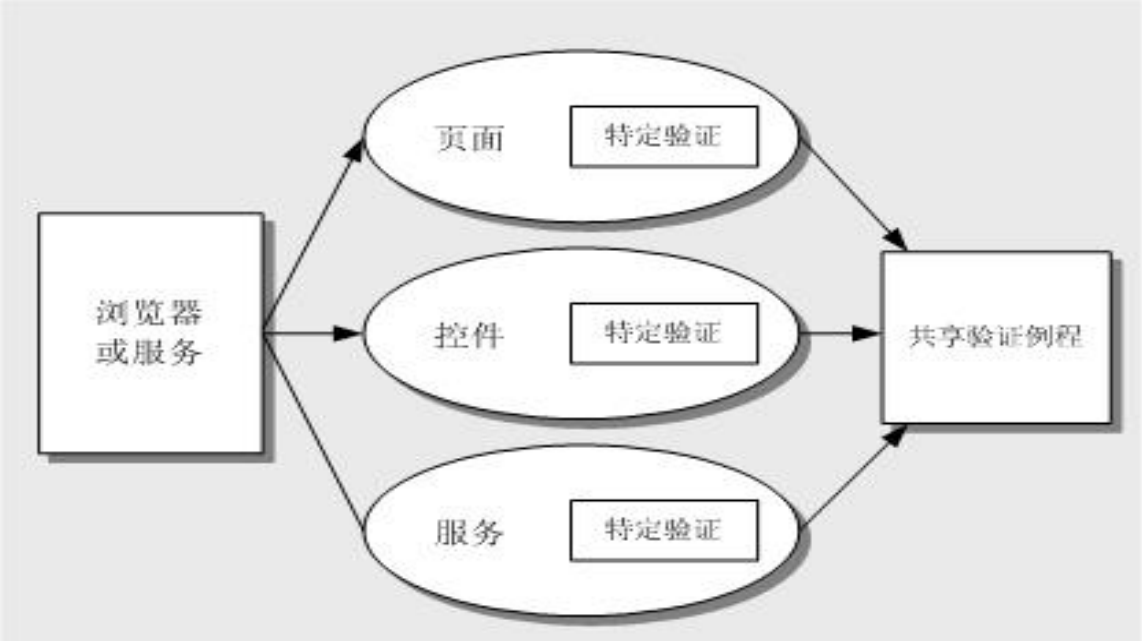
开始输入验证时，首先假定所有输入都是恶意的，除非有证据表明它们并无恶意。无论输入是来自服务、共享文 件、用户还是数据库，只要其来源不在可信任的范围之内，就应对输入进行验证。例如，如果调用返回字符串的外部 Web 服务，如何知道该服务不会执行恶意命令呢？另外，如果几个应用程序写入同一个共享数据库，您在读取数据时，如何知道该数据是否安全呢？

集中方法

将输入验证策略作为应用程序设计的核心元素。考虑集中式验证方法，例如，通过使用共享库中的公共验证和筛选代码。这可确保验证规则应用的一致性。此外，还能减少开发的工作量，且有助于以后的维护工作。

许多情况下，不同的字段要求不同的验证方法，例如，要求使用专门开发的常规表达式。但是，通常可以得到验证常用字段（如电子邮件地址、标题、名称、包括邮政编码在内的通讯地址，等等）的常规方法

下图显示了此验证方法：



不要依赖于客户端验证

应使用服务器端代码执行其自身的验证。如果攻击者绕过客户端或关闭客户端脚本例程（例如，通过禁用 JavaScript 脚本），后果如何？使用客户端验证可以减少客户端到服务器端的往返次数，但不要依赖这种方法进行安全验证。这是一个深入彻底的防御示例。

注意标准化问题

数据的标准形式是最标准、最简单的形式。标准化是指将数据转化为标准形式的过程。文件路径和 URL 尤其倾向于标准化问题，许多广为人知的漏洞利用都直接源自标准化缺陷。例如，请考虑以下字符串，它以标准形式表示了文件及其路径。

/www/public/testfile.jsp

以下字符串都指向同一个文件 testfile.jsp

/www/public/../root/testfile.jsp
/www/public///testfile.jsp
%2fwww%2fpublic%2f%2f%2ftestfile.jsp(url encode)
0x2f7777772f7075626c69632f7465737466696c652e6a7370(十六进制表示)

最后一个示例使用十六进制指定字符:

