在PHP中全面阻止SQL注入式攻击 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/167/2021_2022__E5_9C_A8PHP_E4_B8_AD_E5_c103_167248.htm 在本系列文章中，我们将全面探讨如何在PHP开发环境中全面阻止SQL注入式攻击，并给出一个具体的开发示例。 一、 引言 PHP是一种力量强大但相当容易学习的服务器端脚本语言，即使是经验不多的程序员也能够使用它来创建复杂的动态的web站点。然而，它在实现因特网服务的秘密和安全方面却常常存在许多困难。在本系列文章中，我们将向读者介绍进行web开发所必需的安全背景以及PHP特定的知识和代码-你可以借以保护你自己的web应用程序的安全性和一致性。首先，我们简单地回顾一下服务器安全问题-展示你如何存取一个共享宿主环境下的私人信息，使开发者脱离开生产服务器，维持最新的软件，提供加密的频道，并且控制对你的系统的存取。 然后，我们讨论PHP脚本实现中的普遍存在的脆弱性。我们将解释如何保护你的脚本免于SQL注入，防止跨站点脚本化和远程执行，并且阻止对临时文件及会话的"劫持"。 在最后一篇中，我们将实现一个安全的Web应用程序。你将学习如何验证用户身份，授权并跟踪应用程序使用，避免数据损失，安全地执行高风险性的系统命令，并能够安全地使用web服务。无论你是否有足够的PHP安全开发经验，本系列文章都会提供丰富的信息来帮助你构建更为安全的在线应用程序。 二、 什么是SQL注入 如果你打算永远不使用某些数据的话，那么把它们存储于一个数据库是毫无意义的；因为数据库的设计目的是为了方便地存取和操作数据库中的数据。但是，如果只是

简单地这样做则有可能会导致潜在的灾难。这种情况并不主要是因为你自己可能偶然删除数据库中的一切；而是因为，当你试图完成某项"无辜"的任务时，你有可能被某些人所"劫持"-使用他自己的破坏性数据来取代你自己的数据。我们称这种取代为"注入"。 其实，每当你要求用户输入构造一个数据库查询，你是在允许该用户参与构建一个存取数据库服务器的命令。一位友好的用户可能对实现这样的操作感觉很满意；然而，一位恶意的用户将会试图发现一种方法来扭曲该命令，从而导致该被的扭曲命令删除数据，甚至做出更为危险的事情。作为一个程序员，你的任务是寻找一种方法来避免这样的恶意攻击。 三、 SQL注入工作原理 构造一个数据库查询是一个非常直接的过程。典型地，它会遵循如下思路来实现。仅为说明问题，我们将假定你有一个葡萄酒数据库表格"wines"，其中有一个字段为"variety"（即葡萄酒类型）： 1. 提供一个表单-允许用户提交某些要搜索的内容。让我们假定用户选择搜索类型为"lagrein"的葡萄酒。 2. 检索该用户的搜索术语，并且保存它-通过把它赋给一个如下所示的变量来实现： $variety = $_POST[variety]. 因此，变量$variety的值现在为：lagrein 3. 然后，使用该变量在WHERE子句中构造一个数据库查询： $query = "SELECT * FROM wines WHERE variety=$variety". 所以，变量$query的值现在如下所示：SELECT * FROM wines WHERE variety=lagrein 4. 把该查询提交给MySQL服务器。 5. MySQL返回wines表格中的所有记录-其中，字段variety的值为"lagrein"。 到目前为止，这应该是一个你所熟悉的而且是非常轻松的过程。遗憾的是，有时我们所熟悉并感到舒适的过程却容易导致我们产生自满情绪。现在

，让我们再重新分析一下刚才构建的查询。 1. 你创建的这个查询的固定部分以一个单引号结束，你将使用它来描述变量值的开始： $query = " SELECT * FROM wines WHERE variety = ". 2. 使用原有的固定不变的部分与包含用户提交的变量的值： $query .= $variety. 3. 然后，你使用另一个单引号来连接此结果-描述该变量值的结束： $ query .= "". 于是，$query的值如下所示： SELECT * FROM wines WHERE variety = lagrein 这个构造的成功依赖用户的输入。在本文示例中，你正在使用单个单词(也可能是一组单词)来指明一种葡萄酒类型。因此，该查询的构建是无任何问题的，并且结果也会是你所期望的-一个葡萄酒类型为"lagrein"的葡萄酒列表。现在，让我们想象，既然你的用户不是输入一个简单的类型为"lagrein"的葡萄酒类型，而是输入了下列内容(注意包括其中的两个标点符号)： lagrein or 1=1. 现在，你继续使用前面固定的部分来构造你的查询(在此，我们仅显示$query变量的结果值)： SELECT * FROM wines WHERE variety = 然后，你使用包含用户输入内容的变量的值与之进行连接(在此，以粗体显示)： SELECT * FROM wines WHERE variety = lagrein or 1=1. 最后，添加上下面的下引号： SELECT * FROM wines WHERE variety = lagrein or 1=1. 于是，这个查询结果与你的期望会相当不同。事实上，现在你的查询包含的不是一条而是两条指令，因为用户输入的最后的分号已经结束了第一条指令(进行记录选择)从而开始了一条新的指令。在本例中，第二条指令，除了一个简单的单引号之外别无意义；但是，第一条指令也不是你所想实现的。当用户把一个单引号放到他的输入内容的中间时，他结束了期望的变量的值，并且引入了另一个条件。因此，不