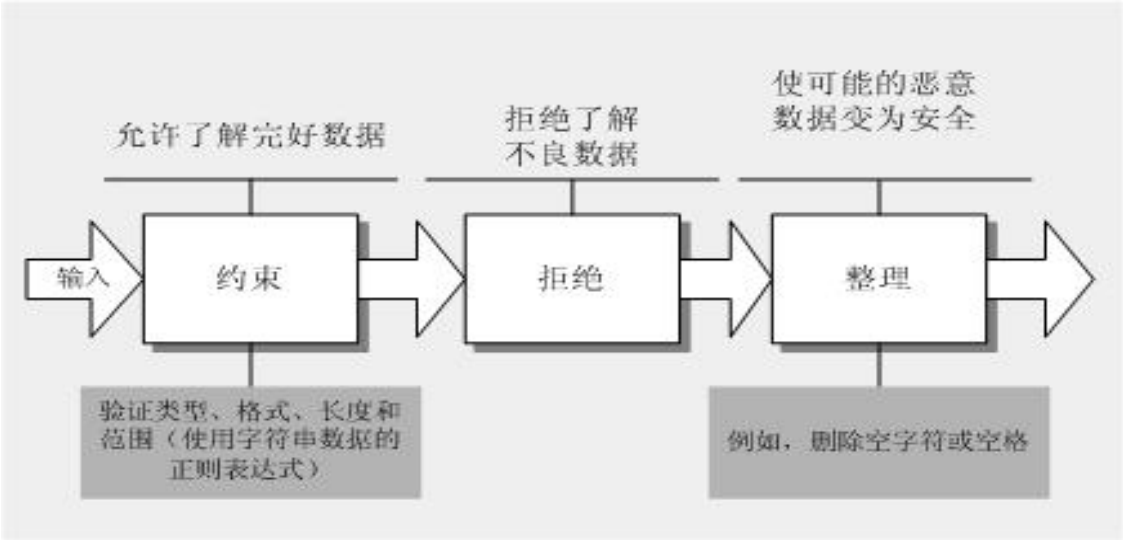
%2E 代表句号
%3A 代表冒号
%5C 代表反斜杠符号
%2f 代表斜杠

通常，应设法避免让应用程序接受用户输入的文件名，以防止标准化问题。可以考虑其他设计方法。例如，让应用程序为用户确定文件名。

如果确实需要用户输入文件名，在作出安全决策（如授予或拒绝对特定文件的访问权限）之前，应确保这些文件名具有严格定义的形式。

限制、拒绝和净化输入

输入验证的首选方法是从开始就限制允许输入的内容。按照已知的有效类型、模式和范围验证数据要比通过查找已 知有害字符的数据验证方法容易。设计应用程序时，应了解应用程序需要输入什么内容。与潜在的恶意输入相比，有效数据的范围通常是更为有限的集合。然而，为 了使防御更为彻底，您可能还希望拒绝已知的有害输入，然后净化输入。下图显示了我们推荐的策略。



限制输入

限制输入与允许输入有益数据类似。这是首选的输入验证方法。其思想是定义一个筛选器，根据类型、长度、格式和范围来筛选输入的数据。定义应用程序字段可以接受的数据输入，并强制应用该定义。拒绝一切有害数据。

限制输入可能包括在服务器上设置字符集，以便在本地建立输入的规范格式。

验证数据的类型、长度、格式和范围

在适当的地方对输入数据使用强类型检查，例如，在用于操作和处理输入数据的类中，以及在数据访问例程中。例如，可以使用参数化的存储过程来访问数据，以便利用输入字段的强类型检查所带来的好处。

应该检查字符串字段的长度，在许多情况下，还应检查字符串的格式是否正确。例如，邮政编码和身份证号码等都具有明确定义的格式，可以使用常规表达式进行验 证。严格的检查不仅是很好的编程习惯，还能让攻击者更难利用您的代码。攻击者可能会通过类型检查，但长度检查会加大攻击者实施其所喜欢的攻击方式的难度。

拒绝已知的有害输入

虽然不能完全依赖于这种方法,但还是应该拒绝“有害”数据。此方法通常不会像使用上述的“允许”方法那样有效，但二者结合使用可以收到最佳效果。要拒绝有害 数据，需假定应用程序知道恶意输入的所有变体。请记住，字符有多种表达方式。这是“允许”方法成为首选方法的另一个原因。

虽然在应用程序已经部署、不能再做重大更改时，“拒绝”方法非常有用，但它不如“允许”方法那样可靠，因为有害数据（如可用于识别常见攻击的样式）不是保持不变的。有效数据的形式是保持不变的，但有害数据的范围却是时常变化的。

净化输入

净化是为了使有潜在危害的数据变得安全。如果所允许的输入范围不能保证输入数据的安全性，那么净化输入是非常有用的。净化包括从删除用户输入字符串后面的空格到去除值（以便按照文字格式处理该数据）等一切行为。

在 Web 应用程序中，另一个常见的净化输入示例是使用 URL 编码或 HTML 编码来包装数据，并将其作为文本而不是可执行脚本来处理。HtmlEncode 方法去除 HTML 字符，而 UrlEncode 方法对 URL 进行编码，使其成为有效的 URI 请求。

在实践中以下是使用上述方法处理常见输入字段的几个示例:

姓氏字段:这是一个很好的应用限制输入的示例。在这种情况下，可以接受的字符串范围为 ASCII A – Z 和 a – z，以及连字符和波浪线（在 SQL 中，波浪线没有意义），以便处理类似 O'Dell 之类的姓氏。还应限制输入内容的最大长度

数量字段:这是应用输入限制的另一个例子。在此示例中，可以使用简单的类型和范围限制。例如，输入数据应该是介于 0 和 1000 之间的正整数

自定义文本字段:示例包括留言版上的备注字段。在这种情况下，您可能允许输入字母和空格，以及省略符号、逗号和连字符等常用字符。允许输入的字符集只包括符号、括号和大括号

有些应用程序可能允许用户使用一组有限的脚本字符修饰文本，如粗体“ <b> ”、斜体“ <b> ”，甚至包含指向他们所喜爱的 URL 的连接。处理 URL 时，验证时应先对所输入的值进行编码，以便将其作为 URL 处理

不验证用户输入的现有 Web 应用程序。 在理想方案中，应用程序将检查每个字段和入口点的输入内容是否可以接受。然而，如果现有 Web 应用程序不验证用户输入，则需要一种权宜方法来降低风险，直到改进应用程序的输入验证策略。以下两种方法都不能确保输入数据的安全处理，因为这要依赖于输 入的来源，以及应用程序使用输入数据的方式，目前，它们作为快速的补救措施，能在短期内提高应用程序的安全性

向客户端写回数据时，对用户输入的数据进行 HTML 编码和 URL 编码。在这种情况下，假设所有输入均未作为 HTML 处，并且向客户端写回的所有输出都包含在受保护的表单中。这是净化操作在起作用

拒绝恶意脚本字符。这是一个拒绝已知有害输入的示例。在这种情况下，将使用可配置的恶意字符集来拒绝输入。如上所述，这种方法存在的问题是，有害数据是与上下文相关的。

7.2 如何处理输入

7.2.1 处理用户的输入

很多针对 web 应用程序的攻击都涉及到提交未预期的输入，它导致了该应用程序设计者没有料到的行为。因此，对于应用程序安全性防护的一个关键的要求是它必 须以一个安全的方式处理用户的输入。基于输入的漏洞可能出现在一个应用程序的功能的任何地方，并与每上通常使用的技术类型相关。对于这种攻击，输入验证是 常用的必要防护。不存在通用的单一的防护机制。

各种类型的输入

一个典型的 web 应用程序在不同范围形式内处理用户提供的数据。某种类型的输入验证可能对于所有这些输入形式是不可行的。

在许多情况下，一个应用程序对于特定的输入项能够实施非常严格的验证检查。比如提交给登录函数的用户名可以要求最大长度和只能包含字母。

在另外一些情况下，应用程序必须容纳更大范围的可能性的输入。比如提交给一个个人信息页面的地址字段，它可以包含字母、数字、空格、连接符，撇号以及其它字符。对于这种类型，仍然需要加以可行限制，如不能超过合适的长度，以及不能包含 HTML 标记。

在某些情形下，一个应用程序可能需要接受来自用户的任意的输入。比如，一个 blog 应用程序的用户，他创建的博客的主题的 web 应用程序攻击，那么他提交 的内容则可以包含明显的所要讨论的攻击字符串。该应用程序需要以一个安全的方法把这些输入存储在一个数据库中并写到磁盘上，以及回显给用户。所以该应用程 序就不能因为输入看起来有潜在的恶意简单拒绝。

除了来自用户的浏览器的各种输入外，典型的应用程序也接受从服务器到客户端，然后回传给服务器的数据。这些项目包括诸如 cookies 和隐藏的表单字段这 些，它们不会被这个应用程序的普通用户所看到，但是对于攻击者来说是可见和可修改的。在这些情况中，应用程序可以对所接收的数据进行特定的验证。例如要求 参数必须有一个指定数值集中的值，如表明用户所偏爱的语言的 cookie，或以一个指定的格式,如一个客户的 ID 号。进一步说，当应用程序发现返回自用户 的由服务器生成的

数据已经被修改了的话，这通常就表明该用户正在探测应用程序的漏洞。在此类情况下，应用程序应该拒绝请求并记录下这个探测事件。

7.2.2 处理输入的方式

处理用户的输入有很多方式.不同的方式适合不同的情形和不同的输入类型,有些时候一个组合的方式是可取的.

黑名单

这种方式通常使用一个黑名单,它包含已知的被用在攻击方面的一套字面上的字符串或模式.验证机制阻挡任何匹配黑名单的数据.

一般来说,这种方式是被认为对于检查用户的输入效果最差的一种方式.主要有两个原因,首先是,web 应用程序中的一个典型的漏洞可以使用很多种不同的输入 来被利用,输入可以是被加密的或以各种不同的方法表示.其二,漏洞利用的技术是在不断地改进的.有关利用已存在的漏洞类型的新的方法不可能被当前黑名单阻挡.

白名单

这种方式采用一个白名单,它包含一套字面上的字符串或模式,或一套标准,它们用来匹配符合要求的输入.这种检查机制允许匹配白名单的数据,阻止之外的任何 数据.这种方式虽然最有效,但不是通用的,比如撇号和连字符可以被用于对数据库的攻击,但是有时应用程序却应该允许它的输入.

过滤

这种方式下,潜在的恶意字符被删除,留下安全的字符,或者在进一步处理被执行之前,它们被适当地加密或去掉.

基于数据过滤的方式通常是十分有效的,并且在许多情形中,可作为处理恶意输入的通用解决方案.比如,针对跨站脚本攻击的通常的防护是在字符被嵌入到应用程 序的页面之前进行 HTML 加密.然而如果几种潜在的恶意数据在一个输入项中话,有效的过滤是困难的.在这种情况下,边界检查方法则是更适用的.

安全地处理数据

非常多的 web 应用程序漏洞的出现是因为用户提供的数据是以不安全的方法被处理的.在一些情况下,存在安全的编程方法能够避免通常的问题.例如,SQL 注入攻击能够通过正确的参数查询被阻止.在另外的情况中,应用程序功能设计的方法存在内在的不安全性,比如把用户的输入传递给操作系统的命令解释器.

安全处理数据的方式不能适用于 web 应用程序需要执行的每种工作,但是它对于处理潜在的恶意输入是通常有效的方式.

语义检查

语义检查用于防止各种变形数据的输入，这些数据的内容被精心制作来干扰应用程序的处理。然而对于有一些漏洞,攻击者的输入在表面看来和普通用户的输入是一 样的,说到底检查就失去了作用。例如一个攻击者可能会通过改变隐藏的表意字段中的帐号来试图获得对他人银行帐户的访问。要阻止这种未授权的访问，应用程序 需要验证帐号是否属于该用户。

边界检查

对于 web 应用程序,核心安全问题的出现是因为接受自用户的数据是不可信任的.尽管在客户端实现的输入验证检查能够能够提高性能和用户体验,但是这对于实 际到达服务器的数据却没有任何担保.

用户的数据在被服务端应用程序第一时间接受到的一刻就是边界,此时应用程序需要采取步骤来防卫恶意的输入.鉴于核心问题的本质,对于互联网边界间的输入检查这一问题的思考是很有意义的.哪

些是"恶意的"或不可信任的,以及服务端应用程序哪些是"好的"或可信任的.我们给出一个简单的想法:输入验证的角色是清除到达的数据中的潜在的恶意数据,然后把干净的数据传递给可信任的应用程序.据此,这些数据可以被信任和处理而不做进一步的检查或考虑可能的攻击.