再是检索那些variety为"lagrein"的记录，而是在检索那些满足两个标准中任何一个（第一个是你的，而第二个是他的-variety为"lagrein"或1等于1）的记录。既然1总是1，因此，你会检索到所有的记录！ 你可能反对：我不会使用双引号来代替单引号来描述用户提交的变量吗？不错，这至少可以减慢恶意用户的攻击。(在以前的文章中，我们提醒过你：应该禁止所有对用户的错误通知信息。如果在此生成一条错误消息，那么，它有可能恰恰帮助了攻击者-提供一个关于他的攻击为什么失败的具体的解释。) 在实践中，使你的用户能够看到所有的记录而不只是其中的一部分乍看起来似乎不太费事，但实际上，这的确费事不少；看到所有的记录能够很容易地向他提供有关于该表格的内部结构，从而也就向他提供了使其以后实现更为恶毒目的的一个重要参考。如果你的数据库中不是包含显然无害的酒之类信息而是包含例如一个含有雇员年收入的列表，那么，刚才描述情形会是特别真实的。而从理论角度分析，这种攻击也的确是一件很可怕的事情。由于把意外的内容注入到你的查询中，所以，此用户能够实现把你的数据库存取转化为用于实现他自己的目的。因此现在，你的数据库已经对他打开-正如对你敞开一样。 四、PHP和MySQL注入 如我们前面所描述的，PHP，从本身设计来说，并没有做什么特别的事情-除了按照你的指示操作之外。因此，如果为恶意用户所用，它也只是按照要求"允许"特别设计的攻击-例如我们前面所描述的那样。 我们将假定，你不会故意地或甚至是偶然地构造一个具有破坏性效果的数据库查询-于是，我们假定问题出在来自你的用户的输入方面。现在，让我们来更为细致地分析一下用户可能向你的脚本提

供信息的各种途径。 五、 用户输入的类型 如今，用户能够影响你的脚本的行为已变得越来越复杂。 用户输入最明显的来源当然是表单上的一个文本输入域。使用这样的一个域，你简直是在故意教唆一个用户输入任意数据。而且，你向用户提供了一个很大的输入范围；没有什么办法能够使你提前限制一个用户能够输入的数据类型(尽管你能够选择限制它的长度)。这正是绝大多数的注入式攻击源主要来自于无防备的表单域的原因。 但是，还存在其它的攻击源，并且稍加思考你就会想到的一种潜于表单后台的技术-POST方法！通过简单地分析显示在浏览器的导航工具栏中的URI，一个善于观察的用户能够很容易地看出是什么信息传递到了一个脚本。尽管典型情况下这样的URI是以编程方式生成的，但是，没有什么办法能够阻止一个恶意的用户简单地把一个带有一个不适当的变量值的URI输入到一个浏览器中-而这样潜在地打开一个可能会被其滥用的数据库。 限制用户输入内容的一个常用策略是在一个表单中提供一个选择框，而不是一个输入框。这种控件能够强制用户从一组预定义的值中进行选择，并且能够在一定程度上阻止用户输入期望不到的内容。但是正如一个攻击者可能"哄骗"一个URI(也即是，创建一个能够模仿一个可信任的却无效的URI)一样，他也可能模仿创建你的表单及其自己的版本，并因此在选项框中使用非法的而不是预定义的安全选择。要实现这点是极其简单的；他仅需要观察源码，然后剪切并且粘贴该表单的源代码-然后一切为他敞开大门。 在修改该选择之后，他就能够提交表单，并且他的无效的指令就会被接受，就象它们是原始的指令一样。因此，该用户可以使用许多不同的方法试图把恶意的代码注入到

一个脚本中。 100Test 下载频道开通，各类考试题目直接下载。详细请访问 www.100test.com