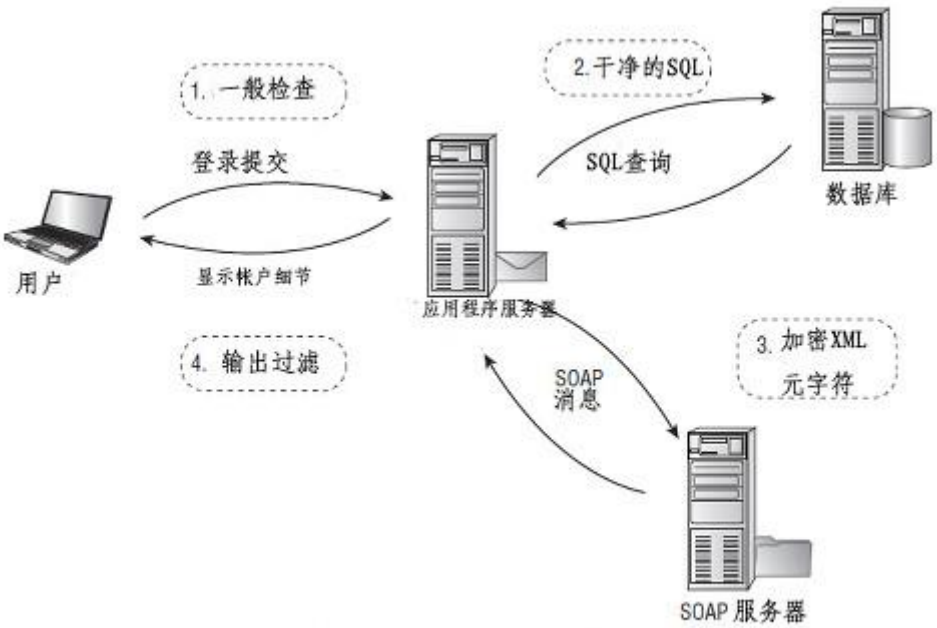
当我们开始检查一些实际的漏洞的时候,会很明显地发现上面的简单的想法是不充分的.原因如下:

(1).鉴于 web 应用程序实现的功能的广泛性,以及所应用的技术的不同,一个典型的 web 应用程序需要防卫大量的不同的基于输入的攻击.每种输入攻击都可能采用了一套不同的数据.针对外部边界仅设计单一的一个机制来防卫所有这些攻击是非常困难的。

(2).许多应用程序的函数包含相互牵连的一系列不同的处理,单个用户所提交的一块数据可能导致不同组件之间的许多操作,上一个的输出可能作为下一个的输 入.当数据被传送时,它可能有所变化,与最初的输入可能有所不同,这样一个熟练的黑客可能能够操纵这个应用程序以在处理的关键阶段导致恶意输入的产生,也 就是攻击接受该数据的组件.这对于在外部边界预见用户输入的每块数据的处理的所有结果来实现一个验证机制是十分困难的。

(3).防卫不同种类的输入攻击可能需要对用户的输入执行不同的验证检查,这些验证检查是不兼容的。例如阻止跨站脚本攻击可以要求 HTML 加密“>”为"&gt";而阻止命令注入攻击(command inject)可能需要阻止包含&和;字符的输入。试图在应用程序的外部边界同时阻止所有种 类的攻击有时是不可能的。

一个使用边界检查概念的更有效的模型是，服务器端的每个组件或功能单元把它的输入当作是来自一个潜在的恶意源。数据检查除了在客户端和服务端之间的边界 外，也在这些认为可信的边界被执行。这个模型对前面列出的问题列表提供了一个解决方案。每个组件针对自身可能的漏洞的特定的输入攻击能够自我防卫。当数据 在不同的组件间传递时，验证检查就可以对前面传来数据进行检查，由于不同有验证检查是在不同的处理阶段被执行的，所以他们之间不会产生冲突。下图演示了一 个防卫恶意输入最有效的方式，用户登录致使对用户提交的输入进行了几步处理，并且每步上都执行了适当的检查。



图示:在多个处理阶段使用边界检查

a) 应用程序接受用户的登录的详细数据.表单处理检查输入的每一项,包括允许的字符、长度是否在指定的范围内、以及不能包含任何已知的攻击特征码.

b) 应用程序执行一个 SQL 查询来验证用户的证书.为了阻止 SQL 注入攻击,用户输入的任何可能攻击数据库的字符在构造查询之前都被去掉.

c) 如果登录成功,应用程序将把来自用户的数据传递给一个 SOAP 服务器以检索他的帐户的更多的信息.为了阻止 SOAP 注入攻击,用户数据中的任何 XML 元字符都被适当地加密处理.

d) 应用程序把用户的账户信息回传给用户的浏览器以显示.为了防止跨站脚本攻击,应用程序 HTML 加密嵌在返回的页面中的用户提供的任何数据.

总之,所有牵涉的组件间都应作边界检查.情况的变化会导致所涉及的组件发生变化.例如如果登录失败后,应用程序会发送一个警告邮件给该用户的话,那么任何合并到该邮件中的用户数据可能需要针对 SMTP 注入攻击作检查。

8.输出编码

8.1 输出的种类

输出编码是转换输入数据为输出格式的过程序，输出格式不包含，或者只是有选择性的包含允许的特殊字符。

输出的种类有:

1）支持 HTML 代码的输出

2）不支技 HTML 代码的输出

3）URL 的输出

4）页面内容的输出 Keywords、Description 等）

5）js 脚本的输出

6）style 样式的输出

7）xml 数据的输出

8）服务控件的输出

8.2 输出编码的必要性

输出编码能有效地防止 HTML 注入(跨站脚本 XSS 攻击)等，也能确保输出内容的完整性和正确性。

8.3 防御

防御手段一：过滤

保护级别：★★★☆

描述:

对于支持 HTML 代码的输出，输出前要确保代码中不含有跨站攻击脚本才能输出。通过编写过滤函数，进行强制过滤。

优点：支持 HTML，有交防止主流 XSS 攻击。
缺点:有可能出错，函数设计难度大。

防御手段二：HTML 编码

描述:
对于不支持 HTML 的输出，在输出到页面前要进行 Server.HtmlEncode 编码，部分服务器控件或者 XSLT 转换本身就支持 Server.HtmlEncode 编码，可不必进行重复编码。
优点：非常可靠。
缺点：不支持 HTML 输出。

防御手段三：URL 编码

保护级别：★★★★★
描述:
对于 URL 的输出，要对输出 URL 进行 Server.UrlEncode 处理。
如： 的输出
要确保输出内容的 url 编码正确,不允许 ” ” ” 的输出。
优点：可靠性高。
缺点：应用范围少。

防御手段四：转换特殊符号的编码形式

描述:
对于页面内容的输出，要确保输出的正确性和允许输出的数据。
如：页面 <meta content="输出内容" name="Keywords" /> 的输出。
要确保输出内容中不包含特殊符号 ” ” ” 输出前要转换特殊符号的编码形式，
将 ” ” ” 转为 " 同样的输出有 title="" 的输出等。
对于 js 脚本的输出，要确保输出代码中不包含跨站脚本，注意 ” ’ ” 和 ” ” ” 的输出，
以免被组合成危险的 js 代码。
如：图片模块，图片地址的输出。
对于 style 样式的输出，要确保样式的正确性，目前系统主要应用到标题颜色的输出，
如果增加其他样式的输出，要确保样式的安全性才能输出。
对于 xml 数据的输出，要确保数据中是否有 XML 不允许的字符，要对特殊字符进行转换才能输出。目前系统中还存在一些地方存在这样的问题。如：标 的数据源，输出时
没有对特殊符号进行处理，造成输出出错。留言标题中出现 “ < ” 等。
优点：防止因殊符号而出现错误，或跨站。
缺点：检查难度大。

防御手段五：其他要注意的地方
描述:

对于服务器控件的输出，要注意输出的环境，对于不同的输出环境进行不同的处理，
如 url 编码，html 编码等。

除上述输出外，还有一些特殊的输出形式，应尽量避免使用，或者处理编码后再使用。如：
表格字段。

9.跨站脚本攻击（XSS）

9.1 定义

什么是跨站脚本攻击

跨站脚本攻击(通常简写为 XSS)是最普遍的 web 应用安全漏洞，当应用程序在发送给浏览器的页面中包含用户提供的数据，没有经过严格验证或转义，那么攻击者就有可能利用网站程序对用户输入过滤不严，输入可以显示在页面上对其他用户造成影响的 HTML 代码，从而盗取用户资料、利用用户身份进行某种动作或者对访问者进行病毒侵害的一种攻击方式。

9.2 危害

- 敏感数据被获取(cookie 盗取)
- 网络钓鱼
- 获取 web 用户的网页内容
- Session Riding(CSRF 攻击)
- 获取用户的键盘击键数据
- web 僵尸
- XSS 蠕虫

攻击者能在受害者浏览器中执行脚本以劫持用户会话、迫害网站、插入恶意内容、重定向用户、使用恶意软件劫持用户浏览器等等。

入侵者便通过技术手段在某个页面里插入一个恶意 HTML 代码，例如记录论坛保存的用户信息（Cookie），由于 Cookie 保存了完整的用户名和密码资料，用户就会遭受安全损失。如这句简单的 javascript 脚本就能轻易获取用户信息：alert(document.cookie)，它会弹出一个包含用户信息的消息框。入侵者运用脚本就能把用户信息发送到他们自己的记录页面中，稍做分析便获取了用户的敏感信息。

跨站脚本攻击的危险，在如今 WEB 安全越来越得到重视，他的危险性也越来越大。

有效防止跨站脚本攻击，是WEB程序是否安全的一个重要标准。

9.3 解决方法

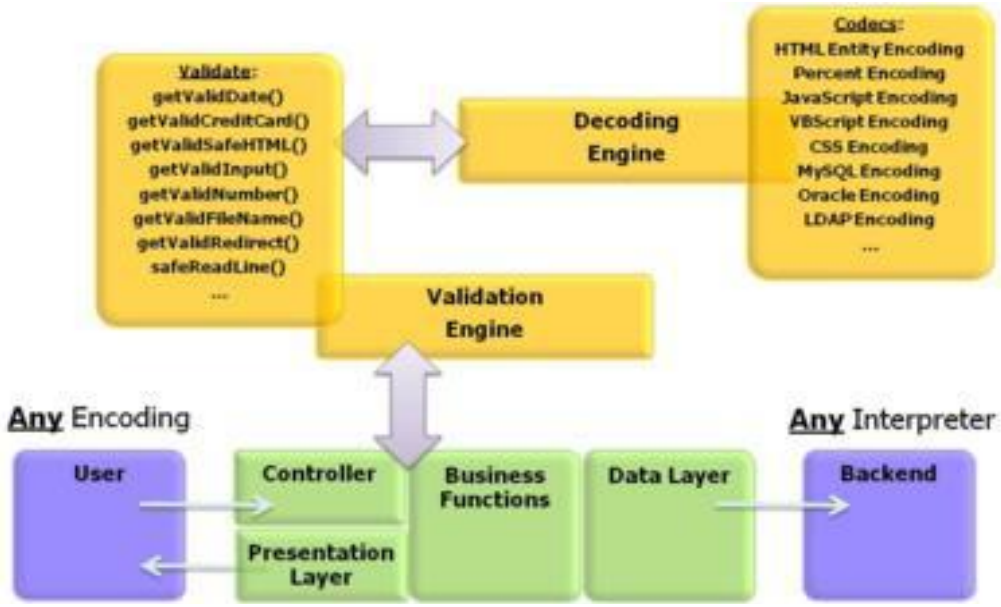
如何防止跨站脚本攻击

主要防御方式

9.3.1 验证输入

验证输入很简单，检查每个输入的有效性。这可能意味着很多东西，但在典型的和简单的情况下，这意味着检查输入类型和数据的长度。例如，如果你是从一个文本框接受一个准的邮政编码，你会知道，唯一有效的类型是一个数字（0-9），而长度应该是6，不能多也不能少。并非所有的案例都如此简单，但很多是相似的。

下图显示验证输入的架构。这里的关键是，一切都进行验证，所有的输入，这并不来自于应用程序（包括用户输入，请求头，Cookie，数据库数据...）。



9.3.2.编码输出

对于不支持HTML代码的地方，可用编码输出。如：Server.UrlEncode等方法编码输出。

优点：安全可靠。

缺点：不支持HTML代码。

对验证输入的另一面就是编码输出。编码输出，是用来确保字符被视为数据，而不是作为HTML元字符被浏览器解析。这些技术定义一些特殊的"转义"字符。没有正确转义的数据它仍然会在浏览器中正确解析。编码输出只是让浏览器知道数据是不是要被解析，达到攻击无法实现的目的。

需要编码的部分：

1、HTML实体

2、HTML属性

3、Javascript

4、CSS

5、URL

9.3.3 辅助防御方式

防御手段一: iframe security="restricted"

保护级别：★★★★
描述：
通过设置 iframe security="restricted",能有效防止 iframe 类的攻击(对 IE 有效).
优点：有效防止 iframe 的攻击。

防御手段二: HttpOnly

保护级别：★★★★
描述：
设置 Cookie 的 HttpOnly 属性，有效地防止 Cookie 通过脚本泄密 （IE6 SP1 以上、Firefox 3）。
优点：有效保护了用户 Cookie 信息。
应用举例：
系统中，所有登陆验证的地方，验证成功后设置 authCookie.HttpOnly = true，设置 Cookie 的 HttpOnly 属性，这些都应用于用户登陆成功的地方。

防御手段三: 字符过滤

保护级别：★★★★
描述：
通过函数进行过滤，能有效防止常见脚站脚本的跨站攻击。主要过滤常见恶意脚本代码，
如：
<applet|meta |xml |blink|link|style|script|embed|object|iframe|frame |frameset|ilayer |layer|bgsound |title|base>
OnX 事件代码、Javascript、Vbscript 和 Style 中的 expression、behaviour、script、position 等。
但过滤可能存在不完全的情况。建立自己的 XSS 攻击库，方便测试和收集新的攻击方式，使过滤函数更加完善。
优点：支持 HTML，有效防止大部份攻击代码。
缺点：可能存在过滤不全的情况。

XSS 漏洞另一个攻击趋势

1）**攻击的本质**:
实际上这是一小段 JAVASCRIPT，我们只是通过漏洞把这段 JS 感染到每一个用户的浏览器，但是他

不再受系统的限制，任何一个有漏洞的浏览器访问了类似页面都会受到攻击。

2）**传播的途径**：

从传播方式上来说它和传统的网页木马攻击方式没有区别，由于这种攻击对于 HTTP 请求包括 AJAX 都没有域的限制，能使用浏览器本地保存的 COOKIE，所以可以劫持用户所有 WEB 应用的身份，它完全能够让已感染主机配合任何网站的应用做蠕虫式的传播。

3）**攻击的危害**：

我们可以像传统的僵尸网络一样控制大量的浏览器肉鸡，控制浏览器做任意的访问行为和动作。同时也能针对单个用户做渗透攻击，劫持他所有 WEB 应用的身份，读取运行本地的任意敏感文件。

4）**攻击的展望**：

当 Active X 等溢出漏洞不再风光的时候，以后利用 XSS 漏洞针对浏览器进行劫持攻击将是一个大的趋势。

10. SQL 注入

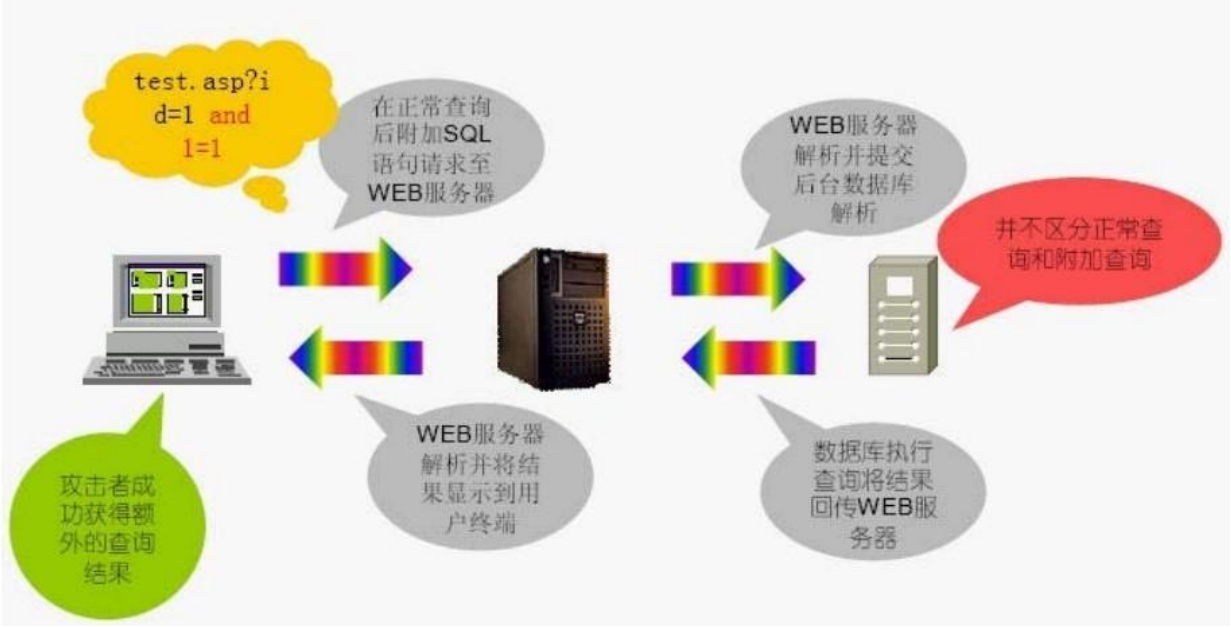
10.1 定义

什么是 SQL 注入

所谓 SQL 注入，就是通过把 SQL 命令插入到 Web 表单递交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的 SQL 命令。通过递交参数构造巧妙的 SQL 语句，从而成功获取想要的数据。

简单来说，注入往往是应用程序缺少对输入进行安全性检查所引起的，攻击者把一些包含指令的数据发送给解释器，解释器会把收到的数据转换成指令执行，注入漏洞十分普遍，通常能在SQL查询、程序参数等中出现。

下图为SQL攻击原理图：



10.2 危害

注入能导致数据丢失或数据破坏、缺乏可审计性或是拒绝服务。注入漏洞有时甚至能导致完全接管主机，主要危害有以下几点:

- ✧ 绕过防火墙进行攻击
- ✧ 绕过 web 应用程序的验证过程
- ✧ 非法越权操作数据库内容
- ✧ 随意篡改网页内容
- ✧ 添加系统帐户或数据库帐户
- ✧ 上传和下载非法文件
- ✧ 本地溢出并获取系统最高权限
- ✧ 安装木马后门/僵尸网络

10.3 解决方法

SQL注入实例:

String sqlString="SELECT * FROM users WHERE fullname='"+form.getFullName()+"'AND

password='"+form.getPassword()+"'";

正常：username=tony，password=123456

SELECT * FROM users WHERE username=tony' AND password='123456'

攻击：username=tony，password='OR'1'='1

SELECT * FROM users WHERE username=tony'ANDpassword='' OR '1'='1'

参数化查询预处理

对于JDBC而言， SQL注入攻击只对Statement有效，对PreparedStatement是无效的，这是因为PreparedStatement 不允许在不同的插入时间改变查询的逻辑结构。

如验证用户是否存在的SQL语句为 ：

select count(*) from usertable where name='用 户 名 ' and pswd='密 码 '

如果在用户名字段中 输入 ' or '1'='1' or '1'='1

或是在密码字段中输入 1' or '1'='1

将绕过验证 ，但这种手段只对Statement有效，对PreparedStatement无效

PreparedStatement 相对 Statement有以下 优 点:

- ✓ 防注入 攻 击
- ✓ 多次运行速度快
- ✓ 防止数据库缓冲区溢出
- ✓ 代码的可读性可维护性好

11.恶意文件执行

11.1 定义

恶意文件执行是一种能够威胁任何网站形式的漏洞，只要攻击者在具有引入（include）功能程式的参数中修改参数内容，WEB服务器便会引入恶意程序内容从而受到恶意文件执行漏洞攻击。

11.2 危害

攻击者可利用恶意文件执行漏洞进行攻击取得WEB服务器控制权，进行不法利益或获取经济利益。

11.3 解决方法

- ➢ **验证输入，验证上传文件名**

- 检查上传文件大小

12.不安全的直接对象引用

12.1 定义

所谓"不安全的对象直接引用"，即Insecure direct objectreferences，意指一个已经授权的用户，通过更改访问时的一个参数，从而访问到原本其并没有得到授权的对象。Web应用往往在生成Web页面时会用它的真实名字，且并不会对所有的目对象访问时来检查用户权限，所以这就造成不安全的对象直接引用的漏洞。

我们看如下的一个示例，也许这样就更容易理解什么是不安全的对象直接引用。

- 攻击者发现他自己的参数是6065，即?acct=6065;
- 他可以直接更改参数为6066，即?acct=6066;
- 这样他就可以直接看到6066用户的账户信息。

12.2 危害

这种漏洞能损害参数所引用的所有数据。除非名字空间很稀疏，否则攻击者很容易访问该类型的所有数据。

12.3 解决方法

检查访问。来自不受信源所使用的所有直接对象引用都必须包含访问控制检测，这样才能确保用户对要求的对象有访问权限

13.信息泄露和错处理不当

13.1 定义

应用程序常常产生错误信息并显示给使用者。很多时候，这些错误信息是非常有用的攻击，因为它

们揭示实施细则或有用的开发信息利用的漏洞。

13.2 危害

- 泄露太多的细节（如错误堆栈跟踪信息、SQL语句等等）；
- 登录失败后，通知用户是否用户ID或密码出错——登录失败可能是由于ID或密码错误造成的。这为一个对关键资产发动蛮力攻击的攻击者提供重要信息。

13.3 解决方法

案例1

通过web.xml配置文件实现

```
<error-page>
<exception-type>java.lang.Throwable</exception-type>
<location>/error.jsp</location>
</error-page>
```

案例2

针对登录尝试的攻击，可以使用相同的报错信息，比如都是提示"输入的用户名或者密码错误！"。

14.残缺的认证和会话管理

14.1 定义

与认证和会话管理相关的应用程序功能往往得不到正确实施，这就导致攻击者破坏密码、密匙、会话令牌或利用实施漏洞冒充其他用户身份。

14.2 危害

这些漏洞可能导致部分甚至全部帐户遭受攻击。一旦攻击成功，攻击者能执行合法用户的任何操作。因此特权帐户会造成更大的破坏。

14.3 解决方法

- 使用内置的会话管理功能。
- 通过认证的问候:
- 使用单一的入口点。
- 确保在一开始登录SSL保护的网页。
- 获取注销的权利;
- 添加超时;
- 确保你使用的是安全相关的功能;
- 使用强大的认证;
- 不进行默认身份验证

15.不安全的通信

15.1 定义

对于不加密的应用程序的网络信息传输，需要保护敏感的通信。加密（通常SSL）的，必须用于所有身份验证的连接，特别是通过Internet访问的网页，以及后端的连接。否则，应用程序将暴露身份验证或会话令牌。

15.2 危害

- 攻击者能够取得或是篡改机密的或是私有的信息;
- 攻击者通过这些私密信息的窃取从而进行进一步的攻击;
- 造成企业形象破损，用户满意度下降，甚至会有法律诉讼等。

15.3 解决方法

- 提供合理的保护机制

- 对于敏感数据的传输，对所有连接都要使用TLS;
- 在传输前对单个数据都要进行加密；（如XML-Encryption）
- 在传输前对信息进行名；（如XML-Signature）
- 正确的使用这些机制
- 使用准的强算法;
- 合理管理密钥和证书;
- 在使用前验证SSL证书

16.限制 URL 访问失效

16.1 定义

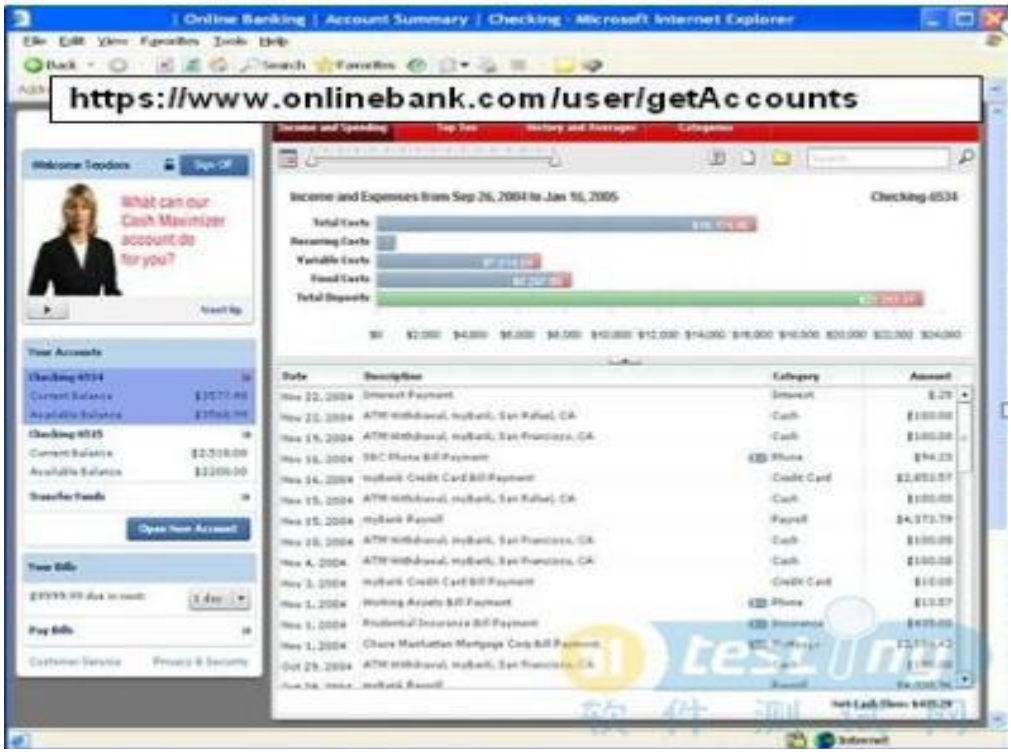
这个漏洞事实上也是与认证相关的，与我们前面提到的不安全的直接对象引用也是类似的，不同在于这个漏洞是说系统已经对URL的访问做限制，但这种限制却实际并没有生效。常见的错误是，我们在用户认证后只显示给用户认证过的页面和菜单选项，而实际上这些仅仅是表示层的访问控制而不能真正生效，攻击者能够很容易的就伪造请求直接访问未被授权的页面。

我们举个例子来说明这个过程:

1、攻击者发现他自己的访问地址为/user/getAccounts;

2、他修改他的目录为/admin/getAccounts或/manager/getAccounts;

3、这样攻击者就能够查看到更多的账户信息。



16.2 解决方法

对每个URL，我们必须做三件事：

- 如果这个URL不是公开的，那么必须限制能够访问他的授权用户
  - ✓ 加强基于用户或角色的访问控制；
  - ✓ 完全禁止访问未被授权的页面类型（如配置文件、日志文件、源文件等）
- 验证你的构架
  - ✓ 在每一个层次都使用简单肯定的模型；
  - ✓ 确保每一层都有一个访问机制
- 验证你的实现
  - ✓ 不要使用自动化的分析工具；
  - ✓ 确保每个URL都被外部过滤器或其他机制保护；
  - ✓ 确保服务器的配置不允许对非授权页面的访问

17.安全审计

17.1.白盒安全审计

安全应贯穿于整个软件生命周期，实现软件开发早期的安全编程，在该过程中需采用有效的手段，对软件代码的安全性进行自始至终的安全审计，及时发现软件开发过程中，由于编码问题而导致的安全隐患，并及时进行整改。

代码白盒审计(SCV)是指采用自动静态分析(ASA)技术扫描某个应用程序的源代码,通过甄别和定位可能存在的薄弱环节数量，以验证该应用程序完整性的校验过程。进行自动静态分析时无须运行或"执行"待分析的软件。这一特点就使自动静态分析技术在查询已知或未知交叉站点脚本(XSS)和SQL注入漏洞时更为高效，同时代码白盒审计过程能与应用软件开发并行也使其为应用程序提出修正意见成为可能，这不仅节约大量研发时间而且节省巨大的成本。

推荐使用专业"白盒代码审计系统"，为软件开发人员自动提供详尽的软件脆弱性的分析、薄弱点发生源的追踪以及如何定位薄弱环节的建议，以帮助软件开发人员更好地理解网络应用程序安全性要求。

17.2.黑盒安全测试

在软件开发后期阶段，对已经成形的软件进行黑盒安全检测，在对整个软件系统内部程序结构完全不解的情况下，输入数据与输出数据的对应关系出发，对目前主流的安全漏洞进行检测和验证，完全模拟用户操作行为，在与软件系统上线使用环境相同的情况下，能够更为客观和全面地发现软件系统自身的安全隐患。

18.日志和监测

做好系统的日志监测，记录好后台管理操作和异常情况，有利于监测系统未知漏洞，通过操作日志，还能找出系统出错或对某些重要操作的管理员、操作时间、IP 等信息。有效地预防，检测,增强系统的安全性。

对重要的日志都进入了记录，如：登陆日志、系统错误日志、数据库错误日志等。

参考资料

[2].http://www.microsoft.com/taiwan/msdn/columns/huang_jhong_cheng/LVSS.htm
[3].http://technet.microsoft.com/zh-cn/library/ms161953.aspx
[4].http://www.microsoft.com/china/technet/security/guidance/secmod94.mspx
[5].http://www.microsoft.com/china/technet/security/guidance/secmod83.mspx
[6]. https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/assembly/639-BSI.html
[7]. https://www.owasp.org/index.php/Main_Page